PiSPICE: Accelerating Post-Layout SPICE Simulation via Essential Parasitic Identification

Zhou Jin[1], Jing Li[2], Jian Xin[3], Tianjia Zhou[3], Xiao Wu[4], Dan Niu[5], and Zuochang Ye[3]
[1]College of Integrated Circuits, Zhejiang University, China, [2]SSSLab, China University of Petroleum-Beijing, China
[3]Tsinghua University, China, [4]Huada Empyrean Software Co. Ltd, China, [5]School of Automation, Southeast University, China
Email: z.jin@zju.edu.cn

***Abstract*—As process nodes scale to more advanced technologies, post-layout simulations for integrated circuits have become increasingly complex, involving billions to trillions of nodes. The growing design complexity and transistor integration require more accurate and efficient post-layout SPICE simulations. However, existing methods for solving large-scale post-layout circuits face significant challenges due to high computational costs. In this paper, we propose a new approach, PiSPICE, which utilizes adjoint sensitivity analysis to identify critical parasitics and eliminate non-critical ones, effectively reducing the simulation scale and improving speed. By modeling parasitics and performing sensitivity analysis on pre-layout circuits, we significantly reduce the computational burden and avoid the overhead of directly analyzing sensitivities in large-scale post-layout circuits. By retaining only the critical parasitics and applying model order reduction to minimize their impact, while eliminating non-critical parasitics, PiSPICE achieves a speedup of up to 17.27x in simulation with an error margin of less than 0.78% compared to the commercial simulator Spectre.**

***Index Terms*—Post-layout SPICE simulation, Parasitic modeling, Sensitivity analysis, Essential parasitic identification, Model order reduction**

I. Introduction

As semiconductor technologies advance towards increasingly smaller process nodes, the complexity of integrated circuits (ICs) has escalated dramatically. At these small process nodes, ICs are composed of billions to trillions of transistors, and layout-induced parasitics (e.g. resistance, capacitance) become increasingly significant. These parasitics affect the circuit's performance, particularly in terms of signal integrity, power consumption, and timing, and must be accurately modeled and included in post-layout simulations [1]–[11]. However, with the increasing transistor integration and shrinking design margins, traditional post-layout SPICE simulations that account for parasitic effects are becoming computationally prohibitive, both in terms of memory and processing time.

The computational burden is primarily due to the large-scale matrix operations involved in simulating complex circuits, which often require solving sparse, high-dimensional systems of equations, particularly when dealing with millions to billions of nodes [12], [13]. To address these challenges, various techniques have been proposed. One common approach is the domain decomposition method (DDM), which divides the circuit into smaller subproblems and constructs a bordered block diagonal matrix. This enables parallel processing and simplifies the solution [14]–[18]. However, DDM faces challenges when solving for coupling nodes, i.e., the connections between subcircuits, which create a global communication bottleneck. As the number of partitions increases, this bottleneck becomes a critical limiting factor, hindering the scalability of the method.

Another widely explored technique is model order reduction (MOR), which simplifies the system by approximating it with a lower-order model that retains the dominant behavior of the circuit [19]–[25]. However, MOR methods often fail to provide sufficient accuracy when dealing with highly complex and parasitic-rich circuits in advanced technologies. Additionally, all existing methods overlook the fact that not all nodes in a circuit are equally sensitive to parasitic parameters, meaning that parasitics only significantly affect the post-layout performance of certain nodes.

In light of these challenges, we present a new post-layout simulation acceleration framework, PiSPICE, which employs sensitivity analysis to identify and retain only the most critical parasitic elements while discarding non-critical ones. PiSPICE leverages pre-layout netlists to perform parasitic modeling and adjoint sensitivity analysis. This approach avoids the computational overhead of analyzing sensitivities directly on large-scale post-layout circuits, which can be time-consuming and resource-intensive. PiSPICE identifies the essential nodes in the pre-layout circuit that are most sensitive to parasitic effects and focuses on retaining the parasitic sub-networks around these nodes. Additionally, MOR techniques are applied to minimize the impact of retained parasitics, reducing the dimensionality of the problem and optimizing the overall simulation process.

Our contributions are summarized as follows.

- To the best of our knowledge, this is the first work to accelerate post-layout simulation by identifying the significance of parasitics on circuit performance, thereby significantly reducing the simulation cost.
- Adjoint sensitivity analysis is performed on pre-layout circuit to identify essential parasitics, avoiding the costly computations of directly applying it to large post-layout circuits.
- The scale of the post-layout circuit is markedly reduced by collapsing non-critical parasitic sub-networks, and applying an improved PRIMA algorithm to critical parasitic sub-networks.
- PiSPICE is especially effective for circuits with identical

topologies but varying parameters, such as sizing, where sensitivity analysis only needs to be performed once.

• PiSPICE achieves up to 93.77x reduction in circuit scale and 17.27x speedup over Spectre, and up to 43.62x reduction in circuit scale and 9.50x speedup over Spectre with TICER.

II. Background

A. Post-layout SPICE Simulation

Post-layout simulation refers to simulating a circuit after its layout design is complete. During this stage, layout information, including parasitic resistances and capacitances, is extracted and added to the circuit model. This provides a more accurate prediction of how the circuit will behave in real-world conditions once fabricated. It is crucial for verifying the circuit's functionality and performance, addressing factors such as timing analysis, signal integrity (including noise and crosstalk), power consumption, and potential manufacturing variability [26]. Post-layout simulation is essential for design signoff, ensuring the final design meets specifications and performs reliably before fabrication.

B. Adjoint Sensitivity Analysis

Adjoint sensitivity analysis is an efficient technique in circuit simulation for evaluating the impact of circuit parameter variations on performance metrics such as output voltage, current, or power consumption [27], [28]. By solving adjoint equations, it calculates sensitivities to multiple parameters simultaneously with far fewer simulations than traditional methods, i.e., finite difference and direct methods, making it especially effective for large-scale circuits and optimization tasks. Applicable in both the time and frequency domains, this approach combines computational efficiency with accuracy, identifying critical parameters to optimize designs and ensure reliable performance in complex systems [29]–[31].

For a circuit in the form of differential algebraic equations (DAE) [32]:

$$g(x,t,p) = \frac{d}{dt}q(x,p) + f(x,p) + b(t,p) = 0 \tag{1}$$

where x is the state vector, p is the parameter vector, t is time, q means dynamic variable, f means static variable, b means inputs to system. The adjoint sensitivity (i.e., the derivative of the objective function O with respect to the parameter p) is analyzed by:

$$\frac{dO}{dp} = \sum_{i=0}^{n}\sum_{k=i}^{n}\left(\frac{\mathrm{d}O}{\mathrm{d}x}\right)_k\left(\frac{\partial g}{\partial x}\right)_k^{-1}\left(\prod_{j=k-1}^{i}\left(\frac{1}{h_t}\frac{\partial q}{\partial x}\right)_j\left(\frac{\partial g}{\partial x}\right)_j^{-1}\right)\left(\frac{\partial \varphi}{\partial p}\right)_i \tag{2}$$

In recent years, advanced methodologies have been developed to enhance computational efficiency in sensitivity analysis, enabling faster processing while maintaining high accuracy. Notable examples include fast adjoint transient sensitivity analysis for multiple time points objective functions [32], time-domain sensitivity analysis approach that significantly reduces runtime without compromising precision [33], and memory-efficient adjoint sensitivity analysis leveraging compression [34]. These innovations address scalability and complexity challenges, making sensitivity analysis more suitable for contemporary circuit design tasks.

C. Model Order Reduction

Model order reduction aims to reduce the order (size) of the system while maintaining an accurate representation of the circuit's behavior, thereby accelerating simulation time.

TICER [36] is a classical MOR algorithm based on node elimination. It efficiently reduces the size of parasitic networks, ensuring the preservation of circuit stability and passivity, and is relatively simple to implement. As a result, it has been widely adopted by commercial design tools, e.g., Calibre xRC. However, as the circuit scale increases, the model reduced by TICER can become denser, leading to higher computational overhead. Additionally, its performance may be limited in scenarios that require high precision.

PRIMA [35], another MOR algorithm, is based on projection methods. PRIMA is particularly effective for sub-networks, which are typically smaller and exhibit simpler dynamics, making it efficient for capturing their behavior. For an $n \times n$ system of equations describing a network, such as:

$$C\dot{x}(t) = -Gx(t) + Bu(t) \tag{3}$$

where G is the conductance matrix, C is the capacitance matrix, and $x(t)$ is the state vector. PRIMA approximates this system by projecting onto a reduced-order subspace with a transformation matrix R, where

$$\hat{G} = R^T G R \quad \hat{C} = R^T C R \quad \hat{B} = R^T B \tag{4}$$

This reduces the size of the system from $n \times n$ to $m \times m$, where $m < n$. The projection matrix R is typically chosen to capture the most important modes of the system, often using techniques like proper orthogonal decomposition.

D. Motivation

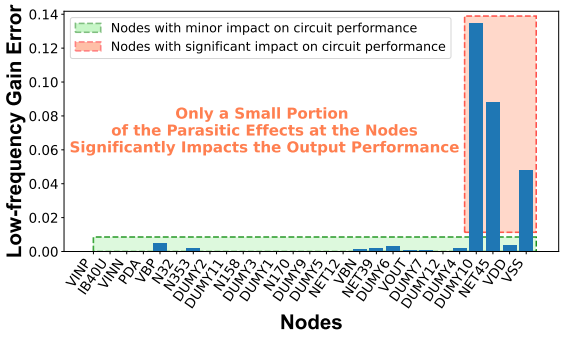


Fig. 1: Low-frequency voltage gain error of an operational amplifier by removing parasitic sub-networks for each node.

This paper is based on a key observation: in most cases, only a small portion of the parasitic effects of pre-layout nodes will significantly affect circuit performance. As illustrated in Figure 1, the majority of parasitic sub-networks have negligible effects on the circuit's output. Therefore, effectively identifying these parasitic-sensitive nodes and eliminating the less influential parasitic networks can substantially reduce the RC scale of post-layout simulations while maintaining accuracy, thereby significantly improving simulation speed.

III. Proposed Method

A. Overall Framework

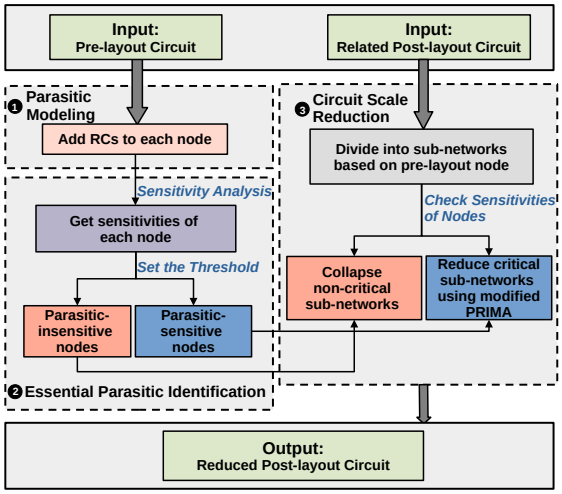


Fig. 2: The workflow of PiSPICE.

In this chapter, we introduce PiSPICE, the proposed post-layout simulation acceleration framework. PiSPICE identifies nodes that are parasitic-sensitive, those whose parasitic sub-networks significantly impact post-layout simulation performance. By retaining only the parasitic networks of these sensitive nodes, PiSPICE reduces the simulation scale and improves post-layout simulation speed. The overall framework is illustrated in Figure 2 and consists of three key components: (1) Parasitic modeling on the pre-layout circuit, (2) Perform an adjoint sensitivity analysis in this modeled circuit to identify parasitic-sensitive pre-layout nodes, (3) Reduce the RC networks according to the sensitivity results.

Performing a sensitivity analysis for every parasitic parameter in the post-layout circuit is computationally expensive. Therefore, instead of analyzing all parasitic parameters in the post-layout circuit, we focus on identifying the nodes in the pre-layout circuit that are parasitic-sensitive and target their parasitic sub-networks for further processing. To achieve this, we first model parasitic parameters using the pre-layout circuit by introducing pseudo RC elements at each circuit node to emulate the corresponding parasitic sub-networks.

Next, we perform an adjoint sensitivity analysis on this pseudo circuit to distinguish between parasitic-sensitive nodes and parasitic-insensitive nodes. Then the extracted parasitic network is partitioned on the basis of pre-layout circuit nodes. According to the results of the sensitivity analysis, insensitive networks are collapsed to significantly reduce their scale, while sensitive networks are retained. Furthermore, an improved MOR algorithm is applied to compress the size of sensitive networks, enhancing simulation efficiency. The entire process is detailed in Algorithm 1.

B. Parasitic Modeling on the Pre-layout Circuit

Parasitic networks typically consist of ground capacitance, coupling capacitance between nodes, and parasitic resistance of interconnects. To accurately model the parasitic sub-network of each circuit node, we take the following approach.

Algorithm 1 The overall framework of proposed PiSPICE.

Input: The pre-layout circuit C_{pre} with N nodes and its related post-layout circuit C_{post}
Output: Reduced post-layout circuits C'_{post}

for $j = 1$ to N **do**
  Add modeled parasitic capacitors $Caps$ and resistors $Ress$ and set there value to $p = 1e-15$
end for
Calculating the sensitivity of $Caps$ and $Ress$
for $j = 1$ to N **do**
  Calculate the mean value of the sensitivity of $Caps$ and $Ress$ to which $Node_j$ is connected
end for
Set the average sensitivity of nodes as the threshold s, and sequentially divide nodes into parasitic-sensitive nodes and parasitic-insensitive nodes
Divide C_{post} into N sub-networks
for $j = 1$ to N **do**
  Collapse the sub-network of parasitic-insensitive nodes and reduce the sub-network of parasitic-sensitive nodes using the improved PRIMA
end for

Pseudo-capacitors are placed between each node and the ground (VSS) to emulate the ground capacitance. Pseudo-capacitors are inserted between each node and all other nodes to simulate the coupling capacitance. Pseudo-resistors are added to the interconnects, with additional circuit nodes introduced to represent the parasitic resistance.

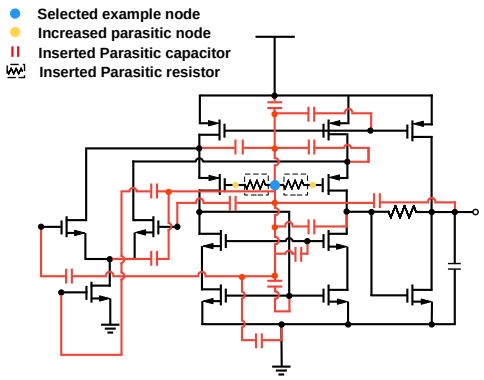


Fig. 3: An example of parasitic modeling for selected blue node. Pseudo-capacitors (red ones) are inserted between it and all other nodes to model coupling parasitic capacitance, and between it and VSS to represent ground parasitic capacitance. Resistors (in dashed box), along with corresponding new nodes (yellow ones), are added to model the parasitic resistance of interconnects.

This setup ensures a comprehensive emulation of the parasitic effects for each node's sub-network, laying the foundation for subsequent sensitivity analysis. An example is shown in Figure 3, where the blue node is demonstrated as an example to illustrate the modeling of its parasitic sub-network.

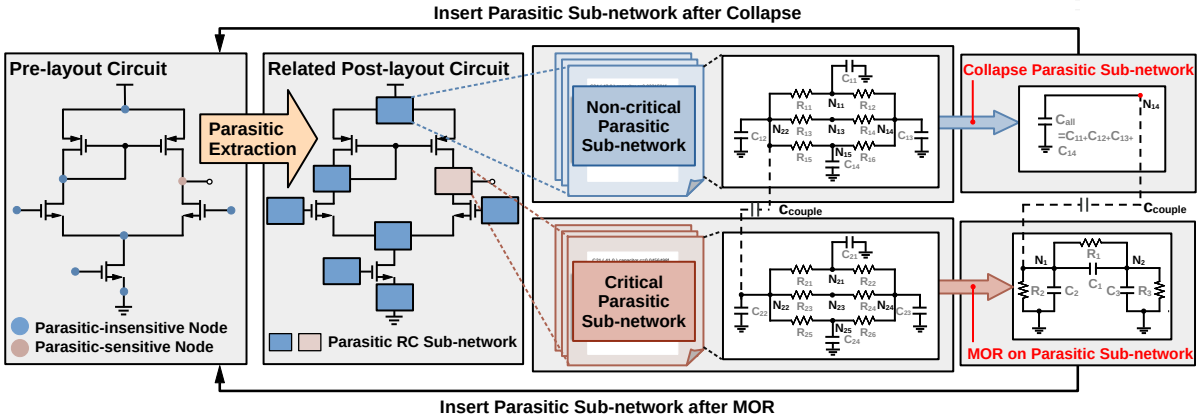


Fig. 4: After identifying parasitic-sensitive nodes, we leverage a hybrid strategy to reduce the simulation scale by handling critical and non-critical parasitic sub-networks separately.

The added parasitic resistors and capacitors are assigned identical values that are effectively close to zero. This deliberate choice ensures that the influence of these parasitic components on circuit performance is determined primarily by their topological placement, rather than their parameter values.

C. Essential Parasitic Identification with Sensitivity Analysis

After constructing the parasitic RC network for each node in pre-layout circuit, a pseudo post-layout circuit is generated. We then perform an adjoint sensitivity to this circuit to identify which nodes are highly sensitive, indicating parasitics that significantly affect the performance of post-layout simulations.

This procedure consists of two steps. First, sensitivity analysis is performed by calculating the derivative of the objective function $f(\mathbf{p})$ (the circuit's performance) with respect to the parameters $\mathbf{p}= p_1, p_2, ..., p_n$ (each parasitic component). This allows us to identify which parasitic components p_i have a significant impact on the circuit's performance, i.e., which components yield large values of $\frac{\partial f}{\partial p_i}$. However, our primary focus is on identifying circuit nodes that are parasitic-sensitive, as this information can assist in subsequent partitioning and reduction of parasitic sub-networks based on the nodes. Therefore, in the second step we calculate the sensitivity S_{node} of each node. It is defined as the average of the total sensitivity of parasitic components p_i connected to the node v_j:

$$S_{\text{node},j} = \frac{1}{|\mathcal{N}_j|} \sum_{i \in \mathcal{N}_j} \left| \frac{\partial f}{\partial p_i} \right| \tag{5}$$

where $\mathcal{N}_j$ is the set of parasitic components connected to node v_j.

To establish a fair threshold for distinguishing between sensitive and insensitive nodes, we compute the mean sensitivity S_{avg} of all nodes by summing up their individual sensitivities and dividing by the total number of nodes.

$$S_{\text{avg}} = \frac{1}{N} \sum_{j=1}^{N} S_{\text{node},j} \tag{6}$$

where N is the total number of nodes. This average sensitivity value S_{avg} serves as the threshold s to classify nodes as sensitive $(S_{\text{node},j} \geq s)$ or insensitive $(S_{\text{node},j} < s)$.

It is worth noting that our method is particularly well-suited for scenarios that involve multiple circuits with the same topology but different parameters, such as the sizing and Monte Carlo analysis. The identification results of essential parasitics can be effectively transferred to similar circuits, further diminishing the perceived overhead of sensitivity analysis. The applicability of our method to these scenarios is demonstrated in the experimental section.

D. Hybrid Strategy to Reduce Simulation Scale

After identifying parasitic-sensitive nodes, we leverage a hybrid strategy to reduce the simulation scale by handling critical and non-critical parasitic sub-networks separately, as illustrated in Figure 4.

Partition the Circuit. The parasitic network extracted by the extraction tool must first be partitioned into N parasitic sub-networks based on pre-layout node. Following the fundamental principles of parasitic formation, the criterion is established as follows: post-layout nodes derived from the same pre-layout node are classified into the same sub-network. In particular, if two post-layout nodes are interconnected through a parasitic resistor, they are regarded as belonging to the same sub-network.

Collapse the parasitic sub-network for nodes that are parasitic-insensitive. To maximize the elimination of non-critical parasitics, a collapsing technique is employed. Specifically, if a node is identified as parasitic-insensitive, all resistors within the parasitic sub-network associated with that node are treated as short-circuit, and all capacitors are treated as open-circuit. The ground capacitors of the sub-network is replaced by an equivalent capacitor, and all internal nodes are merged into a single node. In other words, after transforming the parasitic sub-network into its corresponding matrices, the G matrix is reduced to a first-order zero matrix, while the C matrix is simplified to a first-order matrix, where its element equals sum of the diagonal elements of the original C matrix.

TABLE I: Node reduction and acceleration efficiency comparisons with different methods.

Circuit	#Nodes			Time (ms)			PiSPICE vs Original		PiSPICE vs TICER		Relative Error
	Original	TICER	PiSPICE	Original	TICER	PiSPICE	Reduction Rate	Speedup	Reduction Rate	Speedup	
CKT1 (Ac)	1219	567	16	135.699	40.176	19.686	76.19x	6.89x	35.44x	2.04x	0.05%
CKT1 (Tran)	1219	567	13	189.536	119.603	34.802	**93.77x**	5.45x	**43.62x**	3.44x	0.30%
CKT2 (Ac)	3457	1623	235	616.297	137.021	81.820	14.71x	7.53x	6.91x	1.67x	0.05%
CKT2 (Tran)	3457	1623	568	640.569	261.536	106.922	6.09x	5.99x	2.86x	2.45x	0.16%
CKT3 (Ac)	5048	2447	422	541.906	136.357	53.167	11.96x	10.19x	5.80x	2.56x	0.18%
CKT3 (Tran)	5048	2447	1018	926.216	408.313	140.629	4.96x	6.59x	2.40x	2.90x	0.08%
CKT4 (Ac)	7194	3370	987	3397.520	1159.860	196.770	7.29x	**17.27x**	3.41x	5.89x	0.37%
CKT5 (Ac)	8920	4272	1687	3025.370	733.360	280.880	5.29x	10.77x	2.53x	2.61x	0.34%
CKT6 (Ac)	12255	6329	455	2052.257	447.222	227.815	26.93x	9.01x	13.91x	1.96x	0.25%
CKT7 (Ac)	24306	12743	2171	6430.030	2753.240	888.470	11.20x	7.24x	5.87x	3.10x	0.43%
CKT8 (Tran)	46894	27288	23768	9021930.000	5040080.000	4380000.000	1.97x	2.06x	1.15x	1.15x	**0.78%**
CKT9 (Tran)	68450	37546	29860	10800002.043	6453020.000	679281.022	2.29x	15.90x	1.26x	**9.50x**	0.56%
Average		-			-		21.89x	8.74x	10.43x	3.27x	0.30%

Reduce the size of parasitic sub-networks of parasitic-sensitive nodes with improved PRIMA. Although it is not possible to directly collapse the parasitic sub-networks of parasitic-sensitive nodes, some steps can be taken to reduce its size, i.e., the MOR algorithm can be performed.

In our approach, we apply MOR to non-critical subnetworks, which are ideal candidates for PRIMA due to their smaller size and localized interactions. However, since the parasitic sub-network obtained by partitioning does not necessarily have ground resistor, the resulting G matrix, which is populated by internal interconnect resistor, may not be full-rank or invertible. Therefore, we can not have $R = G^{-1}B$ as the projection matrix like original PRIMA.

To address this issue, we adopt the reordering strategy from sparse implicit projection [37] to enhance PRIMA. By separating the internal and port nodes within each sub-network, the matrix G is reordered into a block matrix form, where A represents the conductance matrix between port nodes, B is the coupling conductance matrix between port and internal nodes, and D is the conductance matrix between internal nodes. This reordering enables us to compute $G^{-1}B$, which leads to the new projection matrix. Through further simplifications, we derive Eq. 7.

$$G = \begin{pmatrix} A & B \\ B^T & D \end{pmatrix}, \quad G^{-1}B = \begin{pmatrix} -A^{-1}B \\ I \end{pmatrix} \tag{7}$$

By using the projection matrix $R = G^{-1}B$ for MOR, the size of critical parasitic sub-network is further reduced.

IV. Experiments

A. Experimental Setup

To comprehensively evaluate the efficiency, accuracy, and applicability of PiSPICE, we use Python to implement the whole framework. The experiment involves a total of nine circuits as test cases, with node counts ranging from 1k to 60k. Specifically, CKT1-CKT6 are operational amplifier circuits of varying scales, CKT7 is a bandgap reference circuit, CKT8 is a 4-bit SAR ADC circuit, and CKT9 is an ultrafast clock fan-out buffer circuit. All experiments are executed on a system equipped with an AMD EPYC 2.3 GHz CPU and 377 GB of memory.

Both transient and AC analysis simulation are conducted. We compare among three different configurations: the original post-layout circuit, the post-layout circuit simplified using TICER, and the post-layout circuit simplified using our method, referred to as "Original", "TICER" and "PiSPICE", respectively. We use Calibre xRC [39] for parasitic extraction and Cadence Spectre [38] as the simulator. TICER is chosen as the baseline for comparison due to its widespread recognition as a reliable MOR method integrated into Calibre xRC, making it a well-respected tool across various design applications.

B. Reduction and Acceleration Efficiency

Table I presents the number of total nodes ($\#Nodes$) in post-layout simulation and the simulation time ($Time(ms)$) for the test circuits across different methods. Additionally, it highlights the improvements in parasitic network reduction and simulation acceleration efficiency achieved by PiSPICE compared to Original and TICER.

From this table, it is clear that PiSPICE achieves a substantial reduction in the number of nodes compared to TICER, due to its aggressive reduction strategy for non-critical parasitic sub-networks. Specifically, for the transient analysis simulation of CKT1, PiSPICE reduces the circuit to 93.77x fewer nodes compared to the original, offering a 43.62x improvement over TICER. This highlights the superior efficiency of PiSPICE in parasitic network simplification.

In addition, it can be also seen that PiSPICE significantly enhances acceleration efficiency by substantially reducing post-layout SPICE simulation time through effective circuit size reduction, thereby expediting the simulation solving process. Specifically, compared to the original post-layout SPICE simulation, i.e., Spectre, PiSPICE achieves a speedup ratio of 2.06x-17.27x. When compared to TICER, PiSPICE also demonstrates a speedup of 1.15x-9.50x.

C. Accuracy Analysis

To verify the accuracy of PiSPICE, we analyze the maximum relative error of the simulation results using our proposed PiSPICE and compare to the original simulation results, denoted as $Relative\ Error$ in Table I. It can be observed that the maximum error introduced by PiSPICE remains below 0.78%, which is deemed acceptable. Figure 5 further presents

the gain and CMRR curves, along with the corresponding absolute error, for CKT1, CKT2, and CKT3 during AC analysis simulation. It can be observed that the gain curves (or CMRR curves) of the original circuit and PiSPICE nearly overlap, with an absolute error of less than 0.6dB, demonstrating that PiSPICE achieves a high level of accuracy.

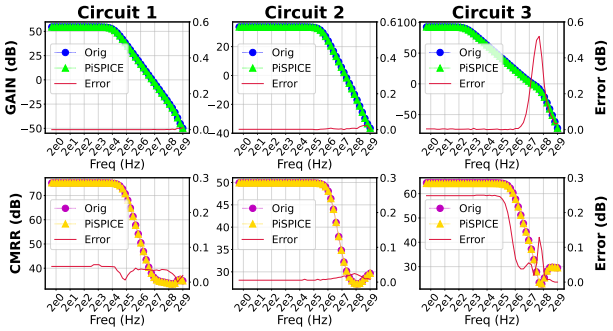


Fig. 5: The gain curves, CMRR curves and absolute error in CKT1, CKT2 and CKT3 under different methods.

TABLE II: The circuit scale and AC simulation accuracy of CKT3 after adjusting s_u.

Circuit	Value of s_u	#Nodes	Time (ms)	Relative Error
CKT3 (Ac)	1.0	10	36.793	3.10%
	0.8	196	48.980	1.10%
	0.5	**422**	**53.167**	**0.18%**
	0.2	2747	336.357	0.06%
	0.0	5048	541.906	0.00%

For PiSPICE, we enable a user-adjustable safety factor s_u to provide more flexibility in adjusting the threshold based on user requirements. When $s_u = 0.5$, it corresponds to the default threshold calculated by our method. When $s_u = 0$, it implies that all nodes are considered parasitic-sensitive, meaning the original post-layout circuit is used without any reduction. On the other hand, when $s_u = 1$, it indicates that all nodes are regarded as non-sensitive to parasitics and can be eliminated, resulting in a circuits identical to the pre-layout circuit. Table II presents the performance and error results after dynamically adjusting s_u. Users can balance the relaxation of sensitivity judgment between desired accuracy and speed.

D. Applicability Analysis

As mentioned in section III. C, our method achieves greater performance improvements in scenarios where the topology is the same but the parameters differ, such as in sizing. We only need to perform sensitivity analysis once and then transfer this information to similar circuits, significantly reducing the cost of sensitivity analysis. Here, we use CKT3 as an example. CKT3_1 in Table III represents the original circuit, while CKT3_2 to CKT3_4 are three similar circuits where transistor parameters are adjusted during the sizing process. We perform sensitivity analysis on CKT3_1 to identify the critical nodes in the circuit, and then use this information to directly reduce the RC networks of the other three circuits.

Table III shows the simulation results. As shown in the table, all the three netlists, CKT3_2 to CKT3_4, achieve significant performance improvements, with an average simulation speedup of 8.34x, while maintaining errors within 0.5%. Similar conclusions can be drawn from the reduction in circuit size (i.e., number of total nodes). Compared to directly obtain the RC networks from Calibre xRC, our method results in an average reduction of 13.1x in the size of the post-layout netlist.

TABLE III: Performance of PiSPICE in post-layout simulation with the same topology but different transistor parameters.

Circuit	#Nodes		Time (ms)			Speedup	Relative Error
	Original	PiSPICE	Original	PiSPICE			
CKT3_1 (Ac)	5048	422	541.906	53.167	**Sensitivity**	10.19x	0.18%
CKT3_2 (Ac)	3955	157	287.349	37.225	**Analysis**	7.72x	0.26%
CKT3_3 (Ac)	4699	976	323.481	50.390	**Time:**	6.42x	0.20%
CKT3_4 (Ac)	21262	2053	1805.040	199.978	**221.526**	9.03x	0.50%
Average	-		-			8.34x	0.29%
Time Sum	-		**2957.776**	**562.286**		5.26x	-

E. Overhead of Sensitive Analysis

Although our method significantly reduces the post-simulation circuit size and improves simulation efficiency, sensitivity analysis does come with a certain computational cost. As shown in Table III, the cost of sensitivity analysis in our method ranges from approximately 0.12x to 0.77x of a single simulation run (compared to the original simulation time). Fortunately, in scenarios where sensitivity results information is transferred across similar circuits, the total simulation time for four circuits (including the preprocessing cost of sensitivity analysis and the simulations themselves) still achieves a performance improvement of 5.26x. If more netlists that share the same topology need to be simulated during the sizing optimization process, the cost of sensitivity analysis becomes negligible.

V. Conclusion

In this paper, we propose PiSPICE, a novel framework designed to accelerate post-layout SPICE simulation by identifying and retaining only the essential parasitics. First, we leverage the topological correlation between pre- and post-layout circuits for parasitic modeling. Next, critical parasitics are identified through sensitivity analysis of the parasitic model. Finally, we collapse the critical parasitic sub-networks of parasitic-insensitive nodes, while the parasitic sub-networks of parasitic-sensitive nodes are reduced using a modified PRIMA approach. Experiment results demonstrate that PiSPICE achieves superior acceleration while maintaining comparable accuracy to the commercial simulator Spectre with TICER.

Acknowledgment

This work was supported by NSFC (Grant No.62234010, 92473107, 62204265, U23A20301, 62374031), Beijing National Research Center for Information Science and Technology (BNRist), Semiconductor Technology Innovation Center (Beijing) (Grant No.QYJS-2021-0900-B) and NSF of Jiangsu Province (BK20240173). Dan Niu and Zuochang Ye are corresponding authors.

REFERENCES

[1] T. Nakura, "SPICE Simulation," *Essential Knowledge for Transistor Level LSI Circuit Design*, pp. 19-47, 2016.

[2] D. Niu, Y. Dong, Z. Jin, C. Zhang, Q. Li and C. Sun, "OSSP-PTA: An Online Stochastic Stepping Policy for PTA on Reinforcement Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, no. 11, pp. 4310-4323, 2023.

[3] T. Wang, W. Li, H. Pei, Y. Sun, Z. Jin and W. Liu, "Accelerating Sparse LU Factorization with Density-Aware Adaptive Matrix Multiplication for Circuit Simulation," *ACM/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2023.

[4] Z. Jin, W. Li, Y. Bai, T. Wang, Y. Lu and W. Liu, "Machine Learning and GPU Accelerated Sparse Linear Solvers for Transistor-level Circuit Simulation: A Perspective Survey (Invited Paper)," *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 96–101, 2024.

[5] Z. Jin, T. Feng, X. Wu, D. Niu, Z. Zhou and C. Zhuo, "MSH: A Multi-Stage HiZ-Aware Homotopy Framework for Nonlinear DC Analysis," *Design, Automation and Test in Europe Conference (DATE)*, pp. 1-6, 2024.

[6] Z. Jin, H. Pei, Y. Dong, X. Jin, X. Wu, W.W. Xing and D. Niu, "Accelerating Nonlinear DC Circuit Simulation with Reinforcement Learning," *ACM/IEEE Design Automation Conference (DAC)*, pp. 619–624, 2022.

[7] W.W. Xing, X. Jin, T. Feng, D. Niu, W. Zhao and Z. Jin, "BoA-PTA: A Bayesian Optimization Accelerated PTA Solver for SPICE Simulation," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 28, no. 2, pp. 1-26, 2022.

[8] Y. Chen, H. Pei, X. Dong, Z. Jin and C. Zhuo, "Application of Deep Learning in Back-End Simulation: Challenges and Opportunities," *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 641-646, 2022.

[9] G. Feng, H. Wang, Z. Guo, M. Li, T. Zhao, Z. Jin, W. Jia, G. Tan and N. Sun, "Accelerating Large-Scale Sparse LU Factorization for RF Circuit Simulation," *International European Conference on Parallel and Distributed Computing (Euro-Par)*, pp. 182-195, 2024.

[10] Y. Dong, D. Niu, Z. Jin, C. Zhang, C. Sun and Z. Zhou, "ISPT-Net: A Noval Transient Backward-Stepping Reduction Policy by Irregular Sequential Prediction Transformer," *Design, Automation, and Test in Europe Conference (DATE)*, pp. 1-6, 2024.

[11] Y. Bai, X. Yang, Y. Lu, D. Niu, C. Zhuo, Z. Jin and W. Liu, "Efficient Spectral-Aware Power Supply Noise Analysis for Low-Power Design Verification," *Design, Automation, and Test in Europe Conference (DATE)*, pp. 1-6, 2024.

[12] X. Fu, B. Zhang, T. Wang, W. Li, Y. Lu, E. Yi, J. Zhao, X. Geng, F. Li, J. Zhang, Z. Jin and W. Liu, "PanguLU: A Scalable Regular Two-Dimensional Block-Cyclic Sparse Direct Solver on Distributed Heterogeneous Systems," *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 1-15, 2023.

[13] J. Zhao, Y. Wen, Y. Luo, Z. Jin, W. Liu and Z. Zhou, "SFLU: Synchronization-Free Sparse LU Factorization for Fast Circuit Simulation on GPUs," *ACM/IEEE Design Automation Conference (DAC)*, pp. 37-42, 2021.

[14] A. Zhao, T. Gu, Z. Bi, F. Yang, C. Yan, X. Zeng, Z. Lin, W. Hu and D. Zhou, "D3PBO: Dynamic Domain Decomposition-Based Parallel Bayesian Optimization for Large-Scale Analog Circuit Sizing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 29, no. 44, pp. 1-25, 2024.

[15] W. Yu, T. Zhang, X. Yuan and H. Qian, "Fast 3-D Thermal Simulation for Integrated Circuits With Domain Decomposition Method," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 32, no. 12, pp. 2014-2018, 2013.

[16] F. Bizzarri, A. Brambilla and G. Storti Gajani, "FastSpice Circuit Partitioning to Compute DC Operating Points Preserving Spice-Like Simulators Accuracy," *Simulation Modelling Practice and Theory*, vol. 81, pp.51-63, 2018.

[17] Z. Jin, T. Feng, Y. Duan, X. Wu, M. Cheng, Z. Zhou and W. Liu, "PALBBD: A Parallel ArcLength Method Using Bordered Block Diagonal Form for DC Analysis," *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 327-332, 2021.

[18] Y. Jiang, J. Song, X. Yang, X. Dong, S. Sun, Y. Lin, Z. Jin, X. Yin and C. Zhuo, "A Parallel Simulation Framework Incorporating Machine Learning-Based Hotspot Detection for Accelerated Power Grid Analysis," *ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, pp. 1-7, 2024.

[19] Y. Su, F. Yang and X. Zeng, "AMOR: An Efficient Aggregating Based Model Order Reduction Method for Many-Terminal Interconnect Circuits," *Design Automation Conference (DAC)*, pp. 295–300, 2012.

[20] J. Rommes and W. Schilders, "SparseRC: Sparsity Preserving Model Reduction for RC Circuits with Many Terminals," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 30, pp. 1828-1841, 2011.

[21] D. Oyaro and P. Triverio, "TurboMOR-RC: An Efficient Model Order Reduction Technique for RC Networks With Many Ports," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 35, pp. 1695-1706, 2016.

[22] P. Liu, S.X.-D. Tan, B. McGaughy, L. Wu and L. He, "TermMerg: An Efficient Terminal-Reduction Method for Interconnect Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 8, pp. 1382-1392, 2007.

[23] Z. Zhang, X. Hu, C. -K. Cheng and N. Wong, "A Block-Diagonal Structured Model Reduction Scheme for Power Grid Networks," *Design, Automation and Test in Europe Conference (DATE)*, pp. 1-6, 2011.

[24] P. Chen, D. Niu, Z. Jin, C. Sun, Q. Li and H. Yan, "TSA-TICER: A Two-Stage TICER Acceleration Framework for Model Order Reduction, "*Design, Automation, and Test in Europe Conference (DATE)*, pp. 1-6, 2024.

[25] Z. Zhang, D. Niu, Z. Jin, P. Chen, Z. Zhou and C. Sun, "P-TICER: An Effective Parallel TICER Acceleration Method for Model Order Reduction," *International Symposium of Electronics Design Automation (ISEDA)*, pp. 125-130, 2024.

[26] L. Hao and G. Shi, "Realizable Reduction of Multi-Port RCL Networks by Block Elimination," *IEEE Transactions on Circuits and Systems I (TCAS-I)*, vol. 70, no. 1, pp. 399-412, 2023.

[27] S. Director and R. Rohrer, "Automated Network Design-The Frequency Domain Case," *IEEE Trans. Circuit Theory*, vol. 16, no. 3, pp. 330–337, 1969.

[28] O. Aydogmus and A.TOR, "A Modified Multiple Shooting Algorithm for Parameter Estimation in ODEs Using Adjoint Sensitivity Analysis," *Applied Mathematics and Computation*, vol. 390, pp. 125644, 2021.

[29] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah and C. Wu, "JiffyTune: Circuit Optimization Using Time-Domain Sensitivities," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 17, pp. 1292-1309, 1998.

[30] Z. Gao and R. Rohrer, "Efficient Non-Monte-Carlo Yield Estimation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 5, pp. 1222–1235, 2022.

[31] W. Hu, Z. Wang, S. Yin, Z. Ye and Y. Wang, "Sensitivity Importance Sampling Yield Analysis and Optimization for High Sigma Failure Rate Estimation," *ACM/IEEE Design Automation Conference (DAC)*, pp. 895-900, 2021.

[32] W. Hu, Z. Ye and Y. Wang, "Adjoint Transient Sensitivity Analysis for Objective Functions Associated to Many Time Points," *ACM/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2020.

[33] J. Li, D. Ahsanullah, Z. Gao and R. Rohrer, "Circuit Theory of Time Domain Adjoint Sensitivity," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 42, pp. 2303-2316, 2023.

[34] C. Li, B. Zhang, Y. Duan, Y. Li, Z. Ye, W. Liu, D. Tao and Z. Jin, "MASC: A Memory-Efficient Adjoint Sensitivity Analysis through Compression Using Novel Spatiotemporal Prediction," *ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. 2024.

[35] A. Odabasioglu, M. Celik and L. Pileggi, "PRIMA: Passive Reduced-Order Interconnect Macromodeling Algorithm," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 58–65, 1997.

[36] B. N. Sheehan, "TICER: Realizable Reduction of Extracted RC Circuits," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 200-203, 1999.

[37] Z. Ye, D. Vasilyev, Z. Zhu and J. Phillips, "Sparse Implicit Projection (SIP) for Reduction of General Many-Terminal Networks," *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 736–743, 2008.

[38] Cadence, "Cadence Spectre Simulation Platform, " Cadence. [Online]. Available: https://www.cadence.com.

[39] Siemens, "Calibre xRC," Siemens. [Online]. Available: https://eda.sw.siemens.com.

SDM-PEB: Spatial-Depthwise Mamba for Enhanced Post-Exposure Bake Simulation

Ziyang Yu[1], Peng Xu[1], Zixiao Wang[1], Binwu Zhu[2], Qipan Wang[3], Yibo Lin[3], Runsheng Wang[3], Bei Yu[1], Martin Wong[4]
[1]The Chinese University of Hong Kong [2]Southeast University&NCTIEDA [3]Peking University [4]Hong Kong Baptist University

***Abstract*—The post-exposure bake (PEB) process is a critical step in semiconductor lithography, directly impacting resist profile accuracy and circuit pattern fidelity. Precise modeling of PEB is essential for controlling photoacid diffusion and inhibitor reactions. In this paper, we introduce SDM-PEB, an advanced modeling framework designed to enhance the accuracy of PEB simulations by capturing both intra-layer spatial dependencies and inter-layer depthwise interactions. Leveraging a unique hierarchical feature extractor with overlapped patch merging and efficient self-attention, our approach effectively captures both coarse and fine features at multiple scales. The spatial-depthwise Mamba-based attention unit, centered on a customized selective scan and structured state space model, efficiently captures spatial and depthwise dependencies, enabling precise 3D PEB simulation. Additionally, a PEB focal loss and differential depth divergence regularization term improve the sensitivity to both spatial and depthwise variations, addressing inherent data imbalances in 3D PEB simulations. Our framework is validated with commercial rigorous model, and experimental results demonstrate that the SDM-PEB outperforms previous methods in accuracy and efficiency.**

I. Introduction

Lithography simulation plays a vital role in semiconductor manufacturing, providing precise predictions and optimizations for transferring circuit patterns onto silicon wafers to create integrated circuits [1]. As devices become increasingly miniaturized and designs grow more intricate, achieving high-resolution patterns becomes increasingly challenging [2], [3]. This amplifies the role of lithography simulation in electronic design automation (EDA), where it accelerates design cycles, integrates manufacturability considerations early, and ensures adherence to strict performance and yield standards. By minimizing the cost and time of experimental iterations, lithography simulation has become essential for addressing the complexities of modern semiconductor technologies and advancing the efficiency and accuracy of the entire EDA workflow.

Predictive simulation of lithographic processes requires the modeling of various physical and chemical phenomena. Typical lithography simulation flow for chemically amplified resist (CAR) consists of optical simulation and photoresist simulation, as can be seen in Fig. 1. Optical simulation models the light exposure process where light interacts with photomasks to project patterns onto the photoresist layer. In recent years, significant advancements have been made in optical simulation, driven by the need for precise control over light exposure and pattern fidelity [4]–[10]. Advanced techniques, such as source-mask optimization (SMO), further enhance optical simulation by fine-tuning exposure parameters [11], [12]. Photoresist simulation, on the other aspect, models the chemical and physical processes occurring within the photoresist layer during lithography, from exposure through post-exposure bake (PEB) to development. PEB is crucial in lithography, as it mitigates the standing wave effect and ensures consistent feature development [13]. In positive-tone CAR, exposure to an aerial image decomposes the photoacid generator, creating an acid latent image within the resist. During the subsequent PEB process, the resist is baked at elevated temperature, triggering an acid-catalyzed deprotection reaction that decomposes the inhibitor material [14]. By the end of PEB, the resist in the exposed area is deprotected, resulting in a higher development rate compared to unexposed area. The inhibitor concentration distribution determines the final resist profile after development. In regions where inhibitor concentration decreases due to acid reactions, the resist dissolves, forming the desired pattern. PEB simulation simulates the remaining inhibitor concentration based on the initial 3D photoacid profile, working as a connection between optical simulation and final printed features. Fine-tuning simulation tools is essential to accurately model the acid's catalytic behavior and predict final feature sizes.

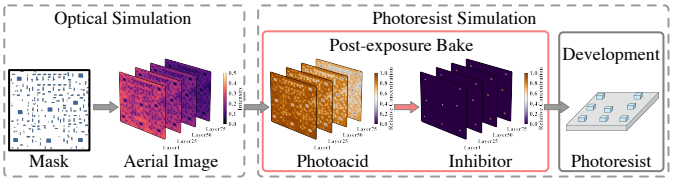


Fig. 1 A typical flow of lithography simulation for chemically amplified resist: from optical simulation to photoresist simulation.

According to the studies in [15], PEB simulation accounts for up to 30% of the runtime in the rigorous lithography simulation flow of Synopsys Sentaurus Lithography (S-Litho), which is non-negligible. Compared to machine learning-based optical simulation, PEB simulation often requires several times more computational time, highlighting the need for more efficient modeling approaches. Early research from the 1990s to early 2000s focused on acid-catalyzed reactions and physical dynamics governing pattern formation, using simplified reaction-diffusion equations [16], [17], and 3D diffusion profile simulations [18]. These methods demand significant computational resources, especially when simulating large-scale patterns. Methods such as finite element analysis (FEA) and finite difference methods (FDM) achieve spatial and temporal precision but are computationally intensive and limited in scalability for large-scale patterns. The lack of adaptive mesh refinement and seamless integration into lithography workflows further hinders efficiency and accuracy. Despite advancements in optical simulation, PEB simulation, crucial for accurate patterning, remains comparatively underexplored and could benefit from further development. Unlike 2D simulations, the 3D distribution characteristic of PEB introduces significant challenges in modeling complexity and computational cost. Recent state-of-the-art work DeePEB [15] addresses PEB modeling in CAR using neural networks, leveraging Fourier Neural Operator (FNO) [19] and convolution network to capture low-frequency global information and high-frequency local details. However, this approach struggles to fully capture continuous spatial and depthwise dependencies in 3D space, which are critical in PEB simulation where gradients and interactions across all three dimensions influence outcomes. Additionally, the frequency segmentation in FNO risks incomplete representations, while convolution operations may miss intricate high-frequency details required in 3D structures. The computational overhead of FNO

also impacts its efficiency, posing further challenges.

To address the above challenges in precisely and efficiently modeling PEB process, we propose SDM-PEB, designed to fully capture the spatial and depthwise dependencies inherent in complex physical and chemical reactions. Inspired by the recent advancements in Vision Transformers (ViT) [20]–[22], we adopt hierarchical transformer-based encoders with efficient spatial self-attention to extract both high-resolution coarse features and low-resolution fine-grained sequence features. Given that neighboring layers in the photoacid distribution provide valuable context, spatial and depthwise correlations require a carefully designed self-attention mechanism. Mamba [23], [24], a recent innovation based on state space models, efficiently captures long-range dependencies through selective scanning of feature sequences. Building on these new advancements, we introduce a customized spatial-depthwise Mamba-based attention unit (SDM unit) in each encoder layer, integrating parallel three-direction selective scan mechanisms of space state sequences to address both intra-layer spatial coherence and inter-layer interactions. Furthermore, a novel PEB focal loss is proposed to address the issue of imbalanced distribution of inhibitor. A differential divergence regularization is utilized to account for inter-layer variations. Our major contributions are summarized as follows:

- We employ a hierarchical vision transformer encoder to extract multi-scale spatial information within a single photoacid depth level.
- We propose a customized spatial-depthwise Mamba-based attention unit to enhance the model's capability in capturing cross-depth-level dependencies.
- We proposed the PEB focal loss and the differential depthwise divergence regularization to refine the learning objective, addressing reaction imbalances and reducing depthwise divergence.
- We evaluate our framework using the industry-verified software S-Litho, and the results demonstrate superior precision and efficiency compared to existing methods.

II. Preliminaries

A. Photoresist Simulation Process

Post-Exposure Bake Process. Different resists undergo distinct physical and chemical reactions during PEB [13]. This paper focuses on positive-tone chemically amplified resists (CAR), the predominant materials used in ArF (193 nm) exposure for advanced semiconductor manufacturing. Modern CARs comprise protected polymers or dissolution inhibitors ($\mathcal{I}$), photoacid generators (PAG), and quencher bases ($\mathcal{B}$) [25]. The reaction process aims to create solubility contrast in the resist, enabling pattern formation. During lithographic exposure prior to PEB, incident light decomposes the PAG, producing photoacid ($\mathcal{A}$). In the subsequent thermal-driven PEB process, the photoacid catalyzes a reaction that deblocks the inhibitor. This catalytic reaction amplifies the effects of exposure and can be described by the catalytic reaction equation:

$$\frac{\partial[\mathcal{I}]}{\partial t} = -k_c[\mathcal{I}][\mathcal{A}], \tag{1}$$

where $[\mathcal{I}]$ and $[\mathcal{A}]$ denote the normalized concentration distributions of the inhibitor and photoacid, respectively, and k_c represents the catalysis coefficient.

The generated photoacid molecules diffuse within the resist during the process and are neutralized by the base quencher upon contact. This acid-base neutralization regulates acid diffusion, preventing excessive spreading and reactions in undesired regions. The base quencher also diffuses during PEB. The general reaction-diffusion process for normalized concentration distribution of photoacid $[\mathcal{A}]$ and base quencher $[\mathcal{B}]$ can be expressed:

$$\frac{\partial[\mathcal{A}]}{\partial t} = -k_r[\mathcal{A}][\mathcal{B}] + D_{\mathcal{A}}\nabla^2[\mathcal{A}], \tag{2}$$

$$\frac{\partial[\mathcal{B}]}{\partial t} = -k_r[\mathcal{A}][\mathcal{B}] + D_{\mathcal{B}}\nabla^2[\mathcal{B}]. \tag{3}$$

In the above equation, the first term represents the neutralization reaction with k_r the reaction coefficient, the second term is Fick's law of diffusion in three dimensions, $D_{\mathcal{A}}$, $D_{\mathcal{B}}$ are the diffusion coefficients. The diffusion length L depends on diffusion coefficient and PEB duration T: $L = \sqrt{2D \cdot T}$.

For initial conditions (I.C.), without loss of generality, we consider uniform base and inhibitor distributions, while photoacid concentration is derived from the 3D aerial image via the Dill Model [26]. Boundary conditions (B.C.) are critical in confined structures like thin films. In x-y dimensions, we consider a commonly used zero-flux B.C., ensuring no flux across boundaries [27], [28]. In the z dimension, Robin B.C. describes photoacid in/out-diffusion at the resist surface:

$$\frac{\partial[\mathcal{A}]}{\partial \boldsymbol{n}} = 0, \qquad D_{\mathcal{A}}\frac{\partial[\mathcal{A}]}{\partial z} = h([\mathcal{A}]_{top} - [\mathcal{A}]_{sat}), \tag{4}$$

where $\boldsymbol{n}$ is the normal direction, h is the transfer coefficient and $[\mathcal{A}]_{sat}$ is the saturation concentration.

Development Process. After PEB, the development process transforms the chemically modified photoresist into a physical pattern. During this stage, exposed regions dissolve in the developer solution at a rate $\boldsymbol{R}$, which depends on the inhibitor concentration and is described by the Mack kinetic model [29]:

$$\boldsymbol{R}(x,y,z) = R_{max}\frac{(a+1)(1-[\mathcal{I}])^n}{a+(1-[n])^n} + R_{min},\ a = (1-M_{th})^n\frac{n+1}{n-1}. \tag{5}$$

Here R_{max} and R_{min} are the maximum and minimum development rates for fully exposed and unexposed regions, respectively. n is the surface reaction order, M_{th} is the experimentally determined threshold.

Using local development rates, the resist profile at a given development time t_d can be evaluated. The propagation of the development front $S_d(x,y,z)$ is governed by the Eikonal equation [30]: $|\nabla S_d(x,y,z)| = 1/R(x,y,z)$. In this work, we compute development rates from the inhibitor distribution and use an open-source Eikonal Solver [31] to calculate the development profile.

B. State Space Models-based Methodologies

Feature representations in 3D photoresist, characterized by consecutive depthwise distributions, can be naturally modeled as sequences of depth-levels (from shallow to deep). Recent advances in state space models (SSMs) have demonstrated their effectiveness in capturing long-range dependencies while supporting efficient parallel training. Originating from the Kalman filter [32], SSMs are linear time-invariant systems that map a scalar sequence $x(t)$ to another scalar sequence $y(t)$ via a hidden state $\boldsymbol{h}(t) \in \mathbb{R}^N$, where $t \in \{1 \dots L\}$. The system evolves with evolution parameter $\boldsymbol{A} \in \mathbb{R}^{N\times N}$, initialized using HiPPO matrix [33], and the projection parameters $\boldsymbol{B} \in \mathbb{R}^{N\times 1}$ and $\boldsymbol{C} \in \mathbb{R}^{N\times 1}$ as:

$$\boldsymbol{h}'(t) = \boldsymbol{A}\boldsymbol{h}(t) + \boldsymbol{B}x(t),\ y(t) = \boldsymbol{C}\boldsymbol{h}(t). \tag{6}$$

To adapt to deep learning, continuous-time SSMs are discretized using the zero-order hold (ZOH) assumption [34], transforming continuous-time parameters $\boldsymbol{A}$ and $\boldsymbol{B}$ into discrete equivalents:

$$\bar{\boldsymbol{A}} = \exp(\boldsymbol{\Delta A}),\ \bar{\boldsymbol{B}} = (\boldsymbol{\Delta A})^{-1}(\exp(\boldsymbol{\Delta A}) - \boldsymbol{I}) \cdot \boldsymbol{\Delta B}, \tag{7}$$

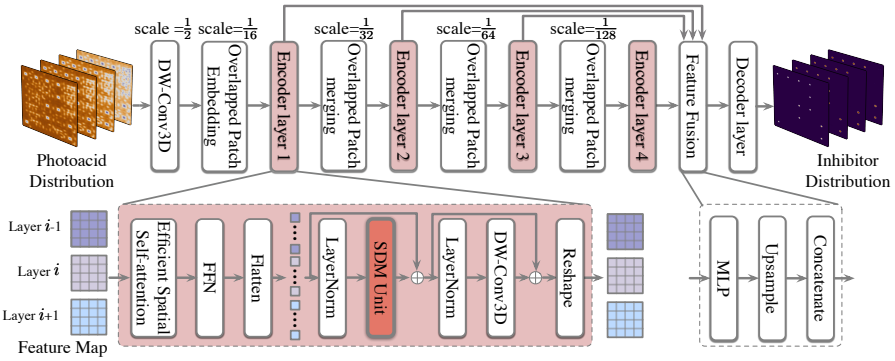


Fig. 2 The architecture overview of our proposed SDM-PEB framework.

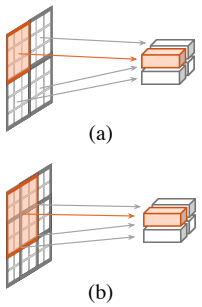


Fig. 3 (a) Non-overlapped patch merging and (b) overlapped patch merging.

with a specified time step $\boldsymbol{\Delta}$. The Equation (6) can be expressed in the discrete form:

$$\boldsymbol{h}_t = \bar{\boldsymbol{A}}\boldsymbol{h}_{t-1} + \bar{\boldsymbol{B}}x_t,\ y_t = \boldsymbol{C}\boldsymbol{h}_t. \tag{8}$$

To improve computational efficiency and scalability, the iterative process in Equation (8) is reformulated as a single global convolution:

$$\boldsymbol{y} = \boldsymbol{x} * \bar{\boldsymbol{K}},\ \bar{\boldsymbol{K}} = (\boldsymbol{C}\bar{\boldsymbol{B}}, \boldsymbol{C}\bar{\boldsymbol{A}}\bar{\boldsymbol{B}}, \cdots, \boldsymbol{C}\bar{\boldsymbol{A}}^{L-1}\bar{\boldsymbol{B}}), \tag{9}$$

where $*$ denotes the convolution operation.

Mamba [23], a recent selective scan SSM, excels in long-sequence modeling with linear complexity. By associating SSM parameters with the input, Mamba selectively focuses on relevant information while ignoring irrelevant inputs. This is achieved by configuring the projection matrix to be dependent on input sequence $\boldsymbol{x}$ with K channels:

$$\boldsymbol{B} = \texttt{Linear}_N(\boldsymbol{x}),\ \boldsymbol{C} = \texttt{Linear}_N(\boldsymbol{x}), \tag{10}$$

$$\boldsymbol{\Delta} = \texttt{softplus}(\texttt{Broadcast}_K(\texttt{Linear}_1(\boldsymbol{x})) + \boldsymbol{D}), \tag{11}$$

where $\texttt{Linear}_N$ projects to a hidden dimension N, $\texttt{Broadcast}_K$ expands 1D data into a K-dimensions, and $\boldsymbol{D}$ is a constant weight matrix. To enhance GPU utilization and memory efficiency, Mamba employs a hardware-aware algorithm for selective SSM computation with linear scalability relative to sequence length. By combining kernel fusion and recomputation within parallel scans, this approach minimizes memory I/O operations, resulting in substantial improvements in computation speed.

C. Problem Formulation

To enhance PEB simulation accuracy, we focus on predicting the 3D inhibitor distribution, known as the PEB latent image. Following prior work [15], we evaluate the predicted inhibitor distribution $\mathcal{I}$, development rate R, and resulting photoresist profile using the following metrics:

Root Mean Squared Error (RMSE). Root Mean Squared Error is commonly used to measure the difference between the predicted $\hat{\boldsymbol{P}}$ and the ground truth $\boldsymbol{P}$:

$$\text{RMSE} = \sqrt{\frac{\left\|\hat{\boldsymbol{P}} - \boldsymbol{P}\right\|^2}{n}},\ \boldsymbol{P} = [\mathcal{I}],\ \boldsymbol{R}. \tag{12}$$

Normalized Root Mean Squared Error (NRMSE). Normalized Root Mean Squared Error (NRMSE) accounts for the magnitude of the true values, making the error more interpretable, especially for datasets with different scales or units:

$$\text{NRMSE} = \frac{\left\|\hat{\boldsymbol{P}} - \boldsymbol{P}\right\|_F}{\|\boldsymbol{P}\|_F},\ \boldsymbol{P} = [\mathcal{I}],\ \boldsymbol{R}, \tag{13}$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

Photoresist Profile Critical Dimension Error (CD Error). In addition to the direct comparison of inhibitor distribution and development rate, CD error assesses the photoresist profile's accuracy by comparing printed feature dimensions CD_d in the x-y plane to target values:

$$\text{CD Error}_d = \sqrt{\frac{(\hat{\text{CD}}_d - \text{CD}_d)^2}{n}},\ d = x,\ y. \tag{14}$$

Problem 1 (PEB simulation problem)**.** *Given initial 3D photoacid distributions $\{[\mathcal{A}]_i\}_{i=1}^N$ and corresponding inhibitor distributions $\{[\mathcal{I}]_i\}_{i=1}^N$ with spatial dimensions D, H, W (depth, height, width), the objective is to develop and train a deep-learning model $\mathcal{F}_\Theta : [\mathcal{A}] \mapsto [\mathcal{I}]$, parameterized by Θ, to accurately predict inhibitor distributions for test mask patterns. The model must capture both spatial and depthwise dependencies, minimizing RMSE, NRMSE, and CD error while ensuring efficient computational runtime.*

III. Framework

A. Overview

In post-exposure bake (PEB) simulation, accurately modeling spatial (x-y plane) and depthwise (z-axis) dependencies is crucial for predicting complex 3D distributions. To enhance the model's ability to capture nuanced details essential for precise pattern representation in lithography, we propose the following:

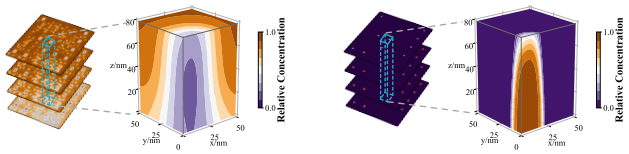
1) **Hierarchical Contextual Feature Extractor**, designed to capture both coarse and fine-grained spatial features at each depth level;
2) **Spatial-Depthwise Mamba-based Attention Unit**, developed to model cross-depth-level dependencies effectively.

Fig. 2 provides an overview of the SDM-PEB architecture. The design of the proposed framework is detailed in this section.

B. Hierarchical Contextual Feature Extractor

A single-scale feature extractor with fixed resolution often struggles to effectively capture the complex patterns and multi-scale dependencies inherent in the PEB simulation task. Inspired by advancements in hierarchical feature extractors [35], [36], we adopt a hierarchical feature encoder to generate high-resolution coarse features and low-resolution fine-grained features. To further optimize the trade-off between performance and efficiency in the PEB simulation task, we incorporate two key customizations.

Depthwise Overlapped Patch Merging. Local dependencies between neighboring regions are essential for modeling reactions influenced by adjacent concentrations. These dependencies help the model understand photoacid diffusion and reaction rate variations.



(a) (b)

Fig. 4 Vertical visualization of distributions: (a) photoacid at the initial stage and (b) inhibitor at the final stage.

To reduce information loss at patch boundaries and enhance local continuity, we apply depthwise overlapped patch merging instead of non-overlapped patch merging to downsample spatial feature maps while retaining depth resolution. A comparative illustration is provided in Fig. 3. Additionally, The channel dimension increases after each merging layer, capturing increasingly complex and abstract feature representations.

Efficient Spatial Self-Attention. ViT-based feature extractors have achieved great success in general computer vision tasks by employing spatial self-attention to dynamically assign importance to input features. However, the computational complexity of standard multi-head self-attention is $O(L^2)$, where L is the sequence length of the input.

In the task of PEB prediction, the input photoacid distributions are typically large due to high-resolution requirements and include a depth dimension, making the computational cost of standard attention mechanisms infeasible. To address this challenge, we apply a reduction ratio r to shorten the sequence length, as suggested in [35]. The transformation is defined as follows:

$$\hat{\boldsymbol{K}} = \texttt{Reshape}\left(\frac{L}{r}, C \cdot r\right)(\boldsymbol{K}), \quad \boldsymbol{K} = \texttt{Linear}_C(\hat{\boldsymbol{K}}), \tag{15}$$

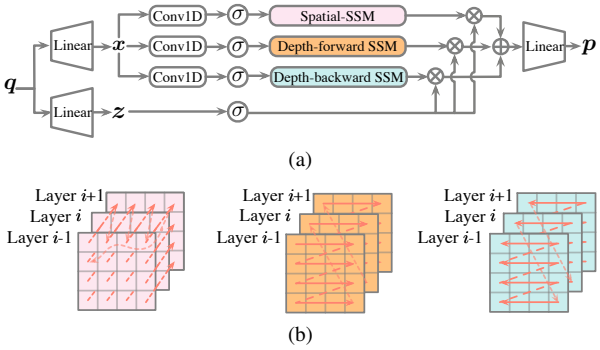
where $\boldsymbol{K}$ is the key in the attention mechanism, and C is the feature dimension of $\boldsymbol{K}$. After the transformation, the sequence length of $\boldsymbol{K}$ is reduced by the reduction ratio r, decreasing the computational complexity from $O(L^2)$ to $O(L^2/r)$, thereby making the computational cost manageable.

C. Spatial-Depthwise Mamba-based Attention Unit

As described in Equations (1) and (2), photoacid molecules generated during exposure diffuse and trigger chemical reactions in adjacent layers, altering the inhibitor profile. Therefore, depthwise interactions are crucial for accurately predicting the final inhibitor distribution and resist profile. This causal depthwise dependency can be observed in the vertical profiles shown in Fig. 4, which exhibit continuous and gradual changes in photoacid and inhibitor distributions.

Given that Mamba [23] is renowned for its ability to model sequential information, we propose the spatial-depthwise Mamba-based attention unit (SDM Unit) to extract inter-depth-level relationships. The SDM Unit extends bidirectional scanning into three dimensions to better aggregate depthwise information and employs structured state space models to effectively capture both spatial and depthwise dependencies.

Spatial-Depthwise PEB Selective Scan. As introduced in Section II, Mamba is traditionally designed for 1D sequences. To extend Mamba to the PEB simulation task and enhance its capability for 3D photoacid-inhibitor modeling, the selective scan path must be carefully designed. Building on the ideas from previous works [24], [37], [38], we propose employing three types of selective scans simultaneously: (1) **Spatial Scan**: The spatial scan operates along the depth dimension, focusing on information at a specific spatial position



(a)

(b)

Fig. 5 (a) The architecture of the spatial-depthwise Mamba-based attention unit; (b) Illustration of the three-direction PEB selective scan, from left to right: spatial scan, depth-forward scan, and depth-backward scan.

across all depth layers; (2) **Depth-Forward Scan**: The depth-forward scan processes the entire shallow level first before transitioning to deeper levels; (3) **Depth-Backward Scan**: The depth-backward scan follows the reverse order of the depth-forward scan, processing deeper levels before moving to shallower ones. We illustrate the scan methods in Fig. 5(b). The spatial-depthwise three-direction PEB selective scan, serves as the core element in the SDM Unit, and leverages parallel SSMs to establish the intra- and inter-depth long-range dependencies.

Structure of SDM Unit. The structure of the spatial-depthwise Mamba-based attention unit (SDM Unit) is illustrated in Fig. 5(a). Specifically, the feed-forward layer output in the i-th encoder layer, comprising feature map embeddings with dimensions $\mathbb{R}^{C_i \times D \times H_i \times W_i}$, is reshaped into a sequence $\boldsymbol{q}_i$ of size $\mathbb{R}^{C_i \times DH_iW_i}$, and normalized through a normalization layer. This sequence is then fed into the SDM Unit, which enhances both inter-layer and intra-layer distribution dependencies. The normalized sequence $\boldsymbol{q}_i$ is linearly projected into $\boldsymbol{x}_i$ and $\boldsymbol{z}_i$ with hidden dimension C_i^h. For each scanning direction, a 1D convolution followed by SiLU activation [39] is applied to $\boldsymbol{x}$, resulting in $\boldsymbol{x}'_d$ as input for the d-direction spatial-depthwise PEB selective scan: $\boldsymbol{B}, \boldsymbol{C}, \boldsymbol{\Delta}$ are derived fromEquation (10), and discretized values $\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}$ are computed using Equation (7). The outputs of the selective scan are calculated via Equation (9). These outputs are then weighted by $\boldsymbol{z}_i$ and combined to produce the final feature $\boldsymbol{p}_i$. A depthwise convolution with a kernel size of 3 is subsequently applied to refine fine-grained features.

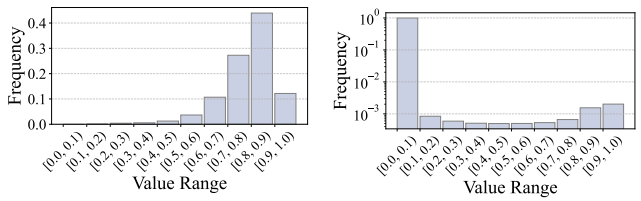
By dynamically adjusting attention weights across spatial and depthwise dimensions, the model effectively captures both local and global patterns within the resist layers. This enables accurate modeling of intricate physical-chemical interactions in 3D inhibitor distributions, resulting in a precise inhibitor profile.

D. Customized PEB Optimization Objectives

To accommodate the exponential variation of inhibitor as indicated in Equation (1), we apply the label normalization technique from [15], specifically predicting the quadratic negative logarithmic transformation of the inhibitor distribution as the output $\mathcal{Y}$ rather than the raw distribution. This transformation is defined as $\mathcal{Y} = -\ln(-\ln([\mathcal{I}])/k_c)$. In previous work [15], the authors used the single maximum squared error (MaxSE):

$$\mathcal{L}_{\text{MaxSE}} = \max_{d,h,w}\left(\hat{\mathcal{Y}}_{d,h,w} - \mathcal{Y}_{d,h,w}\right)^2. \tag{16}$$

However, by emphasizing only the maximum error, MaxSE may not reflect the overall quality of predictions across the entire distribution. Additionally, the model could become unstable, potentially oscillating



(a) (b)

Fig. 6 Distribution frequencies of (a) photoacid value ranges and (b) inhibitor value ranges.

to correct extreme outliers rather than achieving balanced accuracy. In this work, we introduce a customized PEB focal loss and a differential depth divergence regularization to address the complexity of the 3D distribution.

PEB Focal Loss. As can be seen from Fig. 6, the distributions of both photoacid and inhibitor are highly imbalanced: In every layer, photoacid spreads widely but shows significant exception inside contacts meanwhile inhibitor tends to be localized inside contacts. The frequency of the inhibitor distribution can even differ by several orders of magnitude. This imbalance skews the data toward low-variance background regions, making it difficult for the model to focus on subtle but critical changes near contacts. Inspired by the focal loss proposed for hard example mining [40], we design a PEB focal loss to address the imbalance issue:

$$\mathcal{L}_{\text{PEB-FL}} = \sum_{d}^{D}\sum_{h}^{H}\sum_{w}^{W}\left|\hat{\mathcal{Y}}_{d,h,w} - \mathcal{Y}_{d,h,w}\right|^{\gamma}\left(\hat{\mathcal{Y}}_{d,h,w} - \mathcal{Y}_{d,h,w}\right)^{2}, \tag{17}$$

where γ is the focusing parameter, empirically set to 1 in our experiment.

Differential Depth Divergence Regularization. In 3D PEB simulation, understanding how the inhibitor distribution evolves across layers is crucial for enhancing spatial-depthwise dependencies. We further propose the differential depth divergence regularization focusing on aligning inter-layer differences. For every pair $\hat{\mathcal{Y}}, \mathcal{Y} \in \mathbb{R}^{D\times H\times W}$, we first calculate their layer-wise forward difference maps $\Delta\hat{\mathcal{Y}}, \Delta\mathcal{Y} \in \mathbb{R}^{(D-1)\times H\times W}$, in which d-th layer can be expressed:

$$\Delta\hat{\mathcal{Y}}_d = \hat{\mathcal{Y}}_{d+1} - \hat{\mathcal{Y}}_d,\ \ \Delta\mathcal{Y}_d = \mathcal{Y}_{d+1} - \mathcal{Y}_d. \tag{18}$$

We then convert the difference maps into probabilities by applying the softmax function across the height and width dimensions, in which high difference layers are penalized:

$$\sigma(\Delta\hat{\mathcal{Y}}_d) = \frac{\exp(\Delta\hat{\mathcal{Y}}_d/\tau)}{\sum_{h=1}^{H}\sum_{w=1}^{W}\exp(\Delta\hat{\mathcal{Y}}_{d,h,w}/\tau)}, \tag{19}$$

$$\sigma(\Delta\mathcal{Y}_d) = \frac{\exp(\Delta\mathcal{Y}_d/\tau)}{\sum_{h=1}^{H}\sum_{w=1}^{W}\exp(\Delta\mathcal{Y}_{d,h,w}/\tau)}, \tag{20}$$

where τ is the parameter to sharpen the probability distribution, we empirically set to 0.1 here. The differential depth divergence regularization $\mathcal{L}_{\text{Div}}$ is formulated as the Kullback-Leibler divergence between the predicted difference map and the ground difference map:

$$\mathcal{L}_{\text{Div}} = \sum_{d=1}^{D-1}\sigma(\Delta\hat{\mathcal{Y}}_d)\log\frac{\sigma(\Delta\hat{\mathcal{Y}}_d)}{\sigma(\Delta\mathcal{Y}_d)}. \tag{21}$$

The overall loss function is the linear combination of the three weighted by α and β with empirical values 1.0 and 0.1. The learning on the whole distribution can be enhanced:

$$\mathcal{L} = \mathcal{L}_{\text{MaxSE}} + \alpha\mathcal{L}_{\text{PEB-FL}} + \beta\mathcal{L}_{\text{Div}}. \tag{22}$$

TABLE I Important parameters in photoresist simulation process.

PEB			
Normal Diffusion Length $L_{N,\mathcal{A}}, L_{N,\mathcal{B}}$	70, 15 nm	Lateral Diffusion Length $L_{L,\mathcal{A}}, L_{L,\mathcal{B}}$	10, 10 nm
catalysis coefficient k_c	0.9 /s	reaction coefficient k_r	8.6993 /s
transfer coefficient h_A, h_B	0.027, 0	saturation concentration $[\mathcal{A}]_{sat}, [\mathcal{B}]_{sat}$	0.9, 0
$[\mathcal{I}](t=0)$	1.0	$[\mathcal{B}](t=0)$	0.4
Baseline Time step	0.1 s	Duration	90 s
Develop			
R_{max}	40 nm/s	R_{min}	0.0003 nm/s
M_{th}	0.5	n	30
Duration	60 s		

TABLE II Comparison with different PEB solvers.

Methodologies	Inhibitor RMSE (e-3)	Inhibitor NRMSE (%)	Develop Rate RMSE (nm/s)	Develop Rate NRMSE (%)	CD Error x (nm)	CD Error y (nm)	RT/s
DeepCNN [41]	8.25	12.53	0.65	1.63	3.14	6.26	**1.01**
TEMPO-resist [5]	7.67	12.55	0.50	1.26	2.12	2.45	6.48
FNO [19]	7.91	11.68	0.68	1.69	2.34	3.71	1.15
DeePEB [15]	3.99	5.70	0.48	1.19	0.98	1.24	1.37
SDM-PEB	**2.78**	**3.70**	**0.35**	**0.86**	**0.74**	**0.93**	1.06

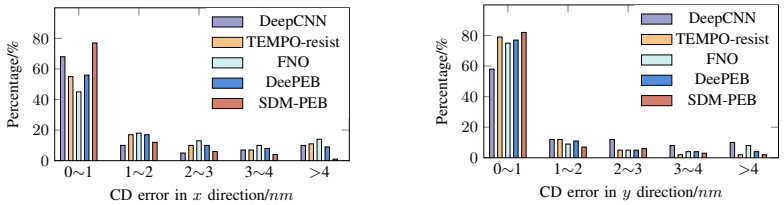
IV. Experiments

We generate 100 mask clips from [42], each sized at $2\times2\mu m^2$, which proved sufficient in our experiments. These mask clips are designed with contact sizes and distribution patterns suitable for technology nodes at $28nm$ and below. Rigorous simulations are conducted using S-Litho. The wavelength is $\lambda = 193nm$ and numerical aperture is $NA = 1.35$ for the optical exposure simulation, the optical influence range of $5\lambda/NA$ and simulation resolution of $0.5nm$ in three directions are configured for the $2\times2\mu m^2$ simulation window. For the PEB simulation and development simulation, the resist thickness is $80nm$, and resolutions in x, y and z directions are $2nm$, $2nm$, and $1nm$ respectively. Other important simulation parameters included are listed in TABLE I.

In the SDM-PEB configuration, the spatial sizes of feature maps from encoder layers 1 to 4 are reduced to $[\frac{1}{16}, \frac{1}{32}, \frac{1}{64}, \frac{1}{128}]$ of raw input, achieved using patch sizes of [15,3,3,3] and strides of [8,2,2,2]. The feature dimensions for these layers are [64, 128, 320, 512], respectively. Self-attention reduction ratios are [64,16,4,1]. The MLP in the feature fusion layer has 768 dimensions. The decoder includes 3 transpose convolution layers, with LeakyReLU activations between them. The training was conducted on 2 NVIDIA RTX 3090 GPUs for 500 epochs using a step decay scheduler, beginning at a learning rate of 0.03 with a step size of 100 and a decay factor of 0.7. We used a batch size of 8 by accumulating gradients over 8 clips before updating the model each epoch, considering the restriction of GPU memory. For a fair comparison, we followed the same train-test split as prior methods and tested runtime on our machines.

Compare With Learning-based PEB solvers. There exists few prior works focused on modeling photoresist simulation. We compared our SDM-PEB with previous state-of-the-art (SOTA) learning-based photoresist solvers. DeepCNN is customized from [41] with a residual connection for adaption to our problem. TEMPO-resist is modified from TEMPO [5], which originally used a conditional-GAN to predict 3D aerial images at various heights, to suit our 3D PEB simulation. DeePEB [15] extends FNO [19] by integrating CNN-based local learning branches to capture high-frequency information.

As is shown in TABLE II, our SDM-PEB outperforms all compared learning-based methods by a significant margin. Compared to the previous best method, DeePEB, SDM-PEB achieves 43.5% lower average RMSE and 54.1% lower NRMSE for inhibitor concentration predictions. For development rate prediction, with values bounded between R_{min} and R_{max}, our method reduces RMSE by 37.1% and



(a) (b)

Fig. 7 Percentage counts of CD errors using different methods: (a) error in the x direction and (b) error in the y direction.

TABLE III Ablation study

Methodologies	NRMSE/%		CD Error	
	Inhibitor	Rate	x/nm	y/nm
Single Layer Encoder	13.09	1.71	2.93	3.49
2-D Scan	8.83	1.58	2.07	3.05
w/o. Focal Loss	5.91	1.22	1.14	1.37
w/o. Regularization	5.98	1.24	1.15	1.42
SDM-PEB	**3.70**	**0.86**	**0.74**	**0.93**

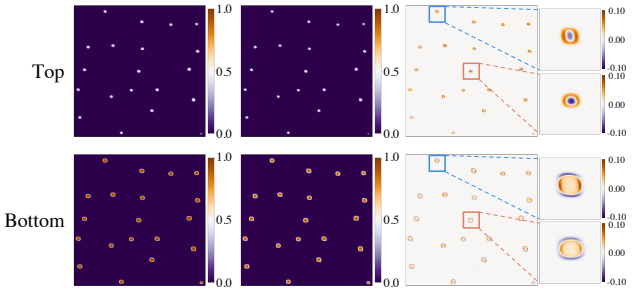


Fig. 8 Top-down visualization examples of predicted distribution results. The upper row is the top surface and the lower row is the bottom surface. (a) Ground truths, (b) predictions and (c) differences.

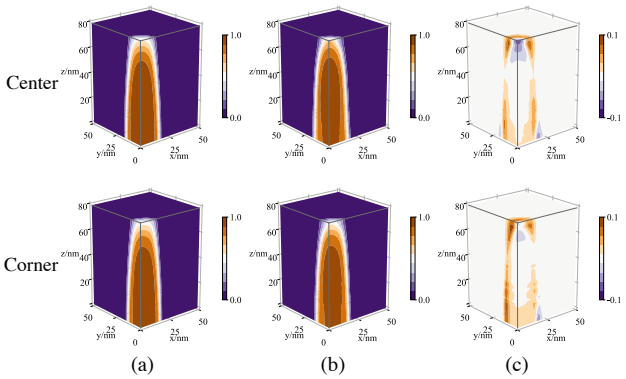


Fig. 9 Vertical visualization of predicted results: the upper row shows the center contact, the lower row shows the corner contact. (a) Ground truths, (b) predictions, (c) differences.

NRMSE by 38.4%. This superior predictive accuracy extends to the final resist profile, improving CD error by 32.4% in the x-direction and 33.3% in the y-direction. Compared to DeepCNN, TEMPO-resist, and FNO, the accuracy improvements are even more pronounced across all evaluated metrics.

The distributions of CD errors in the x and y directions are shown in Fig. 7. Compared to other methods, the CD errors of SDM-PEB are more concentrated within the range of 0-1 nm, demonstrating superior prediction accuracy. Furthermore, the CD error distributions in the x and y directions are more consistent, demonstrating the robustness and stability of our approach.

SDM-PEB also demonstrates high efficiency, achieving an average runtime (RT) of 1.06s, which is 138× faster than S-Litho's 147s. It achieves a 6× speedup compared to TEMPO-resist and reduces runtime by 8.5% and 29.2% compared to FNO and DeePEB, respectively. Although it takes 4.7% more time than DeepCNN, this slight runtime overhead is justified by its substantial performance improvements.

Visualization of Simulation Results. Fig. 8 presents examples comparing the ground truth with SDM-PEB results at both the top layer and bottom surface. The predicted inhibitor shows minimal deviation from the ground truth, with absolute errors across most positions on the plane remaining within 0.1. Spatial information is effectively extracted and translated, as evidenced by the consistency of results at both the center and the corner. Fig. 9 compares the vertical profiles of contacts at the center and corner, corresponding to contacts within the red box and blue box in Fig. 8. Leveraging the three-direction PEB selective scan for depthwise information gathering, SDM-PEB successfully captures relationships between adjacent depth distributions and models causal variations across different depths, achieving consistent simulations along the depth direction. The primary discrepancies occur at the edges of contacts, where concentration changes are more drastic. At the top layer, the inhibitor concentration is lower than at the bottom and is confined to a smaller region, adding to the challenges for the model.

Ablation Study. We conducted an ablation study on each component of our method, with the results presented in TABLE III. In the Single Layer Encoder setting, only the feature map from the first encoder layer is used, which fails to capture complex features and results in the lowest accuracy. In the 2-D scan setting, only the depth-forward and depth-backward scans shown in Fig. 5(b) are utilized, which is adapted from [24] for our 3D tasks. Compared to this setting, SDM-PEB achieves significant improvements across all metrics, demonstrating the importance of depth dependencies in enhancing PEB modeling. Furthermore, we remove the PEB focal loss and differential depth divergence regularization in the 'w/o. Focal Loss' and 'w/o. Regularization', respectively, while retaining the other two optimization objectives. The results show noticeable degradation compared to the full SDM-PEB, highlighting the critical role of each component in refining the framework.

V. Conclusion

This paper introduces SDM-PEB, a novel modeling framework designed to enhance 3D PEB simulation. It effectively captures spatial and depthwise dependencies to model the physical and chemical interactions underlying the inhibitor distributions while maintaining efficiency. Our framework integrates hierarchical feature extraction with efficient self-attention and spatial-depthwise Mamba-based attention to capture long-range dependencies. To address data imbalances and improve sensitivity to 3D variations, we propose a PEB focal loss and differential depth divergence regularization. Experimental results, validated with an industry-grade software S-Litho under real manufacturing conditions demonstrate the superior performance of our method. Through the development of modeling advanced photoresist simulation, we aim to further progress lithography techniques.

ACKNOWLEDGEMENTS

The project is supported in part by Research Grants Council of Hong Kong SAR (No. RFS2425-4S02, No. CUHK14211824 and No. CUHK14210723), and the MIND project (MINDXZ202404).

REFERENCES

[1] C. A. Mack, "Thirty years of lithography simulation," in *Optical Microlithography XVIII*, vol. 5754. SPIE, 2005, pp. 1–12.

[2] Z. Yu, G. Chen, Y. Ma, and B. Yu, "A gpu-enabled level-set method for mask optimization," *IEEE TCAD*, 2022.

[3] X. Zhang, S. Zheng, G. Chen, B. Zhu, H. Xu, and B. Yu, "Fracturing-aware curvilinear ilt via circular e-beam mask writer," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[4] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "Lithogan: End-to-end lithography modeling with generative adversarial networks," in *Proc. DAC*, 2019, pp. 1–6.

[5] W. Ye, M. B. Alawieh, Y. Watanabe, S. Nojima, Y. Lin, and D. Z. Pan, "Tempo: Fast mask topography effect modeling with deep learning," in *Proceedings of the 2020 International Symposium on Physical Design*, 2020, pp. 127–134.

[6] Z. Wang, Y. Shen, X. Yao, W. Zhao, Y. Bai, F. Farnia, and B. Yu, "Chatpattern: Layout pattern customization via natural language," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[7] Z. Yu, P. Liao, Y. Ma, B. Yu, and M. D. Wong, "CTM-SRAF: Continuous transmission mask-based constraint-aware sub resolution assist feature generation," *IEEE TCAD*, 2023.

[8] W. Zhao, X. Yao, S. Yin, Y. Bai, Z. Yu, Y. Ma, B. Yu, and M. D. Wong, "Adaopc 2.0: Enhanced adaptive mask optimization framework for via layers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[9] H. Yang, Z. Li, K. Sastry, S. Mukhopadhyay, M. Kilgard, A. Anandkumar, B. Khailany, V. Singh, and H. Ren, "Generic lithography modeling with dual-band optics-inspired neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 973–978.

[10] G. Chen, Z. Pei, H. Yang, Y. Ma, B. Yu, and M. Wong, "Physics-informed optical kernel regression using complex-valued neural fields," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[11] H. Sun, J. Du, C. Jin, H. Quan, Y. Li, Y. Tang, J. Wang, S. Hu, and J. Liu, "Global optimisation of source and mask in inverse lithography via tabu search combined with genetic algorithm," *Optics Express*, vol. 30, no. 14, pp. 24 166–24 185, 2022.

[12] G. Chen, Z. Wang, B. Yu, D. Z. Pan, and M. D. Wong, "Ultra-fast source mask optimization via conditional discrete diffusion," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[13] A. Erdmann, T. Fühner, F. Shao, and P. Evanschitzky, "Lithography simulation: modeling techniques and selected applications," *Modeling Aspects in Optical Metrology II*, vol. 7390, pp. 13–29, 2009.

[14] J. Byers *et al.*, "Characterization and modeling of a positive chemically amplified resist," in *Proceeding of SPIE*, vol. 2438, 1995, pp. 153–166.

[15] Q. Wang, X. Gao, Y. Lin, R. Wang, and R. Huang, "Deepeb: A neural partial differential equation solver for post exposure baking simulation in lithography," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[16] R. A. Ferguson, J. M. Hutchinson, C. A. Spence, and A. R. Neureuther, "Modeling and simulation of a deep-ultraviolet acid hardening resist," *Journal of Vacuum Science & Technology B: Microelectronics Processing and Phenomena*, vol. 8, no. 6, pp. 1423–1427, 1990.

[17] M. Weiss, H. Binder, and R. Schwalm, "Modeling and simulation of a chemically amplified duv resist using the effective acid concept," *Microelectronic engineering*, vol. 27, no. 1-4, pp. 405–408, 1995.

[18] E. Tomacruz, M. Zuniga, R. Guerrieri, A. Neureuther, and A. Sangiovanni-Vicentelli, "3-d diffusion models for chemically-amplified resists using massively parallel processors," in *Simulation of Semiconductor Devices and Processes: Vol. 5*. Springer, 1993, pp. 109–112.

[19] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2020.

[20] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[21] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *Advances in neural information processing systems*, vol. 34, pp. 12 077–12 090, 2021.

[22] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[23] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[24] L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang, "Vision mamba: Efficient visual representation learning with bidirectional state space model," *arXiv preprint arXiv:2401.09417*, 2024.

[25] M. D. Stewart, K. Patterson, M. H. Somervell, and C. G. Willson, "Organic imaging materials: a view of the future," *Journal of Physical Organic Chemistry*, vol. 13, no. 12, pp. 767–774, 2000.

[26] F. H. Dill, W. P. Hornberger, P. S. Hauge, and J. M. Shaw, "Characterization of positive photoresist," *IEEE Transactions on electron devices*, vol. 22, no. 7, pp. 445–452, 1975.

[27] C. A. Mack, *Modeling solvent effects in optical lithography*. The University of Texas at Austin, 1998.

[28] M. D. Smith and C. A. Mack, "Examination of a simplified reaction-diffusion model for post-exposure bake of chemically amplified resists," in *Advances in Resist Technology and Processing XVIII*, vol. 4345. SPIE, 2001, pp. 1022–1036.

[29] C. A. Mack, M. J. Maslow, A. Sekiguchi, and R. A. Carpio, "New model for the effect of developer temperature on photoresist dissolution," in *Advances in Resist Technology and Processing XV*, vol. 3333. SPIE, 1998, pp. 1218–1231.

[30] C. Mack, *Fundamental principles of optical lithography: the science of microfabrication*. John Wiley & Sons, 2007.

[31] W.-K. Jeong and R. T. Whitaker, "A fast iterative method for eikonal equations," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2512–2534, 2008.

[32] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[33] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," *Advances in neural information processing systems*, vol. 33, pp. 1474–1487, 2020.

[34] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state space layers," *Advances in neural information processing systems*, vol. 34, pp. 572–585, 2021.

[35] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 568–578.

[36] H. Cao, Y. Wang, J. Chen, D. Jiang, X. Zhang, Q. Tian, and M. Wang, "Swin-unet: Unet-like pure transformer for medical image segmentation," in *European conference on computer vision*. Springer, 2022, pp. 205–218.

[37] Y. Liu, Y. Tian, Y. Zhao, H. Yu, L. Xie, Y. Wang, Q. Ye, and Y. Liu, "Vmamba: Visual state space model," 2024.

[38] C. Yang, Z. Chen, M. Espinosa, L. Ericsson, Z. Wang, J. Liu, and E. J. Crowley, "Plainmamba: Improving non-hierarchical mamba in visual recognition," *arXiv preprint arXiv:2403.17695*, 2024.

[39] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural networks*, vol. 107, pp. 3–11, 2018.

[40] T.-Y. Ross and G. Dollár, "Focal loss for dense object detection," in *proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2980–2988.

[41] Y. Watanabe, T. Kimura, T. Matsunawa, and S. Nojima, "Accurate lithography simulation model based on convolutional neural networks," in *Optical Microlithography XXX*, vol. 10147. SPIE, 2017, pp. 137–145.

[42] Y. Lin, M. Li, Y. Watanabe, T. Kimura, T. Matsunawa, S. Nojima, and D. Z. Pan, "Data efficient lithography modeling with transfer learning and active data selection," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 10, pp. 1900–1913, 2018.

HybriMoE: Hybrid CPU-GPU Scheduling and Cache Management for Efficient MoE Inference

Shuzhang Zhong[1,2], Yanfan Sun[5], Ling Liang[2], Runsheng Wang[2,3,4], Ru Huang[2,3,4], Meng Li[1,2,4*]
[1]Institute for Artificial Intelligence, Peking University, Beijing, China
[2]School of Integrated Circuits, Peking University, Beijing, China
[3]Institute of Electronic Design Automation, Peking University, Wuxi, China
[4]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China
[5]School of Computer Science and Engineering, Beihang University, Beijing, China

***Abstract*—The Mixture of Experts (MoE) architecture has demonstrated significant advantages as it enables to increase the model capacity without a proportional increase in computation. However, the large MoE model size still introduces substantial memory demands, which usually requires expert offloading on resource-constrained platforms and incurs significant overhead. Hybrid CPU-GPU inference has been proposed to leverage CPU computation to reduce expert loading overhead but faces major challenges: on one hand, the expert activation patterns of MoE models are highly unstable, rendering the fixed mapping strategies in existing works inefficient; on the other hand, the hybrid CPU-GPU schedule for MoE is inherently complex due to the diverse expert sizes, structures, uneven workload distribution, etc. To address these challenges, in this paper, we propose HybriMoE, a hybrid CPU-GPU inference framework that improves resource utilization through a novel CPU-GPU scheduling and cache management system. HybriMoE introduces (i) a dynamic intra-layer scheduling strategy to balance workloads across CPU and GPU, (ii) an impact-driven inter-layer prefetching algorithm, and (iii) a score-based caching algorithm to mitigate expert activation instability. We implement HybriMoE on top of the kTransformers framework and evaluate it on three widely used MoE-based LLMs. Experimental results demonstrate that HybriMoE achieves an average speedup of 1.33× in the prefill stage and 1.70× in the decode stage compared to state-of-the-art hybrid MoE inference framework. Our code is available at: https://github.com/PKU-SEC-Lab/HybriMoE.**

I. Introduction

Mixture of Experts (MoE) has emerged as a promising solution to enhance computational efficiency of Large Language Models (LLMs) without compromising model performance [1], [2]. By employing dynamic routing functions that allocate input tokens to a subset of experts, MoE enables the scaling of LLM parameters and capabilities without a proportional increase in computational demands.

Despite its advantages, MoE introduces significant memory requirements, which pose a particular challenge for deployment on edge devices with limited memory resources. To mitigate this issue, expert offloading techniques store expert weights in secondary storage, such as CPU memory or SSDs, and load them into GPU memory through PCIE on demand [3]. In such offloading scenarios, the primary bottleneck becomes the overhead associated with on-demand loading, driven by the large communication scale and limited bandwidth. To mitigate this problem, several studies have explored quantization, prefetching or caching techniques to reduce latency [4]–[7].

Previous works in other offloading scenarios have further explored leveraging CPU computation to reduce the frequency of memory transfers [8], [9]. Techniques such as PowerInfer [10] and Caraserve [11] have achieved notable success by exploiting activation patterns or optimizing adapter usage during inference. Similarly, MoE-specific offloading approaches, including Fiddler [12] and kTransformers [13], utilize the CPU to execute expert layers during cache misses. As illustrated in Figure 1, when a cache miss occurs, the CPU processes the corresponding expert computation instead of transferring the layer to the GPU, reducing data transfer overhead.

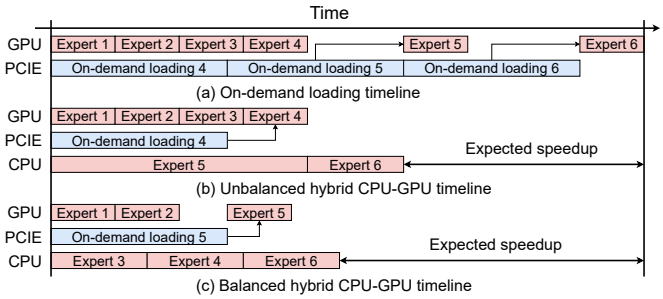


Fig. 1. Execution timeline of three scenarios. Expert computation time on the GPU remains constant, while CPU execution time increases linearly with workload. The balanced scheduling in (c) achieves improved utilization and reduces overall execution time.

While CPU computation is effective for traditional inference tasks, MoE models present unique challenges that complicate their application. Expert activations in MoE models are typically less skewed and exhibit significant variability across iterations, making it difficult to predict which experts will be activated [6]. This dynamic behavior complicates the balancing of workloads between CPU and GPU, as static task allocation strategies fail to adapt to real-time changes in workload distribution. However, existing solutions rely on **fixed mapping strategies** based on historical activation frequencies, neglecting the dynamic and unpredictable nature of MoE inference. These limitations result in suboptimal resource utilization and increased inference latency as illustrated in figure 1(b) and (c).

In light of these challenges and opportunities, we propose **HybriMoE**, a hybrid CPU-GPU scheduling and cache management system to improve the efficiency of MoE inference. Reducing latency in hybrid systems requires maximizing hardware resource utilization, which depends on effective task-to-hardware mapping. However, the dynamic nature of MoE models poses significant challenges to designing optimal mapping strategies. To address this, HybriMoE introduces a comprehensive optimization framework to improve mapping efficiency through three key directions: (i) intra-layer hybrid scheduling, (ii) inter-layer prefetching, and (iii) inter-iteration cache

This work was supported in part by NSFC under Grant 62495102 and Grant 92464104, in part by the National Key Research and Development Program under Grant 2024YFB4505004, in part by Beijing Municipal Science and Technology Program under Grant Z241100004224015, and in part by 111 Project under Grant B18001.
*Corresponding author: meng.li@pku.edu.cn

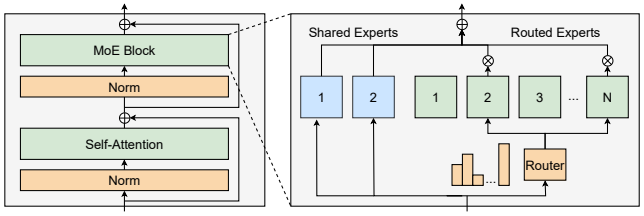


Fig. 2. An example of MoE architecture with shared and routed experts.

management. The key contributions of HybriMoE are as follows:

- **Hybrid MoE CPU-GPU Scheduling.** An efficient hybrid scheduling algorithm for MoE inference that dynamically balances workloads across GPUs and CPUs, optimizing resource utilization and minimizing latency through prioritized task execution and data transfer management.
- **Impact-driven prefetching.** A prefetching mechanism that simulates the potential impact of preloading experts from subsequent layers and prioritizes those with the higher expected gains.
- **MoE-specialized Cache Management.** An expert score-based caching strategy that prioritizes high-demand experts across layers to minimize cache misses.
- **System Implementation.** We implement HybriMoE on top of ktransformers framework. We evaluate HybriMoE on three popular MoE-based LLMs and various platforms. Compared to existing hybrid scheduling methods, HybriMoE achieves 1.33× and 1.70x speedup on prefill and decode stages respectively.

II. Background

A. Mixture-of-Experts

Mixture-of-Experts (MoE) models offer an efficient solution for handling the computational demands of LLMs by activating only a subset of experts [2], [14]–[16]. Unlike traditional dense networks, MoE models use a gating function G to select which experts process a given input token, reducing the number of active parameters and improving computational efficiency. Given an input x and N experts $E_0, ...E_{N-1}$ the output y of the MoE layer can be expressed as:

$$y = \sum_{i=0}^{N-1} \text{Softmax}(\text{TopK}(x \cdot W_g))_i E_i(x) \quad (1)$$

The total number of experts N and the number of activated experts K vary among different MoE implementations. For instance, the Mixtral model employs 8 experts, with only 2 being active at a time [17]. In contrast, DeepSeek utilizes 64 experts, activating 6 at once. This larger, finer-grained expert pool allows for greater specialization and more efficient knowledge acquisition [18], [19]. Additionally, as shown in figure 2, DeepSeek employs a shared expert strategy, where a subset of experts—known as shared experts—are activated for all tokens. The original experts are defined as routed experts. This reduces redundancy among experts, ensuring efficient processing by minimizing unnecessary computational overlap, thus enhancing overall model efficiency.

B. Efficient MoE Offloading

Parameter-offloading techniques have been proposed to address the significant memory requirements of large language models (LLMs) [20], [21]. However, these techniques are primarily designed for dense models and involve loading or prefetching all parameters, leading to unnecessary communication overhead. To accommodate the sparse activation patterns in Mixture-of-Experts (MoE) models, several specialized techniques have been introduced, including advanced gating, prefetching, and quantization strategies [3]–[7], [22]–[26]. These methods aim to minimize the on-demand loading overhead, reducing unnecessary memory transfers and improving overall performance.

TABLE I
COMPARISON WITH EXISTING OFFLOADING WORKS.

	Offload Granularity	CPU Computation	Dynamic Mapping	Cache Optimization
Powerinfer	Neuron	Decode	✓	LFU
llama.cpp	Layer	Prefill+Decode	×	LFU
AdapMoE	Expert	×	✓	LRU
KTrans	Expert	Decode	×	LFU
Ours	Expert	Prefill+Decode	✓	Score-Aware

C. Hybrid CPU-GPU Scheduling

Previous offloading techniques have primarily focused on reducing memory transfer overhead by offloading certain computations to the CPU [9]. For instance, PowerInfer [10] reduces GPU memory demand by executing less frequently activated neurons on the CPU, taking advantage of skewed activation patterns. Caraserve [11] addresses cold-start delays in LoRA serving by utilizing CPU assistance and employing rank-aware scheduling to reduce latency. These methods are effective in scenarios where activations are skewed or tasks have long periods of parameter reuse.

In the context of MoE models, techniques like Fiddler [12] and kTransformers [13] extend this concept by offloading expert layer computation to the CPU during cache misses. Specifically, when an expert is not in the GPU cache, the CPU executes the corresponding expert layer instead of loading it from memory. These approaches aim to optimize memory usage by exploiting CPU-GPU parallelism and mitigating the overhead of loading large models onto the GPU.

In table I, we compare HybriMoE with prior-art works qualitatively. As can be observed, HybriMoE features CPU-GPU hybrid scheduling to improve the efficiency of both prefill and decode stages.

III. Motivation

The primary bottleneck in existing hybrid CPU-GPU scheduling for MoE inference is the **suboptimal resource utilization**. To address this issue, we begin by analyzing the main challenges of finding an efficient **mapping strategy**.

Challenge 1: High Instability of MoE Activation Patterns. In existing hybrid CPU-GPU scheduling research, both sparse models with highly skewed activations, like PowerInfer, and dense models (or LoRA inference) exhibit relatively stable activation patterns. In these models, activation is either concentrated on a few 'hot' neurons or remains consistent over time, making scheduling and workload balancing easier. In contrast, MoE models have unpredictable activation patterns, with experts being activated in a dynamic and frequently changing manner. As shown in figure 3(a), compared with neuron-level sparsity, the activation frequency of MoE is more evenly distributed, making it challenging to predict the future expert usage. This lack of stability makes it difficult to determine an optimal CPU-GPU scheduling strategy in advance, leading to suboptimal resource utilization and inefficiency.

Opportunity 1: MoE-specific Cache and Prefetch Optimization. Despite the instability of MoE activations, certain temporal and structural patterns present opportunities for optimization. The temporal correlation of expert activation provides a basis for cache

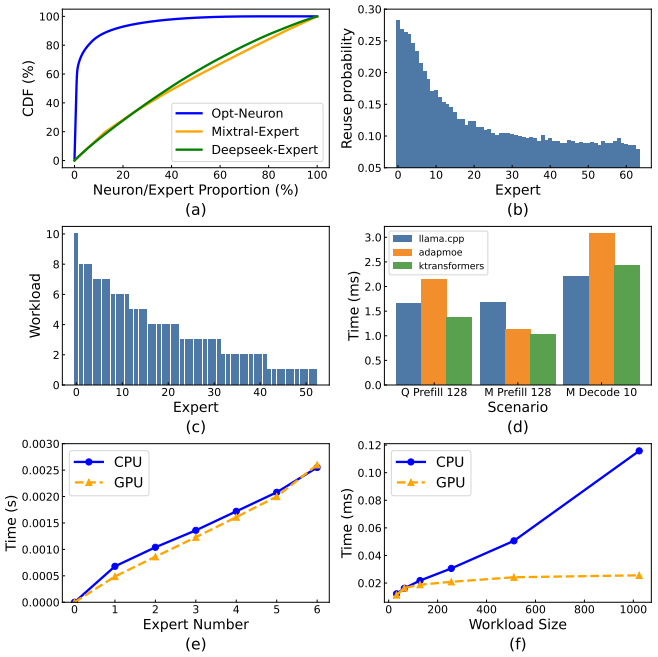


Fig. 3. (a) Cumulative activation frequency(CDF) for neurons and experts, (b) Reuse probability of experts by score, suggesting cache optimization opportunities, (c) Expert workload distribution of DeepSeek in a prefill forward, (d) Latency of prefill 128 tokens for Qwen2(Q), Mixtral(M) and decode 10 tokens for Mixtral with three existing methods, (e) CPU vs. GPU time for varying numbers of experts at fixed load, with CPU benefiting from overlapping computations. (f) CPU and GPU time across workload sizes.

optimization: experts with higher activation scores are more likely to be reused in the next iteration as shown in Figure 3(b), suggesting that retaining high-score experts in cache can reduce access latency. Additionally, MoE models often exhibit high activation similarity between adjacent layers, which can be leveraged for prefetching. These MoE-specific optimizations provide a promising approach to reducing the challenges posed by the dynamic nature of expert activation.

Challenge 2: Complexity of MoE Structure and Dynamic Scheduling. Minimizing latency in MoE inference requires maximizing hardware utilization, but existing fixed-mapping methods often lead to load imbalances and underutilized resources. The scheduling complexity is further increased by the diverse structures of MoE models, with variations in shared expert usage, expert size and number, and runtime cache behavior. Additionally, uneven load distribution and variable execution order in the prefill stage make efficient scheduling even more challenging as shown in figure 3(c). Given the need for layer-by-layer adjustments, a static optimal solution is impractical, making real-time scheduling a significant challenge. As illustrated in figure 3(d), the performance of three existing strategies vary in different stages and models.

Opportunity 2: MoE-specific scheduling rules. Despite the NP-Hard nature of the scheduling problem, in the specific context of MoE inference on CPU-GPU systems, several key observations can guide the design of efficient scheduling rules. First, expert transfer times remain relatively constant, simplifying decision-making. Additionally, GPU computation time scales linearly with the number of activated experts, while CPU computation benefits from overlapping memory access and computation due to its larger cache. As

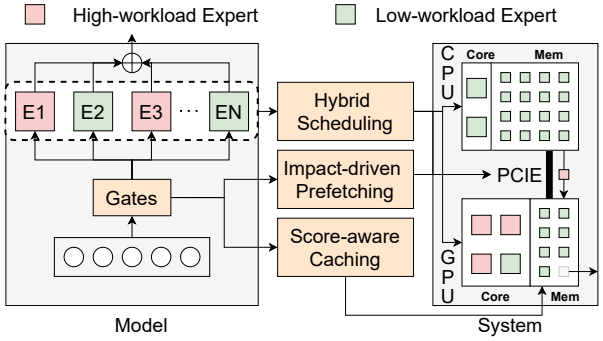


Fig. 4. Overview of HybriMoE.

shown in Figure 3(e), the first expert computation on the CPU is slower, but subsequent tasks are processed faster with better cache utilization. Similarly, Figure 3(f) shows that GPU time remains stable with increasing workload, whereas CPU time grows linearly with workload. Leveraging these patterns, predefined scheduling rules can help achieve efficient workload balancing for MoE models.

IV. HybriMoE Design

A. Overview

This paper introduces HybriMoE, a CPU-GPU hybrid scheduling system tailored for MoE inference on memory-limited devices. HybriMoE addresses the challenges of unbalanced hardware utilization caused by the dynamic activation patterns and structural complexity of MoE models. The system incorporates three key techniques: (i) an efficient hybrid scheduling algorithm that dynamically distributes workloads between GPUs and CPUs, (ii) a score-based expert caching strategy that prioritizes high-demand experts to minimize cache misses, and (iii) an impact-driven prefetching mechanism that predicts and preloads high-demand experts, further enhancing resource utilization and reducing latency.

Figure 4 illustrates the overview of HybriMoE. The system begins with a warmup phase to collect essential performance metrics, such as CPU and GPU processing speeds and data transfer latency. During inference, HybriMoE leverages this information to implement hybrid CPU-GPU scheduling, score-aware caching, and impact-driven prefetching, ensuring efficient task execution and optimized resource usage throughout the inference process.

B. Hybrid Scheduling Strategy

The scheduling problem in MoE inference is inherently complex due to the dynamic nature of expert activation and the need to balance workloads across heterogeneous resources. To address these challenges, HybriMoE proposes a hybrid scheduling strategy that simplifies the task-to-hardware mapping by introducing three key priority rules:

- **GPU Priority**: The GPU prioritizes the computation of cached experts, executing higher-load experts first.
- **CPU Priority**: The CPU prioritizes the computation of uncached experts, focusing on lower-load tasks for efficient execution. Additionally, the CPU can process cached experts when the CPU is idle, following a low-to-high load order.
- **Transfer Priority**: The CPU-GPU transfer mechanism prioritizes the movement of high-load uncached experts from CPU to GPU to minimize computation delays.

These rules constrain the ordering of experts on devices, simplifying the scheduling problem into an allocation problem:

$$\arg\min_{cpu_expert, gpu_expert} \max(CPU_{TIME}(cpu_expert), GPU_{TIME}(gpu_expert)) \quad (2)$$

This formulation does not account for the finish time of data transfers, as expert loading must be completed before GPU computation begins.

Based on these priority rules, HybriMoE divides all activated experts into a GPU queue and a CPU queue. The GPU queue contains cached experts on the GPU, sorted by load in descending order. The CPU queue contains uncached experts on the CPU, sorted by load in ascending order.

Before the actual execution, HybriMoE performs a simulation phase to evaluate scheduling strategies and identify an efficient task allocation plan tailored to the specific workload. This simulation approximates the execution process by iteratively filling the CPU computation, GPU computation, and data transferring timelines, enabling the system to determine a scheduling configuration that minimizes overall latency while balancing resource utilization across heterogeneous hardware.

During each step of the simulation, the system selects the timeline with the earliest completion time and executes the corresponding operation—either a computation task on the CPU or GPU, or a data transfer via PCIE. Task selection adheres to the scheduling priorities: the GPU prioritizes high-load cached experts, the CPU focuses on low-load uncached experts and, when its queue is empty, processes low-load cached experts from the GPU queue, while PCIE prioritizes high-load uncached experts for faster availability on the GPU.

If an expert is transferred from the CPU to the GPU, it is inserted into the GPU queue in descending order of load, ensuring high-load tasks are prioritized for GPU computation. This iterative simulation continues until all experts are computed, effectively modeling the execution process and testing different scheduling strategies.

The scheduling process is illustrated through an example in figure 5. In this scenario, the GPU computation time is assumed to be constant, the CPU computation time is proportional to the expert's load, and the transmission time is fixed at 3 units. The scheduling algorithm in HybriMoE identifies an optimal strategy where the CPU computes the cached expert E while the GPU processes the uncached expert C, effectively improving hardware utilization by balancing workloads across resources.

C. Impact-driven prefetching

Due to the residual connections in LLMs, hidden states across consecutive layers exhibit a high degree of similarity, making expert prefetching an effective method to optimize resource utilization. While several existing works have adopted prefetching mechanisms, none of them discuss the critical trade-offs involved when multiple subsequent layers' experts can be prefetched. Specifically, these works do not explore how to strategically decide which layer's experts should be prioritized for prefetching to maximize resource efficiency.

Inspired by the scheduling algorithm in IV-B, we propose impact-driven prefetching. Before executing a prefetch, the system performs a simulation to evaluate the potential gains of prefetching specific experts. This simulation estimates the impact of preloading a given expert on overall scheduling efficiency, allowing the algorithm to prioritize experts that yield the highest resource utilization improvements. The greedy nature of this simulation ensures minimal computational overhead, making it practical for real-time inference scenarios.

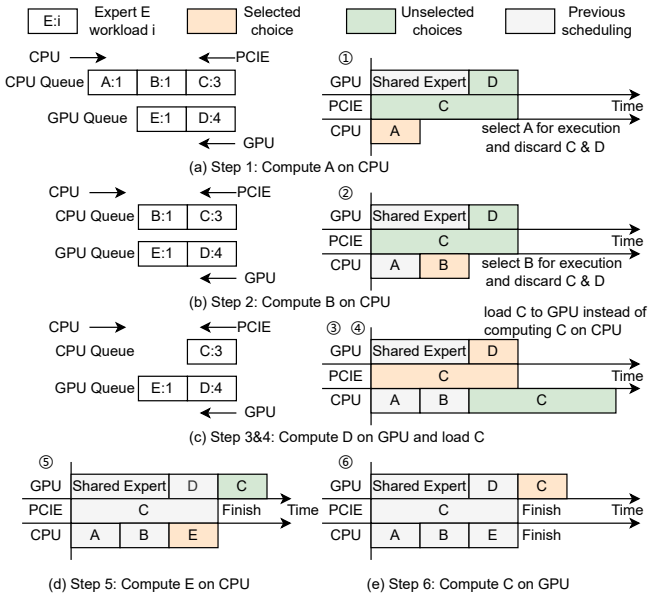


Fig. 5. An example of hybrid scheduling. The CPU computes the cached expert E while the GPU computes the uncached expert C to achieve better hardware utilization.

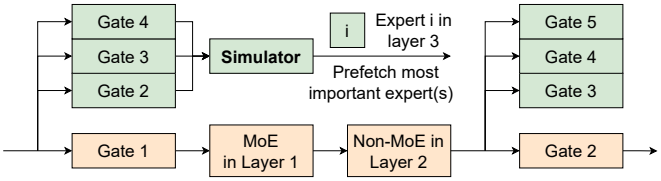


Fig. 6. Impact-driven prefetch workflow.

Specifically, HybriMoE predicts expert activations for the next three layers by reusing the gating information from those layers as illustrated in figure 6. This prediction guides the prefetching mechanism, enabling the system to efficiently preload experts likely to be activated in subsequent computations.

D. Score-aware Caching

Traditionally, the Least Frequently Used (LFU) and Least Recently Used (LRU) algorithms have been employed for MoE cache management. However, these strategies fail to account for the specific activation patterns observed in MoE models, where expert scores provide valuable predictive signals for future activations. As discussed in III, not only are currently activated experts likely to be reused in the future, but experts with high scores that were not activated also exhibit a higher probability of being selected in subsequent iterations.

To leverage this insight, we propose Score-Aware Caching, a novel cache replacement strategy tailored for MoE models. Specifically, we introduce the **Minus Recent Score (MRS)** replacement policy, which prioritizes retaining experts based on their routing scores.

Define s as the routing scores of all experts in the current iteration, S as the estimated priority score, α as the averaging coefficient, the update of the estimated priority can be expressed as :

$$S = \alpha \times TopP(s) + (1 - \alpha) \times S \quad (3)$$

Here, only the top p expert scores will be accumulated. This is derived from the observation in figure 3(b) that the reuse probability

TABLE II
CONFIGURATION OF THREE EVALUATED MOE MODELS.

	Mixtral	Qwen2	DeepSeek
#Layers	32	28	26
#Shared Experts	0	1	2
#Routed Experts	8	64	64
#Activated Experts	2	8	6
Shared Expert Size	/	(3584, 20480)	(2048, 1408)
Routed Expert Size	(4096, 14336)	(3584, 18944)	(2048, 1408)

of experts with lower scores does not exhibit significant differences. Typically, we set p to twice the number of activated experts.

V. SYSTEM IMPLEMENTATION

We implement the HybriMoE system on top of the kTransformers framework and llama.cpp kernels. KTransformers provides a flexible infrastructure for kernel injection, enabling seamless support for hybrid CPU-GPU execution. To optimize the system workflow, we incorporate parallel execution across CPU, GPU, and PCIe transfers, utilizing fine-grained CUDA stream scheduling for efficient resource management. Additionally, we modify the C++ kernels to handle expert computation task allocation directly, minimizing redundant Python overhead and improving execution efficiency. For quantization, we leverage Marlin quantization, a state-of-the-art 4-bit quantization kernel from llama.cpp, to significantly enhance computational efficiency and reduce memory usage.

VI. EXPERIMENTAL RESULTS

A. Experimental Setup

1) Platforms: We evaluate HybriMoE on the NVIDIA RTX A6000. For the CPU, we utilize an Intel Xeon Gold 5220R processor, restricting usage to 10 cores to simulate real-world edge deployment scenarios with limited resources. To assess the system's performance and scalability under varying hardware configurations, we adjust the upper bound of the GPU expert cache ratio.

2) Models: We evaluate our system using three widely adopted MoE models with distinct characteristics: Mixtral-8x7B-Instruct [17] (Mixtral), DeepSeek-V2-Lite-Chat [18] (DeepSeek), and Qwen2-57B-A14B-Instruct [16] (Qwen2). As summarized in Table II, these models differ in the number and size of experts, as well as their architectural configurations. Mixtral represents MoE models with a smaller number of larger experts, while Qwen2 and DeepSeek are representative of models with a larger number of smaller experts. Notably, Qwen2 and DeepSeek also incorporate shared experts, which are activated for all input tokens.

3) Baselines: We evaluate HybriMoE against three representative open-source MoE inference frameworks: llama.cpp [27], AdapMoE [5], and kTransformers [13], each representing a distinct scheduling approach. llama.cpp is a CPU-GPU hybrid scheduling baseline that statically maps model layers to CPU or GPU. AdapMoE is the SOTA for GPU-centric MoE scheduling, minimizing on-demand loading overhead through adaptive prefetching and caching. kTransformers is the SOTA for CPU-GPU hybrid MoE scheduling, mapping high-activation-frequency experts (e.g., shared experts) to the GPU to maximize efficiency.

4) Metrics: Auto-regressive decoding consists of two stages: the prefill stage and the decoding stage. We evaluate the performance of HybriMoE separately for these two stages. For the prefill stage, we use Time To First Token (TTFT), which measures the latency from receiving the input prompt to generating the first token. For the decoding stage, we use Time Between Tokens (TBT), which captures the time taken to generate each subsequent token. These metrics provide a clear assessment of both initial latency and sustained efficiency during inference.

5) Datasets: For the prefill stage, we evaluate performance under varying input lengths by sampling traces of different lengths from multiple datasets, including MT Bench [28], Vicuna Bench [29] and ChatGPT Prompts [30]. In contrast, for the decoding stage, as performance is not sensitive to input length, we use only the ChatGPT Prompts dataset to evaluate the TBT metric.

TABLE III
MOE INFERENCE SPEEDUP BREAKDOWN OF PROPOSED TECHNIQUES.

	Technique	Latency(s)	Speedup
Prefill	Baseline	1.47	
	Baseline+Scheduling	1.17	1.26×
	Baseline+Prefetching	1.39	1.06×
	All	1.13	1.31×
Decode	Baseline	0.21	
	Baseline+Scheduling	0.14	1.46×
	Baseline+Prefetching	0.18	1.15×
	Baseline+Caching	0.15	1.38×
	All	0.11	1.86×

B. End-to-End Performance

1) Prefill Stage: We evaluate the prefill stage performance of HybriMoE by comparing it against three baselines: llama.cpp, AdapMoE, and kTransformers. Figure 7 presents the TTFT results across various input legnths(around 32, 128, 512 and 1024 tokens) and different GPU expert cache ratios(25%, 50% and 75%).

HybriMoE demonstrates consistent improvements over the baselines across all input lengths and cache configurations. llama.cpp exhibits significantly higher prefill latency due to its naive static mapping strategy, which allocates entire layers of experts to the CPU. This approach fails to balance workloads effectively, particularly in the prefill stage where computational demand is high, leading to substantial delays. Compared to kTransformers, HybriMoE achieves an average speedup of **1.33**× across different input lengths and cache configurations. This improvement is driven by HybriMoE's hybrid scheduling and impact-driven prefetching mechanisms, which dynamically balance workloads and reduce cache misses, enabling more efficient resource utilization.

2) Decode Stage: Figure 8 illustrates the decode performance results for three MoE models. HybriMoE consistently achieves the highest throughput across all cache ratios and models, demonstrating its ability to dynamically balance workloads and fully utilize hardware resources during the decode stage. Compared to kTransformers, HybriMoE achieves an average throughput improvement of **1.70**×. Also, it is worth noting that llama.cpp demonstrates relatively strong performance in this stage, especially compared to its prefill stage results. This is primarily due to the smaller computational load per expert in the decode stage, which allows CPU-based computation to proceed faster. Additionally, the impact of uneven expert mapping is less pronounced compared to the prefill stage, and the overall resource overhead remains low, favoring llama.cpp's static scheduling strategy in this specific context.

C. Ablation Study

We further explore how the components of our method contribute to the result. Performance was measured for Qwen2 under 25%

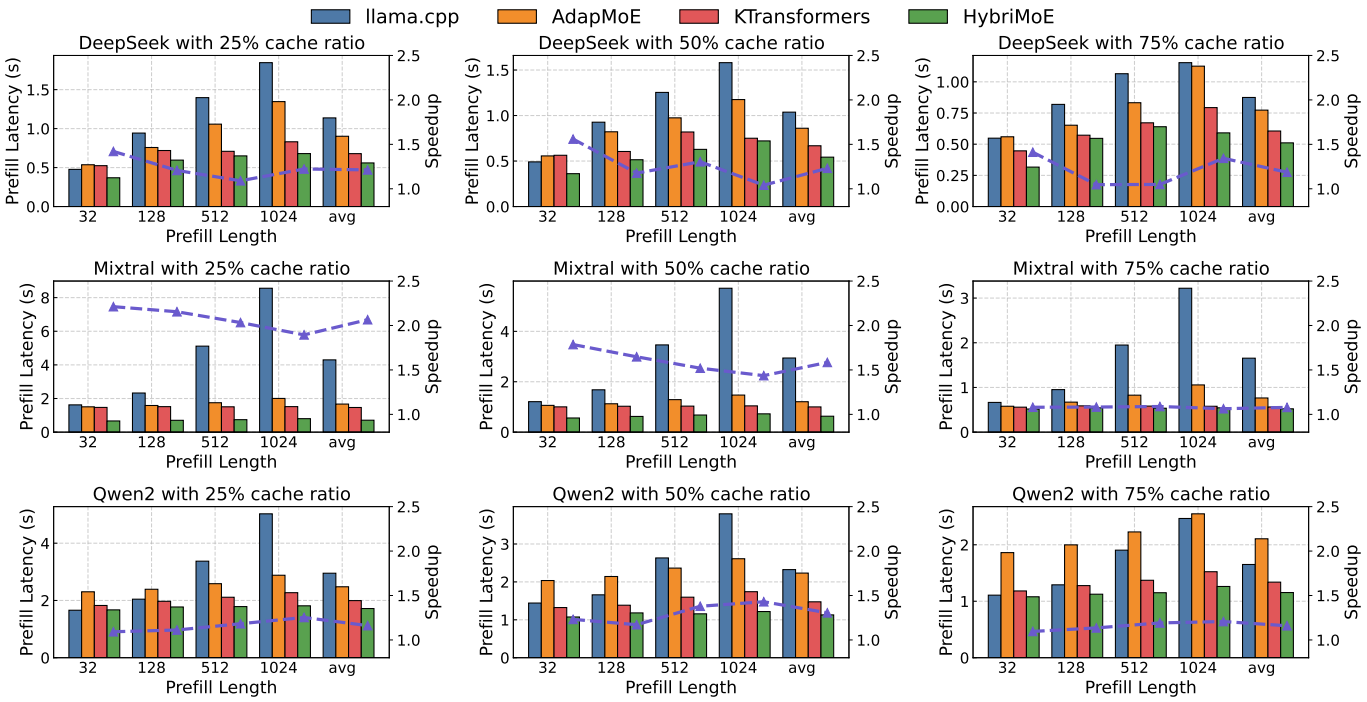


Fig. 7. Prefill stage performance comparison across different input lengths and cache ratios, highlighting relative speedups against kTransformers.

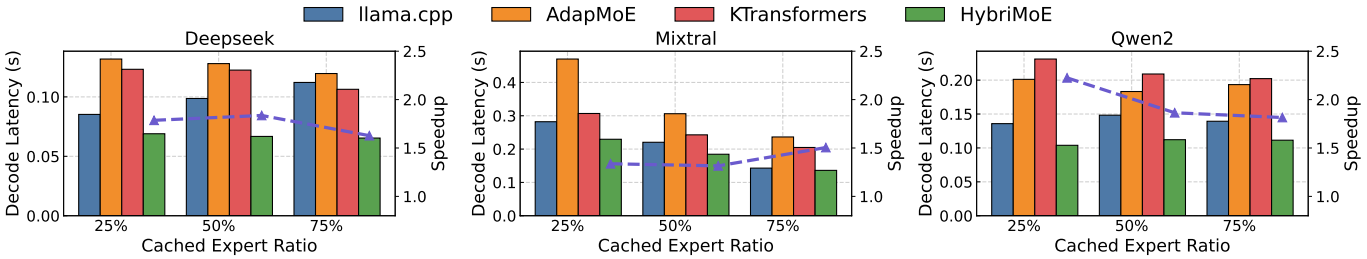


Fig. 8. Decode stage performance comparison across different cache ratios.

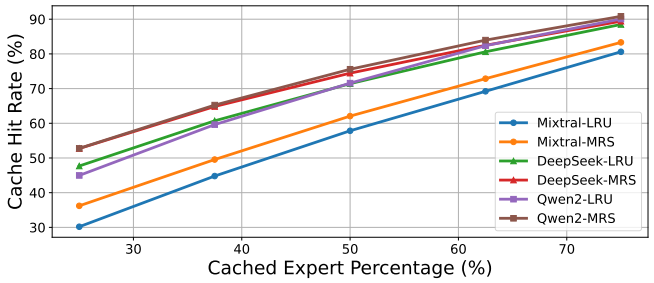


Fig. 9. Cache Hit Rate Comparison Between MRS and LRU Across Different Cached Expert Percentages.

expert cache ratio for the two stages. The baseline is ktransformers framework. The results are illustrated in table III.

D. Discussions

1) Score-aware Cache Management Analysis: Figure 9 compares the cache hit rates of HybriMoE's Minus Recent Score (MRS) strategy and the traditional Least Recently Used (LRU) strategy across three models under varying cached expert percentages. At 25% cache capacity, MRS outperforms LRU by 6% to 8%, with Mixtral improving from 30.2% to 36.2%, DeepSeek from 47.7% to 52.7%, and Qwen2 from 45.0% to 52.8%. As cache capacity increases to 75%, the gap narrows (e.g., Mixtral: 83.3% vs. 80.6%), as higher capacities reduce expert competition, diminishing the relative impact of the caching strategy. These results highlight MRS's effectiveness, particularly under limited cache settings.

VII. CONCLUSION

This paper presents HybriMoE, a hybrid CPU-GPU scheduling and cache management system designed to address the challenges of MoE inference, including dynamic expert activations and workload imbalances. By incorporating dynamic intra-layer scheduling, impact-driven prefetching, and score-aware caching, HybriMoE achieves efficient resource utilization and reduced latency. Experiments on various MoE models demonstrate that HybriMoE achieves an average speedup of 1.33x in prefill latency and 1.70x in decode latency compared to state-of-the-art methods. These results highlight HybriMoE's effectiveness in optimizing hybrid MoE inference and its potential for scalable deployment on resource-constrained devices.

REFERENCES

[1] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[2] S. Masoudnia and R. Ebrahimpour, "Mixture of experts: a literature survey," *Artificial Intelligence Review*, vol. 42, pp. 275–293, 2014.

[3] A. Eliseev and D. Mazur, "Fast inference of mixture-of-experts language models with offloading," *arXiv preprint arXiv:2312.17238*, 2023.

[4] R. Hwang, J. Wei, S. Cao, C. Hwang, X. Tang, T. Cao, M. Yang, and M. Rhu, "Pre-gated moe: An algorithm-system co-design for fast and scalable mixture-of-expert inference," *arXiv preprint arXiv:2308.12066*, 2023.

[5] S. Zhong, L. Liang, Y. Wang, R. Wang, R. Huang, and M. Li, "Adapmoe: Adaptive sensitivity-based expert gating and management for efficient moe inference," *arXiv preprint arXiv:2408.10284*, 2024.

[6] X. Song, Z. Zhong, and R. Chen, "Promoe: Fast moe-based llm serving using proactive caching," *arXiv preprint arXiv:2410.22134*, 2024.

[7] P. Tang, J. Liu, X. Hou, Y. Pu, J. Wang, P.-A. Heng, C. Li, and M. Guo, "Hobbit: A mixed precision expert offloading system for fast moe inference," *arXiv preprint arXiv:2411.01433*, 2024.

[8] J. You, J. Wu, X. Jin, and M. Chowdhury, "Ship compute or ship data? why not both?" in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 633–651.

[9] D. Park and B. Egger, "Improving throughput-oriented llm inference with cpu computations," in *Proceedings of the 2024 International Conference on Parallel Architectures and Compilation Techniques*, 2024, pp. 233–245.

[10] Y. Song, Z. Mi, H. Xie, and H. Chen, "Powerinfer: Fast large language model serving with a consumer-grade gpu," *arXiv preprint arXiv:2312.12456*, 2023.

[11] S. Li, H. Lu, T. Wu, M. Yu, Q. Weng, X. Chen, Y. Shan, B. Yuan, and W. Wang, "Caraserve: Cpu-assisted and rank-aware lora serving for generative llm inference," *arXiv preprint arXiv:2401.11240*, 2024.

[12] K. Kamahori, Y. Gu, K. Zhu, and B. Kasikci, "Fiddler: Cpu-gpu orchestration for fast inference of mixture-of-experts models," *arXiv preprint arXiv:2402.07033*, 2024.

[13] KVCache-AI, "Ktransformers: A flexible framework for experiencing cutting-edge llm inference optimizations," 2024, https://github.com/kvcache-ai/ktransformers.

[14] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *Journal of Machine Learning Research*, vol. 23, no. 120, pp. 1–39, 2022.

[15] M. R. Costa-jussà, J. Cross, O. Çelebi, M. Elbayad, K. Heafield, K. Heffernan, E. Kalbassi, J. Lam, D. Licht, J. Maillard *et al.*, "No language left behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.

[16] A. Yang, B. Yang, B. Hui, B. Zheng, B. Yu, C. Zhou, C. Li, C. Li, D. Liu, F. Huang *et al.*, "Qwen2 technical report," *arXiv preprint arXiv:2407.10671*, 2024.

[17] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024.

[18] A. Liu, B. Feng, B. Wang, B. Wang, B. Liu, C. Zhao, C. Dengr, C. Ruan, D. Dai, D. Guo *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[19] D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu *et al.*, "Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models," *arXiv preprint arXiv:2401.06066*, 2024.

[20] R. Y. Aminabadi, S. Rajbhandari, A. A. Awan, C. Li, D. Li, E. Zheng, O. Ruwase, S. Smith, M. Zhang, J. Rasley *et al.*, "Deepspeed-inference: enabling efficient inference of transformer models at unprecedented scale," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 1–15.

[21] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, B. Chen, P. Liang, C. Ré, I. Stoica, and C. Zhang, "Flexgen: High-throughput generative inference of large language models with a single gpu," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 094–31 116.

[22] J. Li, Q. Su, Y. Yang, Y. Jiang, C. Wang, and H. Xu, "Adaptive gating in mixture-of-experts based language models," *arXiv preprint arXiv:2310.07188*, 2023.

[23] Y. Wei, J. Du, J. Jiang, X. Shi, X. Zhang, D. Huang, N. Xiao, and Y. Lu, "Aptmoe: Affinity-aware pipeline tuning for moe models on bandwidth-constrained gpu nodes," in *2024 SC24: International Conference for High Performance Computing, Networking, Storage and Analysis SC*. IEEE Computer Society, 2024, pp. 1436–1449.

[24] P. Li, X. Jin, Y. Cheng, and T. Chen, "Examining post-training quantization for mixture-of-experts: A benchmark," *arXiv preprint arXiv:2406.08155*, 2024.

[25] L. Xue, Y. Fu, Z. Lu, L. Mai, and M. Marina, "Moe-infinity: Activation-aware expert offloading for efficient moe serving," *arXiv preprint arXiv:2401.14361*, 2024.

[26] X. He, S. Zhang, Y. Wang, H. Yin, Z. Zeng, S. Shi, Z. Tang, X. Chu, I. Tsang, and O. Y. Soon, "Expertflow: Optimized expert activation and token allocation for efficient mixture-of-experts inference," 2024. [Online]. Available: https://arxiv.org/abs/2410.17954

[27] G. Gerganov, "ggerganov/llama.cpp: Port of facebook's llama model in c/c++," 2023. [Online]. Available: https://github.com/ggerganov/llama.cpp

[28] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing *et al.*, "Judging llm-as-a-judge with mt-bench and chatbot arena," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[29] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, H. Zhang, J. E. Gonzalez, and I. Stoica, "Judging llm-as-a-judge with mt-bench and chatbot arena," 2023. [Online]. Available: https://arxiv.org/abs/2306.05685

[30] MohamedRashad, "Chatgpt-prompts," 2023. [Online]. Available: https://huggingface.co/datasets/MohamedRashad/ChatGPT-prompts

PairGraph: An Efficient Search-space-aware Accelerator for High-performance Concurrent Pairwise Queries

Yutao Fu†, Zhongtian Long†, Yu Zhang†, Zirui He†, Jin Zhao†, Qiyuan Niu†, Zixiao Wang†, Hai Jin†
†*National Engineering Research Center for Big Data Technology and System, Services Computing Technology and System Lab, Cluster and Grid Computing Lab, Huazhong University of Science and Technology, Wuhan, China*
{fuyutao, longzt, zhyu, hezirui, zjin, qyniu, zwang62, hjin}@hust.edu.cn

***Abstract*—Pairwise queries have been widely used in many applications. Although several approaches have been recently proposed to accelerate a single query, they still suffer from irregular memory access and fragmented data sharing when processing *Concurrent Pairwise Queries* (CPQ) because of the poor temporal and spatial locality of traversal overlaps (i.e., graph structure data traversed by several queries). To address these challenges, this paper presents an accelerator named PairGraph to effectively support CPQ based on a novel *Search-space-aware Processing Model* (SPM). The key insight is the strong similarity of queries' search spaces, which are primarily concentrated on the graph topology between source and destination vertices. Consequently, our approach identifies the graph structure data traversed by most of the queries according to the graph topology between multiple pairs of vertices, and then fully reuses the data worth sharing to reduce off-chip communications. The experimental results indicate that PairGraph gains speedups of 5.59×∼14.25× and 3.76×∼7.58× compared with the state-of-the-art CPU-based system Gemini and the GPU-based system Gunrock, respectively. Compared with three cutting-edge accelerators, i.e., LCCG, ScalaGraph, and ReGraph, it gains speedups of 1.67×∼2.72×, 1.93×∼4.26×, and 2.66×∼4.28×, respectively.**

I. Introduction

Pairwise queries are widely used in domains such as navigation systems [24], recommendation systems [25], and bioinformatics [22], where identifying relationships between two vertices on the graph is essential. Recently, several systems have been proposed to accelerate a single pairwise query [3], [4], [11]. However, as application demands grow, merely accelerating a single query is no longer sufficient to handle the high volume of concurrent query requests in practical scenarios. This has introduced the concept of *Concurrent Pairwise Queries* (CPQ), which aims to process multiple pairwise queries in highly concurrent environments.

Although many solutions [3]–[11] can support the CPQ execution, they still suffer from irregular memory access and fragmented data sharing for two reasons. First, CPQ exhibits traversal overlaps, as some portions of the graph are often traversed by multiple queries. However, these overlaps demonstrate low temporal locality because the same graph structure data is always accessed by queries at different times, leading to unnecessary memory accesses. Second, because of the complex topological structure between two vertices on the graph, each pairwise query holds a search space that covers an unknown subset of the graph. As a result, the traversal overlaps demonstrate low spatial locality. It means that the graph structure data traversed by most of the queries, namely *shared regions*, cannot be identified at runtime, leading to fragmented data sharing. Specifically, when CPQ is processed on general-purpose processors, the data within the shared regions is fragmented and scattered across different locations in the main memory, while all the traversed graph structure data is loaded into the shared cache, including the data that can be only shared with a few queries. As a result, this data occupies the scarce on-chip memory but provides minimal sharing benefits.

Challenges come with opportunities. By analyzing the characteristics of the queries' search spaces, we have two key observations. First, although multiple queries access the graph structure data at different times, a significant portion of the data is identical, demonstrating *temporal similarity*. This indicates that many traversals of different queries can amortize the cost of accessing the same graph structure data if the data can be fully shared by all the queries. Second, most of queries' traversal paths are concentrated on the graph topology between the source and destination vertices, which shows *spatial similarity*. It indicates that the graph structure data traversed by most of the queries can be identified in the main memory according to the graph topology between multiple pairs of vertices before the queries are submitted.

Based on the above observations, in this paper, we propose PairGraph, a hardware accelerator based on a *Search-space-aware Processing Model* (SPM) to achieve efficient CPQ by fully exploiting the temporal and spatial locality of traversal overlaps. Unlike existing solutions, PairGraph employs a novel processing mechanism: shared region generation and overlap-driven processing. The former identifies the shared regions through tracking the graph topology bidirectionally between the source and destination vertices so as to leverage the spatial similarity of search spaces. In order to exploit the spatial locality of traversal overlaps, the overlap-driven processing mechanism leverages the data routing component to precisely fetch the data from the shared regions into the shared cache, while bypassing other non-shareable data, so as to coalesce the data worth sharing on-chip. Then the cached data is used to drive the CPQ execution to fully exploit temporal locality.

The experimental results show that PairGraph achieves speedups of 5.59×∼14.25× and 3.76×∼7.58× compared with the state-of-the-art CPU-based system Gemini [2] and the GPU-based system Gunrock [20], respectively. Compared with three cutting-edge accelerators, i.e., LCCG [5], ScalaGraph [12], and ReGraph [13], it gains speedups of 1.67×∼2.72×, 1.93×∼4.26×, and 2.66×∼4.28×, respectively.

II. Background and Motivation

A. Concurrent Pairwise Queries

Pairwise Query Algorithms. Pairwise queries are lightweight graph processing algorithms that explore the relationship between two vertices on the graph. Unlike typical graph processing algorithms that traverse the entire graph, a pairwise query leverages the *pruning mechanism* to reduce the search space to a subset of the graph. Specifically, once a path connecting the source and destination vertices is identified, it serves to prune other paths. For instance, when querying the shortest path between two vertices, the length of a discovered path can be utilized to constrain subsequent query, as

Yu Zhang (zhyu@hust.edu.cn) is the corresponding author of this paper.
This paper is supported by National Key Research and Development Program of China (No. 2024YFB4504200). This paper is also supported by NSFC (No. 62472183), Key Research and Development Program of Hubei Province (No. 2023BAB078). This work was supported by Ant Group through CCF-Ant Research Fund (No. CCF-AFSG RF20240204).

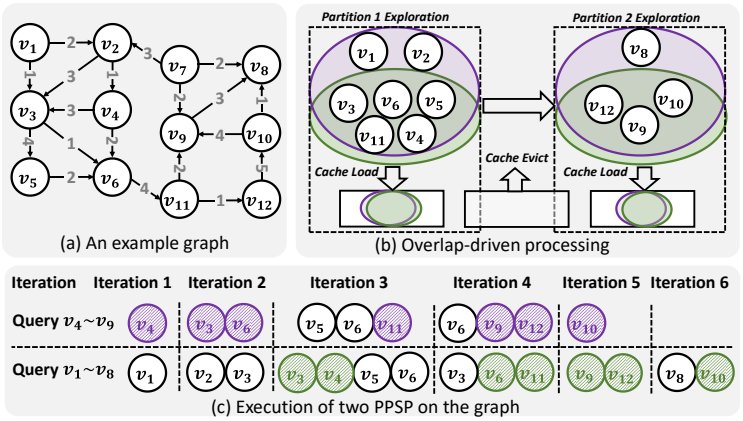


Fig. 1. An example of CPQ

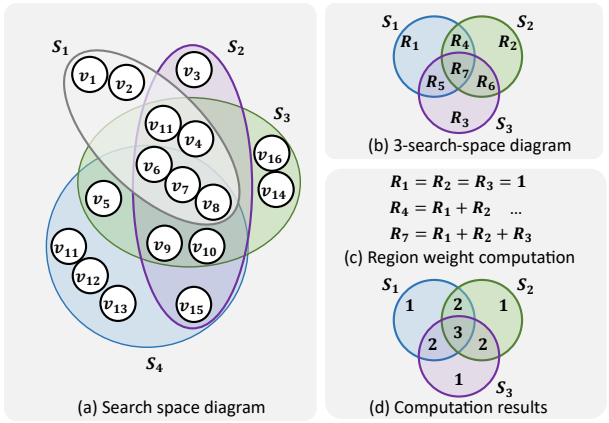


Fig. 2. An example of a search space diagram

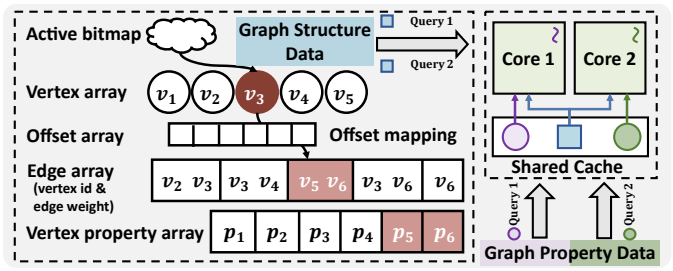


Fig. 3. Data access in CPQ under CSR storage structure

any longer paths can be disregarded. Consequently, pairwise queries can be defined as two-phase algorithms. The first phase aims to find a path from the source vertex to the destination vertex. Note that the path found in this phase is not necessarily optimal, and pruning does not occur during this phase. In the second phase, pruning begins once a path is found, enabling early termination of invalid traversals.

Concurrent Execution of Pairwise Queries. In practice, processing CPQ is more common than handling a single pairwise query because applications always require quick responses to handle a large number of queries in real-world data centers [5]. Concurrent execution has been shown to reduce inter-query blocking time [6], which enhances overall response speed. Consequently, this execution scheme is effective in improving the performance of pairwise query applications when facing thousands of query requests.

B. Limitations of Existing Solutions

There are two types of schemes that can support CPQ: one focuses on optimizing a single pairwise query, such as SGraph [4] and Pnp [3], while the other, including Glign [8] and LCCG [5], targets concurrent graph processing. However, neither type of scheme effectively supports CPQ, because the single-query scheme fails to consider data sharing between queries, while the concurrent-processing scheme overlooks the characteristics of pairwise queries. To investigate the causes of CPQ's high runtime overhead, we measure the performance of CPQ on Gemini, which is the state-of-the-art graph processing system, using different real-world graphs. Details of the baselines, datasets, and algorithms are presented in Section V. As shown in Fig. 4(a), we can observe that a significant portion of the execution time for processing CPQ is dedicated to memory access, while only a small portion is spent on computation. The results suggest that memory access is still the key bottleneck in processing CPQ, due to the following two problems.

Irregular Memory Access. Fig 1(c) depicts the accessing order of the vertices of two PPSP [4] queries on the illustrative graph in Fig. 1(a), where query a~b denotes a query with source vertex a and destination vertex b. We can observe that query v_4~v_9 and query v_1~v_8 traverse v_3, v_4, v_6, v_9, v_{10}, v_{11}, and v_{12} at different iterations. Consequently, as shown in Fig. 3, the vertex array, offset array, and edge array, which are collectively referred to as graph structure data in the widely used *Compress Sparse Row* (CSR) format [15], [16], are repeatedly accessed by CPQ, and the data is redundantly loaded into the shared cache, leading to significant off-chip communications. Note that the graph property data (i.e., the vertex property array in Fig.3) is private to each query and cannot be shared across queries. To quantify the impact, we conducted experiments to measure the redundant memory accesses caused by the irregular patterns during CPQ processing. As shown in Fig. 4(b), over 73.1% of memory accesses are found to be redundant.

Fragmented Data Sharing. Due to the diverse search spaces of pairwise queries, their traversal paths overlap in unknown portions of the graph. Consequently, the graph structure data accessed by most of the queries is fragmented within the main memory. This further leads to fragmented data sharing in CPQ and hinders the utilization of the shared cache. For example, vertices like v_1, v_2, and v_8, along with their edges in Fig. 1(a) should not be shared, as the search spaces of query v_4~v_9 and query v_1~v_8 do not overlap on these vertices. However, existing approaches still load these vertices and their edges into the shared cache, which may evict the data worth sharing. As shown in Fig. 4(c), we find that less than 13.9%, 1.1%, and 0.1% of the graph structure data can be shared by more than two, four, and eight queries during an iteration, respectively. This indicates that most of the data loaded into the shared cache is inefficiently shared. As a result, the cached data consumes the scarce on-chip memory but provides minimal benefits for data reuse.

C. Observations

We observe the search spaces of pairwise queries exhibit significant temporal and spatial similarity. It provides the opportunity to improve the performance of CPQ processing.

Temporal Similarity of Search Spaces. We observe that a large amount of the same graph structure data is accessed by various queries at different times. This suggests that if this data can be fully shared by all the queries in CPQ before being evicted from the shared cache, the off-chip communications can be significantly reduced. As shown in Fig 1(c), if the data traversed by query v_4~v_9 (i.e., v_4, v_3, and their edges) is pinned in the shared cache during iterations 1 and 2, then this data can be shared with query v_1~v_8 during iteration 3. It motivates the use of data that needs to be shared to drive CPQ execution, thereby leveraging temporal similarity to reduce off-chip communications.

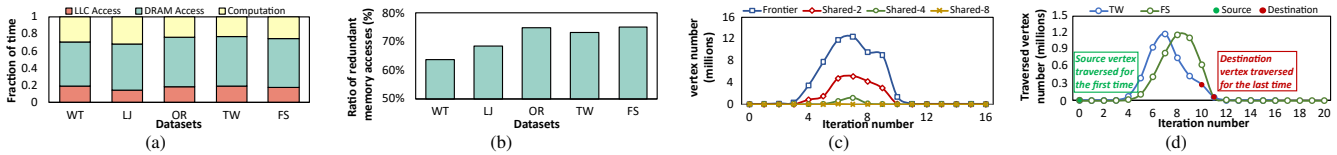


Fig. 4. Studies of the characteristics of CPQ over Gemini: (a) The execution time breakdown; (b) Ratio of redundant memory accesses; (c) Data that can be shared in each iteration on TW; (d) Spatial similarity of the search spaces for pairwise queries on TW and FS

Spatial Similarity of Search Spaces. We observe that the search spaces of pairwise queries exhibit notable spatial similarity. Fig. 4(d) illustrates that most traversals are made between the source and destination vertices, since any further exploration inferior to the existing optimal path is pruned after reaching the destination vertex. This indicates that, based on the position of the source and destination vertices in a pairwise query, a majority of the query's traversal paths can be identified according to the graph topology between these two vertices. Consequently, the graph structure data traversed by most of the queries (i.e., shared regions) can be identified before the queries are submitted.

Based on the spatial similarity, shared regions on the graph can be modeled using the *search space diagram* as shown in Fig. 2(a). The search spaces of pairwise queries in CPQ, denoted as S_i, overlap on the graph and generate several regions. The importance of data across these regions is determined by the *region weight*, denoted as R_i in Fig. 2(b). Initially, the region weight for each query is set to 1, and the weights for new regions are computed using the operations as shown in Fig. 2(c). The computed result is [1, 1, 1, 2, 2, 2, 3] for [R_1, R_2, R_3, R_4, R_5, R_6, R_7] in Fig. 2(d). Regions with higher region weight are more frequently accessed by queries, indicating that the data in these regions should be prioritized for sharing. The average region weight is used as the threshold for selecting the shared regions, which is demonstrated to be an effective way (see Section V-B).

III. Search-Space-Aware Processing Model

Based on the observations above, we propose *Search-space-aware Processing Model* (SPM) to achieve efficient CPQ. It achieves temporal and spatial locality of traversal overlaps by generating shared regions on the graph and fully sharing the graph structure data within these regions.

A. Redundancy-free Shared Region Generation

The spatial similarity of search spaces motivates us to generate the shared regions based on the search space diagram. A useful method is bidirectionally tracking the graph topology between multiple pairs of source and destination vertices in a breadth-first manner. For each query, a sub-tracking propagates the region weight between the source and destination vertices. The new region weight is obtained by summing the propagated weights across vertices from several sub-tracking.

However, this straightforward scheme is inefficient due to redundant propagations. Assume that the region weight is propagated between v_4 and v_9, and between v_1 and v_8 to generate the shared regions for query $v_4 \sim v_9$ and query $v_1 \sim v_8$ in Fig. 1(a). The propagations $v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_3$ and $v_4 \rightarrow v_3$ are ongoing asynchronously, with the region weight being repeatedly propagated across the successors of v_4. This leads to repeated access to the same edges and unnecessary computations on the same vertices. Luckily, there is an opportunity to gather the propagations at v_4 and deliver them to the following vertices all together in a single step. To address this, the process of generating shared regions is divided into two stages: the tagging stage and the propagating stage. In the tagging stage, the graph structure is traversed bidirectionally from multiple pairs of source and destination vertices, with each edge being only traversed once. The number of times vertices are traversed, referred to as the tag, is recorded. Then in the propagating stage, the region weight is propagated and temporarily gathered at vertices with non-zero tags, as it indicates that some propagations have not yet arrived. Each time a propagation arrives at a vertex, the tag of the vertex decreases by one. The gathered propagation is forwarded only when the tag decreases to zero or when no other propagations are executing. This approach effectively eliminates redundant propagations, which enhances the efficiency of shared region generation.

Moreover, when generating shared regions, a path result between two vertices can be already obtained by propagating the path data along with the region weight. Although this result is not optimal, it can be used for pruning. This means that pruning can begin immediately when the queries are submitted, thereby reducing unnecessary computations.

B. Overlap-driven Processing

Based on the temporal similarity of search spaces, we adopt an overlap-driven processing approach to fully reuse the graph structure data within the shared regions. Specifically, the query process is coarse-grained into multiple steps of partition exploration. At each step, data from the shared regions is coalesced in the shared cache to drive the execution of multiple queries. The number of edges loaded from the shared regions is strictly controlled by the cache size to ensure that data from each exploration step can be contained within the cache. The cached data is pinned until all the queries have processed it, after which the execution proceeds to the next step.

The following one-step partition exploration illustrates the execution process described above. Assume that the cache can contain six edges. As shown in Fig. 1(b), for the query $v_4 \sim v_9$ and query $v_1 \sim v_8$, they first explore partition 1 in parallel, where the graph structure data in the shared regions (i.e., $v_4 \rightarrow v_3$, $v_4 \rightarrow v_6$, $v_3 \rightarrow v_5$, $v_3 \rightarrow v_6$, $v_5 \rightarrow v_6$, and $v_6 \rightarrow v_{11}$) are loaded into the cache and shared. Note that $v_1 \rightarrow v_2$, $v_1 \rightarrow v_3$, $v_2 \rightarrow v_3$, and $v_2 \rightarrow v_4$ are bypassed since they are identified as data outside the shared regions. After all the cached data is processed by the two queries, it is evicted and the execution proceeds to the next step of partition exploration.

C. Limitations of Implementing SPM on Typical Architectures

SPM faces significant challenges when implemented on general-purpose processors (i.e., CPU and GPU) since typical architectures present the following limitations. First, SPM relies on overlap-aware memory access mode, where data is cached or bypassed based on the region weight. However, such fine-grained control is not supported by typical architectures. Second, SPM requires pinning the data worth sharing on-chip, which is infeasible with the cache management strategies employed by typical architectures. Third, while SPM improves data sharing efficiency among queries at the algorithmic level, its performance is constrained by the prevalence of random memory accesses in CPQ. This issue, coupled with the low memory bandwidth utilization of typical architectures limits the overall performance gains. Consequently, it highlights the need for dedicated hardware architecture to fully realize the potential of SPM.

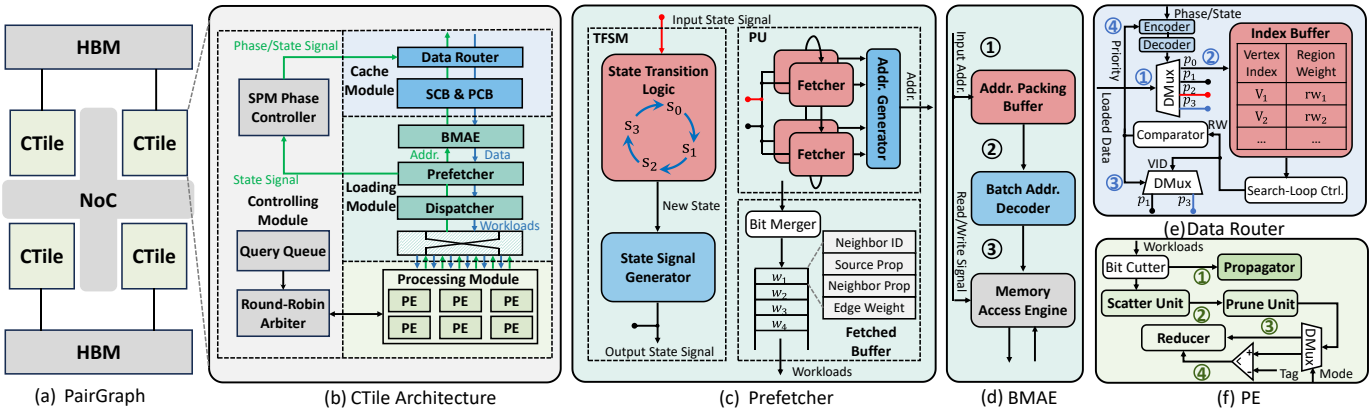


Fig. 5. PairGraph system architecture

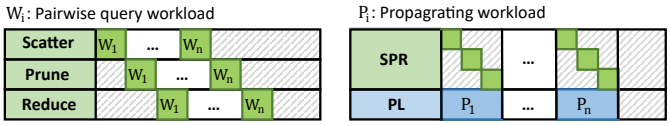


Fig. 6. Processing SPM-customized workloads: (a) Scatter-prune-reduce pipeline; (b) Propagating lane

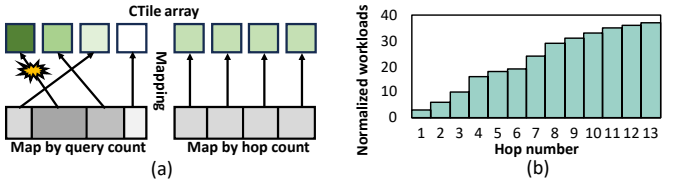


Fig. 7. Correlation-aware workload mapping: (a) Impact of different mapping strategies; (b) Correlation between workloads and hops

	1 bit Exec Phase	2 bit Pref State	1 bit Priority	
code 1	RSG (0)	\	\	SCB
code 2	OP (1)	S_0 (00)	\	INB
code 3	OP (1)	S_1 (01)	1	SCB
code 4	OP (1)	S_2 (10)	1	SCB

code 5	OP (1)	S_3 (11)	\	PCB
code 6	OP (1)	S_1 (01)	0	Bypass
code 7	OP (1)	S_2 (10)	0	Bypass

Fig. 8. Routing code

IV. PairGraph Architecture

A. Architecture Overview

As shown in Fig. 5(a), PairGraph comprises the *Concurrent Tile Array*, *Network-On-Chip* (NoC), and *High-Bandwidth-Memory* (HBM). The system architecture of a *Concurrent Tile* (CTile) is shown in Fig. 5(b). It consists of the *Controlling Module*, *Processing Module*, *Loading Module*, and *Cache Module*.

PairGraph's workflow aligns with the two phases of SPM, which first executes the *Redundancy-free Shared Region Generation* (RSG), and then maps the queries to CTiles for *Overlap-driven Processing* (OP). As shown in Fig. 5(b), the *Controlling Module* manages the execution phases by transmitting phase/state signals through the *SPM Phase Controller*, and schedules the queries in the *Query Queue* using the *Round-Robin Arbiter* to ensure execution fairness. The *Loading Module* achieves efficient memory access via the *Prefetcher* and *Batch Memory Access Engine* (BMAE). The *Prefetcher* leverages a task-level data prefetch pipeline to efficiently prefetch the graph data, while the BMAE applies a memory request packing approach to effectively utilize the memory bandwidth. The fetched workloads are evenly dispatched to the PEs in the *Processing Module* by the *Dispatcher*. And the PE is specifically designed to process the SPM-customized workloads. *The Cache Module* contains the *Shared Cache Block* (SCB) for graph-structure data sharing, and the *Private Cache Block* (PCB) for graph property data caching. To fully reuse the data worth sharing, the *Data Router* employs an overlap-aware data routing mechanism to route the data from the shared regions to the SCB and bypass other non-shareable data.

B. Graph Data Prefetching and Memory Request Packing

Graph Data Prefetching. We developed a task-level prefetch pipeline consisting of four sub-tasks to efficiently fetch graph data. As shown in Fig. 5(c), the *Traversal Finite State Machine* (TFSM) manages the four states (i.e., S_0: fetch_source, S_1: fetch_offsets, S_2: fetch_edges, and S_3: fetch_properties) of the *Prefetch Unit* (PU) by generating state signals. Under TFSM control, the PU first gets the active vertex index in S_0, followed by fetching edge offsets in S_1. In S_2, it iteratively prefetches 16 B of the associated edge data based on the offsets. Finally, in S_3, it fetches the vertex property data of the current active vertex and its neighboring vertices. The fetched data is merged into a workload and transmitted to the *Dispatcher* via the *Fetched Buffer*.

Memory Request Packing. When processing CPQ, off-chip memory traffic generated during an iteration in SPM can be substantial, due to the *power-law property* [5] and concurrent execution. With 36 concurrent pairwise queries, we find that the traffic can reach 280 KB∼3.9 MB per iteration. However, this traffic is always fragmented into very small memory requests, each request only asking for a few bytes of data. This leads to extremely low memory bandwidth utilization in HBM (about 2%∼4%), thereby constraining the overall performance. To better utilize the memory bandwidth, the BMAE packs the memory requests during an iteration. Specifically, it merges multiple memory requests to generate a single request asking for 96 KB of data. As shown in Fig. 5(d), the input addresses are temporarily stored in the *Address Packing Buffer* and counted (①). Once the count reaches the limit, addresses are decoded in a batch (②) and sent to the *Memory Access Engine* (③). The packed addresses are then put on the address bus, and all the corresponding data is accessed in one go. Note that when there are no addresses streaming into the *Address Packing Buffer*, the remaining addresses are directly issued.

C. Overlap-aware Data Routing

To exploit the spatial locality of the traversal overlaps, we designed the *Data Router* with an overlap-aware data routing mechanism. In the RSG phase, data is directly cached in SCB, because all the data is useful for generating the shared regions. In the OP phase, when the queries are submitted, only the graph structure data in the shared regions is cached.

The workflow of the *Data Router* is shown in Fig. 5(e). Once

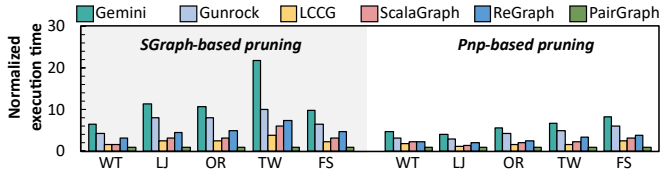


Fig. 9. Execution time normalized to PairGraph

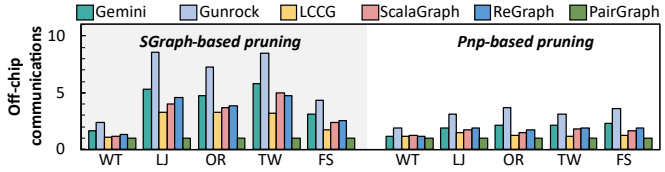


Fig. 10. Off-chip communications normalized to PairGraph

the data is loaded (①), the *Data Router* routes the data using a 4-bit routing code (see Fig. 8) encoded by the phase/state and priority signals. The phase/state signal determines the execution phase (RSG/OP) and data prefetch states, while the priority signal decides whether the data is in the shared regions. Based on the decoded information, the loaded data is sent to the *Index Buffer* (INB) (p_0), SCB (p_1), PCB (p_2), or directly bypassed (p_3) (②). For the case p_0, if the region weight of the loaded vertex exceeds the threshold (i.e., the average region weight), the fetched vertex id is routed to the SCB via the DMux based on the priority signal (③), and the corresponding bit of the routing code in the *Encoder*, which represents priority, is set to 1 (④). Otherwise, the bit is set to 0, and the vertex id is bypassed.

D. SPM-customized Workloads Processing

The workloads in SPM consist of two parts: the original pairwise query workloads, and the propagating workloads for generating shared regions. The two types of workloads occur simultaneously at the propagating stage of the RSG phase. To process them, the edge should be processed by a function to determine whether the neighboring vertices should be activated. Additionally, the region weight needs to be propagated to the neighboring vertices for generating shared regions. To efficiently handle these workloads, we design *Scatter-Prune-Reduce Pipeline* (SPR) and *Propagating Lane* (PL) as shown in Fig. 6. The key insight is that there is no data dependency when processing the two types of workloads, which indicates that they can be processed in parallel.

As shown in Fig. 5(f), when a workload is dispatched to a PE, it is cut and sent to the *Scatter Unit* and *Propagator* simultaneously, enabling parallel processing by the SPR and PL. In the PL, the region weight is directly propagated to the neighboring vertex by the propagator (①). In the SPR, temporary data is firstly computed by the *Scatter Unit* using the property data of the source vertex and the edge weight. Then it is sent to the *Prune Unit*, where it is compared with the property data of neighboring vertices and existing computation results to determine whether to be pruned (②). If not pruned, the data is written to the vertex property array for the neighboring vertex by the *Reducer* (③), and the neighboring vertex is activated in the next iteration. Note that in the propagating stage of the RSG phase, the temporary data is compared with the tag instead of being sent to the *Prune Unit* to decide whether it should be reduced (④).

E. Correlation-aware Query Mapping

In the OP phase, queries are allocated to different CTiles for concurrent execution. However, directly mapping queries to CTiles according to the query count leads to load imbalance. As illustrated in Fig. 7(a), queries with various workloads are distributed across different CTiles based on the query count, often leading to load

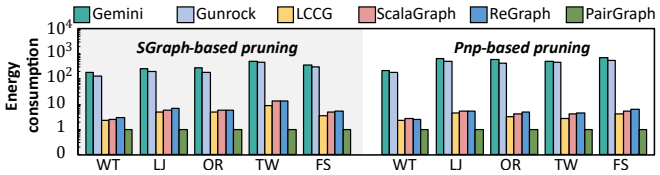


Fig. 11. Energy consumption normalized to PairGraph

imbalance due to the significant variation in workloads across queries. It means that, even when an equal number of queries are allocated to each CTile, some of them still become idle prematurely, which leads to low CTile utilization.

Fortunately, we discover the workloads of a pairwise query are strongly correlated with the minimum hop count between the source and destination vertices, as illustrated in Fig. 7(b). The reason is that a smaller hop count facilitates faster path discovery for pruning in pairwise query algorithms, thereby reducing overall number of convergent iterations. Based on this correlation, queries with the same total hop count are allocated to each CTile so as to achieve workload balance among CTiles. It should be noted that the minimum hop count can be easily recorded by the *propagator* in the RSG phase.

V. Evaluation

A. Experimental Setup

PairGraph Simulation and Configurations. A cycle-accurate simulator is developed to model the behaviors of PairGraph. We wrote Verilog to implement the modules of PairGraph, and synthesized the RTL using Synopsys Design Compiler with TSMC 28nm standard library at 1 GHz to obtain the consumption of area and power for processing units. For the on-chip cache and buffers, we used CACTI [18] to model the area and power consumption. The behaviors of the off-chip memory access are modeled using Ramulator [17]. PairGraph consists of 4 CTiles, each containing 6 PEs. For on-chip memory on a CTile, we used 7 MB SCB and 1 MB PCB for data caching, 2 KB *Query Queue* for query scheduling, 4 KB *Fetched Buffer* for workload fetching, and 8 KB *Index Buffer* for region weight indexing.

Baselines. We compared PairGraph with a CPU-based system Gemini [2]; a GPU-based system Gunrock [20]; and three advanced graph processing accelerators: LCCG [5], ScalaGraph [12], and ReGraph [13]. We integrated two cutting-edge pruning functions: Pnp-based pruning function [3] and SGraph-based pruning function [4] into these baselines to implement pairwise query applications and adapted the baselines for concurrent query support. The CPU baseline is implemented on a server with two 32-core, 2.3 GHz Intel Xeon Gold 6314U processors, while the GPU baseline runs on an NVIDIA Tesla V100 GPU. Since the original configurations of the compared accelerators are different, we make the following efforts to ensure fair comparisons. First, we use the same on-chip cache capabilities (32 MB) as PairGraph for data caching. Second, we use the same number of compute units (1 GHz 24×SIMT) to achieve equivalent computational performance. Last, we use main memory with the same performance (512 GB/s HBM 1.0) to ensure I/O fairness.

Datasets and Algorithms. We use six pairwise query algorithms, namely *Shortest Path* (PPSP) [4], *Widest Path* (PPWP) [4], *Narrowest Path* (PPNP) [4], *Breadth First Search* (BFS) [4], *Connectivity* (CN) [4], and *Reachability* (RC) [4], and five real-world graphs listed in Table I to evaluate the performance. To generate CPQ, the algorithms are submitted in a batch of 36 queries with each algorithm submitted six times, running between different pairs of source and destination vertices.

TABLE I
GRAPH DATASET USED IN THE EVALUATION

Datasets	Vertices	Edges	Directed
Wikipedia Talk (WT) [1]	2,394,385	5,021,410	Yes
LiverJournal (LJ) [1]	4,847,571	68,993,773	Yes
Orkut (OR) [1]	3,072,441	117,185,083	No
Twitter (TW) [1]	41,652,230	1,468,364,884	Yes
Friendster (FS) [1]	65,608,366	1,806,067,135	No

TABLE II
VARIOUS BATCHES OF QUERIES

Batch	The Batch of Query Algorithms
B_1	{36×PPSP}
B_2	{18×PPSP, 6×PPWP, 6×PPNP, 6×BFS}
B_3	{12×PPSP, 6×PPWP, 6×PPNP, 6×BFS, 6×RC}
B_4	{6×PPSP, 6×PPWP, 6×PPNP, 6×BFS, 6×CN, 6×RC}
B_5	{24×PPSP, 12×BFS}
B_6	{36×BFS}

B. Performance of PairGraph

Overall Performance. Fig. 9 shows that PairGraph outperforms the other solutions on all the datasets. Compared with Gemini, Gunrock, LCCG, ScalaGraph, and ReGraph, PairGraph improves the performance by 5.59×∼14.25×, 3.76×∼7.58×, 1.67×∼2.72×, 1.93×∼4.26×, and 2.66×∼4.28×, respectively. These improvements are primarily due to the reduction in off-chip communications.

Off-chip Communications. As shown in Fig. 10, PairGraph reduces off-chip communications to 32.88%, 21.46%, 52.83%, 41.25%, and 38.76% of Gemini, Gunrock, LCCG, ScalaGraph, and ReGraph on average, respectively. The reduction stems from two key factors: First, PairGraph generates shared regions to effectively identify the graph structure data worth sharing, which can help to reduce fragmented data sharing during execution. Second, the data in the shared regions is fully shared by pinning in the SCB at each step of partition exploration, significantly enhancing data reuse.

Energy Savings. We use CPU Energy Meter [13] and nvidia-smi [13] to measure the energy consumption of Gemini and Gunrock, respectively. Fig. 11 shows that PairGraph reduces the energy consumption of Gemini, Gunrock, LCCG, ScalaGraph, and ReGraph by 443.62×, 343.13×, 4.15×, 5.38×, and 5.71× on average, respectively.

Power and Area Cost. The total power and area cost of PairGraph are 3.21 W and 11.23 mm^2, respectively. The *Cache Module* uses most on-chip memory resources, occupying 71.2% of the area and consuming 26.2% of the power. The *Processing Module* occupies 13.4% of the area and consumes 52.1% of the power for workloads processing. The *Loading Module* takes up 11.1% of the area and uses 16.9% of the power for graph data prefetching and workloads dispatching. Lastly, the *Controlling Module* occupies 4.1% of the area and costs 4.8% of the power.

Sensitivity Analysis. Fig. 12(a) shows the performance of PairGraph under different thresholds for selecting shared regions. We evaluated the impact of setting the threshold to the minimum, maximum and average region weight. The results indicate that the scheme using the average region weight outperforms the others. Setting the threshold too high causes the *Data Router* to classify the majority of data as not requiring to be shared, while setting it too low hinders PairGraph's ability to reduce fragmented sharing. Fig. 12(b) illustrates the sensitivity to the number of concurrent queries. The result shows that PairGraph achieves better performance as the number of queries increases. Fig. 12(c) shows the performance of PairGraph when processing various batches of pairwise query algorithms. We can observe that PairGraph consistently outperforms other solutions for different cases listed in Table II. Fig. 12(d) shows that PairGraph outperforms other solutions due to higher cache utilization. And it obtains higher performance when the cache size is larger since more data can be effectively cached and shared by the queries.

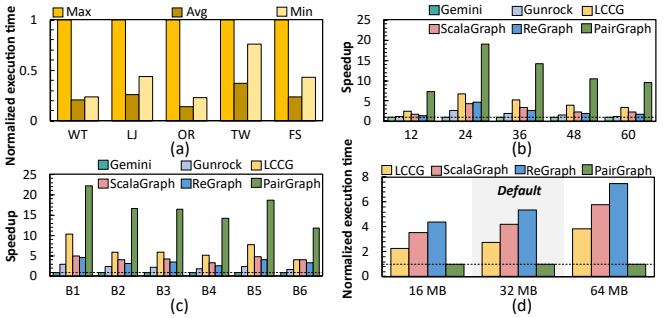


Fig. 12. Sensitivity studies: (a) Sensitivity to the threshold; (b) Sensitivity to the concurrent number on TW; (c) Speedups when using different batches of queries on TW; (d) Sensitivity to the cache size on TW

VI. RELATED WORK

Pairwise Query. Several systems have been proposed to effectively handle a single pairwise query. Hub^2 [11] introduces a hub-centric strategy to limit the search scope of high-degree vertices, which significantly speeds up the PPSP process. Pnp [3] proposes a direction-aware pruning approach for processing pairwise queries with no preprocessing overhead. SGraph [4] applies both upper and lower bounds to impose stringent constraints so as to enhance pruning. However, these works still suffer from significant irregular memory accesses when processing CPQ.

Concurrent Graph Processing. Many concurrent graph processing schemes are proposed to address large-scale graph data. For distributed systems, Seraph [9] supports concurrent graph processing by decoupling the graph structure data. MultiLyra [21] and BEAD [23] improve batched query evaluation by distributing communication costs through graph sharing. For out-of-core systems on a single machine, prior works like CGraph [6] and GraphM [19] follow the idea of block-driven execution to exploit data sharing. Glign [8] introduces an inter-iteration alignment approach to enhance sharing efficiency. For in-memory graph processing, Congra [7] improves throughput by utilizing dynamic query scheduling and being mindful of atomic operations, while Krill [10] adopts an SAP model to simplify attribute data management and reduce memory accesses. Targeting hardware accelerators, LCCG [5] is proposed to eliminate redundant memory accesses along the graph topology so as to reduce off-chip communications. Although these approaches support the concurrent execution of typical point-to-all algorithms, they overlook the negative impact of the pairwise query's unique search space on data sharing.

VII. CONCLUSION

This paper introduces PairGraph, a search-space-aware hardware accelerator for efficient CPQ processing. According to the characteristics of pairwise queries' search spaces, it identifies the graph structure data traversed by most of the queries and coalesces the data in the shared cache to drive the CPQ processing, which fully exploits the temporal and spatial locality of traversal overlaps. As a result, PairGraph gains speedups of 5.59×∼14.25× and 3.76×∼7.58× compared with the state-of-the-art CPU-based system and GPU-based system, respectively. Compared with three cutting-edge accelerators, it gains speedups of 1.67×∼2.72×, 1.93×∼4.26×, and 2.66×∼4.28×, respectively.

REFERENCES

[1] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun.2014.

[2] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *Proceedings of OSDI*, 2016, pp. 301–316.

[3] C. Xu, K. Vora, and R. Gupta, "Pnp: Pruning and prediction for point-to-point iterative graph analytics," in *Proceedings of ASPLOS*, 2019, pp. 587–600.

[4] H. Chen, M. Zhang, K. Yang, K. Chen, A. Zomaya, Y. Wu, and X. Qian, "Achieving sub-second pairwise query over evolving Graphs," in *Proceedings of ASPLOS*, 2023, pp. 1–15.

[5] J. Zhao, Y. Zhang, X. Liao, L. He, B. He, H. Jin, and H. Liu, "LCCG: A locality-centric hardware accelerator for high throughput of concurrent graph processing," in *Proceedings of SC*, 2021, pp. 1–14.

[6] Y. Zhang, X. Liao, H. Jin, L. Gu, L. He, B. He, and H. Liu, "CGraph: A correlations-aware approach for efficient concurrent iterative graph processing," in *Proceedings of ATC*, 2018, pp. 441–452.

[7] P. Pan and C. Li, "Congra: Towards efficient processing of concurrent graph queries on shared memory machines," in *Proceedings of ICCD*, 2017, pp. 217–224.

[8] X. Yin, Z. Zhao, and R. Gupta, "Glign: Taming misaligned graph traversals in concurrent graph processing," in *Proceedings of ASPLOS*, 2022, pp. 78–92.

[9] J. Xue, Z. Yang, Z. Qu, S. Hou, and Y. Dai, "Seraph: An efficient, low-cost system for concurrent graph processing," in *Proceedings of HPDC*, 2014, pp. 227–238.

[10] H. Chen, M. Shen, N. Xiao, and Y. Lu, "Krill: a compiler and runtime system for concurrent graph processing," in *Proceedings of SC*, 2021, pp. 1–16.

[11] R. Jin, N. Ruan, B. You, and H. Wang, "Hub-accelerator: Fast and exact shortest path computation in large social networks," in *arXiv preprint arXiv:1305.0507*, 2013.

[12] P. Yao, L. Zheng, Y. Huang, Q. Wang, C. Gui, Z. Zeng, X. Liao, H. Jin, and J. Xue, "ScalaGraph: A scalable accelerator for massively parallel graph processing," in *Proceedings of HPCA*, 2022, pp. 199–212.

[13] X. Chen, Y. Chen, F. Cheng, H. Tan, B. He, and W. Wong, "ReGraph: Scaling graph processing on HBM-enabled FPGAs with heterogeneous pipelines," in *Procceedings of MICRO*, 2022, pp. 1342-1358.

[14] A. Mukkara, N. Beckmann, M. Abeydeera, X. Ma, and D. Sanchez, "Exploiting locality in graph analytics through hardware-accelerated traversal scheduling," in *Procceedings of MICRO*, 2018, pp. 1–4.

[15] N. Bell and M. Garland, "Implementing sparse matrix-vector multiplication on throughput-oriented processors," in *Procceedings of SC*, 2009, pp. 1–11.

[16] J. Shun and G. E. Belloch, "Ligra: a lightweight graph processing framework for shared memory," in *Procceedings of PPoPP*, 2013, pp. 135–146.

[17] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible DRAM simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[18] R. Balasubramonian, A. B. Kahng, N. Muralimanobar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization*, vol. 14, no. 2, pp. 1–25, 2017.

[19] J. Zhao, Y. Zhang, X. Liao, L. He, B. He, H. Jin, H. Liu, and Y. Chen, "GraphM: an efficient storage system for high throughput of concurrent graph processing," in *Proceedings of SC*, 2019, pp. 1–14.

[20] Y. Wang, A. Davidson, Y. Pan, Y. Wu, A. Riffel, and J. D. Owens, "Gunrock: A high-performance graph processing library on the GPU," in *Proceedings of PPoPP*, 2016, pp. 1–12.

[21] A. Mazloumi, X. Jiang, and R. Gupta, "Multilyra: Scalable distributed evaluation of batches of iterative graph queries," in *Proceedings of BigData*, 2019, pp. 349–358.

[22] Y. Huang, L. Zheng, H. Liu, Z. Zhou, D. Chen, P. Yao, Q. Wang, X. Liao, and H. Jin, "Meg2: In-memory Acceleration for genome graph analysis," in *Proceedings of DAC*, 2023, pp. 1–6.

[23] A. Mazloumi, C. Xu, Z. Zhao, and R. Gupta, "BEAD: Batched evaluation of iterative graph queries with evolving analytics demands," in *Proceedings of BigData*, 2020, pp. 461–468.

[24] H. Wang, G. Li, H. Hu, S. Chen, B. Shen, H. Wu, W. Li, and K. Tan, "R3: A real-time route recommendation system," in *Proceedings of the VLDB Endowment*, 2014, pp. 1549–1552.

[25] K. Joseph and H. Jiang, "Content based news recommendation via shortest entity distance over knowledge graphs," in *Proceedings of WWW*, 2019, pp. 690–699.

Blaze: An Efficient Bit-Sparse Attention Architecture With Workload Orchestration Optimization

Runzhou Zhang*, Faxian Sun, Yiming Wang, Kunchen Zou, Zhinan Qin, Jianli Chen, Jun Yu and Kun Wang†
State Key Lab of Integrated Chips & Systems, and School of Microelectronics, Fudan University, Shanghai, China
Email: *23212020028@m.fudan.edu.cn, †kun.wang@ieee.org

***Abstract*—The attention mechanism is a core neural network primitive widely utilized in state-of-the-art models of Natural Language Processing (NLP) applications. However, the high computational complexity and substantial power consumption hinder its deployment and efficient inference. To address these challenges, various methods leveraging sparsity and quantization have been proposed. Compared to these methods, the exploitation of abundant bit-level sparsity in attention-based models presents great potential for the performance enhancement of attention inference. Existing bit-sparsity optimization methods primarily focus on Convolutional Neural Networks (CNNs), which are not ideally suitable for the attention mechanism, and they have not effectively solved the workload imbalance and hardware under-utilization issues caused by the irregular distribution of non-zero bits in tensor data. In this work, we introduce Blaze, an efficient attention architecture that leverages both value and bit-level sparsity in tensor data along with workload orchestration optimization. To mitigate the workload imbalance issues often encountered by sparse bit-serial architecture, we propose an Approximate-Computing-Based (ACB) workload orchestration mechanism. Additionally, to fully exploit the redundancy in the attention mechanism, we propose a Leading-Booth mechanism to further enhance the performance of attention computation. We also design a reconfigurable computing engine to support both mechanisms. Experimental results indicate that, compared to state-of-the-art (SOTA) attention accelerators, our Blaze can achieve $2.37\times \sim 6.18\times$ improvement in performance and $9.69\times \sim 43.96\times$ enhancement in energy efficiency. Our accelerator can reach up to $1.58\times$ speedup in attention computing performance compared with the SOTA bit-sparse accelerator.**

I. INTRODUCTION

Recently, the attention mechanism [1] has revolutionized the field of Artificial Intelligence (AI), leading to the development of advanced models such as Transformer [1], BERT [2], and GPT-2 [3] that have demonstrated unparalleled performance across a wide array of tasks. However, the deployment of attention-based models is often hindered by their high power consumption and substantial computational complexity. As the demand for attention-based model deployment expands across various applications, the pursuit of energy-efficient attention inference architectures has become increasingly critical. This necessity fuels the search for innovative approaches that can enhance both performance and energy efficiency in attention accelerators.

To improve the inference efficiency of attention-based models, various accelerators utilizing sparsity and quantization have been proposed. SpAtten [4] implements token pruning to leverage the sparsity present in sentences. Sanger [5] utilizes dynamic structured sparsity for attention acceleration. DTQAtten [6] and DTATrans [7] dynamically quantize tokens using mixed precision, while DEQ [8] further proposes the dynamic quantization of elements with mixed precision. Although these sparsity and quantization methods have significantly enhanced the inference efficiency of the attention mechanism, they primarily focus on value-level optimizations, overlooking the potential for bit-level enhancements. For non-zero values, not all bits are effective, which motivates us to exploit bit-level sparsity to improve the performance of attention inference.

Sparse bit-serial architectures are widely employed to exploit bit-level sparsity. Rather than processing operands directly, these architectures treat integer operands as sequences of bits to enable the skipping of zero bits [9]–[11]. Specifically, Bit-Pragmatic [9] serializes activations while keeping weights fixed within the bit-serial architecture. Laconic [10] utilizes bit sparsity in both weights and activations. AdaS [11] selectively serializes the side with the less effective Booth term. However, these sparse bit-serial accelerators face workload imbalance and hardware under-utilization due to the irregular distribution of non-zero bits in activations and weights. This inconsistency results in some processing elements (PEs) being active while others remain idle. Although AdaS [11] partially mitigates this issue through group-wise bi-directional inner-join for workload orchestration within the same PE, the imbalance across the entire PE array remains unresolved.

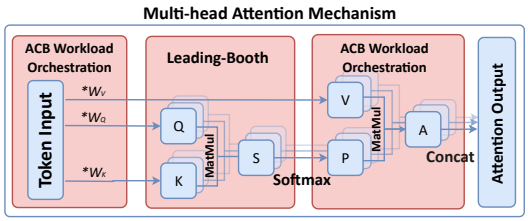


Fig. 1. The application of ACB workload orchestration and Leading-Booth computing schemes in the attention mechanism.

To alleviate the workload imbalance problem, Bit-Balance [12] proposes a channel-wise bit-sparsity quantization method, while BitCluster [13] introduces a hardware-friendly quantization approach to ensure that each scalar element in the weight tensor has the same number of effective bits. Both methods tackle the imbalance by serializing weights and equalizing effective bits through software processing. These methods are applicable to Convolutional Neural Networks (CNNs) and the linear layers in Transformer models, as the weights of linear layers can be predetermined before inference. However, these approaches are less suitable for the attention mechanism, where the Queries (Qs), Keys (Ks), and Values (Vs) in the attention are generated at run-time. Therefore, it is necessary to dynamically schedule the workload at the hardware level. BSViT [14] aims to accelerate ViT [15] through a weight bit-group quantization scheme and a dense-sparse hybrid data flow for workload orchestration among PEs. However, in terms of exploiting the inherent characteristics of Transformers, while BSViT employs a dynamic patch quantization strategy for input data processing, it overlooks the architectural and acceleration potential within the attention mechanism.

To address the aforementioned challenges, we propose Blaze, an efficient bit-sparse attention architecture, featuring a dedicated optimization scheme for workload orchestration optimization. The contributions of this work are as follows:

- We propose a dedicated sparse bit-serial hardware architecture, Blaze, to fully exploit both the value and bit-level sparsity for attention acceleration. We design a reconfigurable computing array to support our optimization schemes, including the Leading-Booth and the Approximate-Computing-Based workload orchestration mechanisms.

†Corresponding author

- We present an Approximate-Computing-Based (ACB) workload orchestration mechanism to tackle the workload imbalance problem by skipping certain least significant Booth terms in a PE line. This approach achieves workload orchestration across PEs within the line, as well as bit-serial multipliers inside each PE.
- To utilize the abundant redundancy in attention, we introduce the Leading-Booth computing mechanism in the attention computation to replace full-precision calculations. By computing only the first non-zero Booth term, the computational workload is significantly reduced while maintaining the accuracy.
- Experimental results demonstrate that our Leading-Booth and ACB dynamic workload orchestration mechanisms incur at most a 0.9% accuracy loss. Compared with the SOTA attention accelerators, our design can achieve up to $2.37\times \sim 6.18\times$ improvement in performance and $9.69\times \sim 43.96\times$ enhancement in energy efficiency. Meanwhile, our design can reach up to $1.58\times$ speedup in attention computing performance compared with the SOTA bit-sparse accelerator.

II. BACKGROUND AND MOTIVATIONS

A. Attention Mechanism

As shown in Fig. 1, the attention mechanism consists of multiple linear projections and non-linear activation layers. Initially, input features are projected onto the representations Q, K, and V through three linear transformations. Attention scores are then computed by performing matrix multiplication $Q \cdot K^T$. After scaling, these attention scores (S) are passed to a Softmax function to obtain attention probabilities P. Finally, P is multiplied by V to produce the final output matrix, which represents the weighted combination of V vectors based on the relevance determined by Q and K pairs.

B. Bit-Serial Architecture

Bit-sparse accelerators typically employ a bit-serial architecture to leverage bit-level sparsity. The bit-serial computation paradigm can be denoted as follows:

$$A \times B = \sum A \times b_i \times 2^i = \sum (A \times b_i) << i, \tag{1}$$

where A is fixed and b_i is each bit of B. Basic bit-serial multiplication can be optimized through zero-bit skipping, which leads to the derivation of sparse bit-serial multiplication. However, the irregular distribution of effective bits often results in workload imbalances among the PEs within the PE array, ultimately diminishing the performance of sparse bit-serial computation.

C. Motivation

Our work is inspired by the following considerations. First, bit-level sparsity is abundant in attention-based models such as Transformers. As illustrated in Table I, bit-level sparsity is abundant in the attention computation stage of the BERT [2] models, which is on average $2.28\times$ times over the value sparsity in Table I. This finding motivates us to develop architectures that utilize bit-level sparsity in the attention mechanism. Secondly, as Sanger [5] has demonstrated, the inherent redundancy in the attention mechanism is exceedingly large. For example, Sanger can even achieve 92% attention sparsity without incurring accuracy loss. This insight motivates us to develop aggressive bit-level optimization techniques to exploit the inherent redundancy in attention computation. Finally, unlike the weights that can be predetermined before inference, the Q, K, and V matrices are generated at run-time. Therefore, dynamic mitigation schemes at the hardware level are more suitable for the workload orchestration of attention computation.

Motivated by the above considerations, we propose a sparse bit-serial architecture to leverage the abundant value and bit-level sparsity in attention computation. As shown in Fig. 1, to further exploit the substantial redundancy in attention, we have developed the Leading-Booth computing mechanism for $Q \cdot K^T$. Finally, we propose the ACB workload orchestration mechanism for QKV generation and $P \cdot V$ stages to dynamically address the issue of workload imbalance.

TABLE I
VALUE AND BIT-LEVEL SPARSITY OF BERT-BASE MODEL ON DIFFERENT NLP DATASETS.

Dataset/Tensor		Weight	Embed	QKV	P
VS/**BS** percentage (%)	CoLA	2.7/**49.8**	16.3/**52.3**	2.7/**50.5**	93.2/**98.3**
	MRPC	2.4/**49.4**	7.5/**50.1**	2.6/**50.5**	82.6/**96.9**
	QNLI	3.6/**51.2**	18.1/**53.3**	3.5/**51.3**	93.4/**98.3**
	RTE	3.2/**51.3**	14.8/**54.7**	2.9/**50.6**	82.5/**96.8**
	SST-2	4.4/**53.4**	17.8/**53.3**	2.7/**50.5**	92.8/**98.3**
	STS-B	3.3/**48.8**	8.3/**49.0**	2.6/**50.5**	85.4/**97.1**
	SQuaD	2.6/**52.3**	6.5/**50.5**	2.6/**50.8**	86.0/**98.0**
	CLOTH	3.2/**49.9**	6.0/**53.8**	2.7/**50.5**	82.0/**97.5**

III. OPTIMIZATION SCHEME

A. Efficient Sparse Bit-Serial Multiplication

To harness the bit sparsity inherent in attention, we adopt the sparse bit-serial computing paradigm, which can skip zero partial products. We prefer Adaptive One-sided Bit-serial Multipliers (AOBMs) compared to basic ones because it can lead to better computational performance by serializing the multiplicand with less effective bits [11]. As for the encoding method to extract non-zero term, we choose Booth encoding because it allows for a more natural processing for signed values, and it can figure out more bit sparsity messages in scalar data [10]. The computation of Booth-encoding-based adaptive one-sided bit-serial processing can be denoted as follows:

$$A_1 \times A_2 = \begin{cases} (\sum_{i=0}^{N_1-1} C_{1i} \times 2^{S_{1i}}) \times A_2 & N_1 \le N_2 \\ A_1 \times (\sum_{i=0}^{N_2-1} C_{2i} \times 2^{S_{2i}}) & N_1 > N_2, \end{cases} \tag{2}$$

where C, S, and N refer to non-zero Booth terms, their corresponding shift, and Number of non-zero (Nnz) Booth codes, respectively. The hardware structure of AOBM will be introduced in Section IV-B.

B. Overview of ACB Workload Orchestration Mechanism

The sparse bit-serial multiplication described above focuses on the scalar multiplication (SM) level. To facilitate matrix multiplication (MM) tasks, we need to design a PE array with each PE consisting of multiple AOBMs. Due to the irregular distribution of non-zero Booth terms, the AOBMs exhibit different working cycles, resulting in intra-PE workload imbalances. In our design, PEs within the same row of the array share the same data-fetching state; however, the differing working cycles among same-row PEs lead to inter-PE workload imbalance issues.

To mitigate both the inter-PE and intra-PE workload imbalances, it is essential to synchronize the workloads (WLs) across all PEs and AOBMs. Given that attention-based models are robust and exhibit high tolerance to noise, we propose an Approximate-Computation-Based (ACB) workload orchestration mechanism for the sparse bit-serial PE array structure, in order to achieve inter-PE and intra-PE workload orchestration, as well as the reduction of computation workload. The general idea is that each vector multiplication (VM) task consists of multiple SMs, and approximations can be applied to the last non-zero Booth term of the scalar element to adjust the workload at the SM computation level. The approximate computation mode of scalar multiplication can be illustrated as follows:

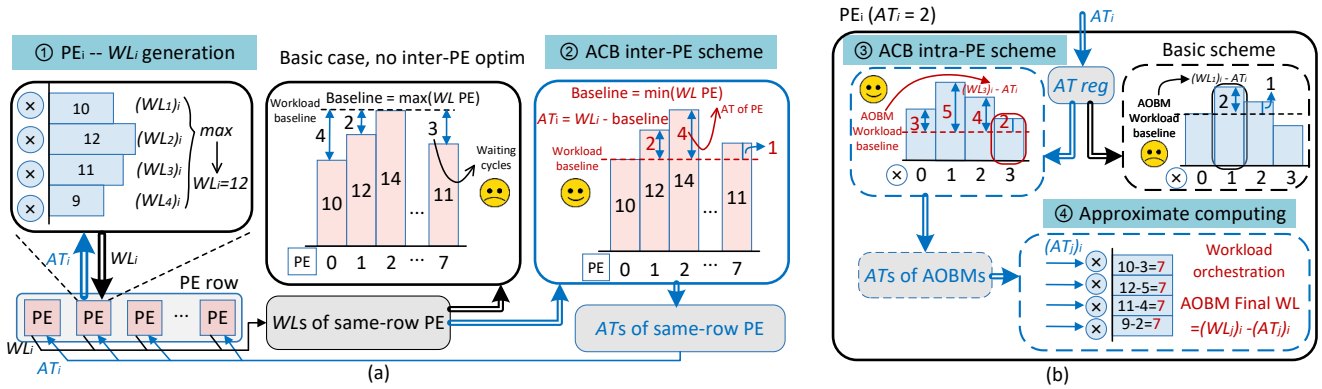


Fig. 2. ACB Workload orchestration analysis, with eight PEs in each row and four AOBMs in each PE. (a) Inter-PE workload orchestration analysis; (b) Intra-PE workload orchestration analysis.

$$A_1 \times A_2 = \begin{cases} (\sum_{i=0}^{\mathbf{N_1-2}} C_{1i} \times 2^{S_{1i}}) \times A_2 & N_1 \le N_2, N_1 > 1 \\ A_1 \times (\sum_{i=0}^{\mathbf{N_2-2}} C_{2i} \times 2^{S_{2i}}) & N_1 > N_2, N_2 > 1, \end{cases} \tag{3}$$

where the least significant bit (LSB) Booth term is skipped if the operand has more than two non-zero Booth terms.

C. Optimization for Inter-PE Workload Orchestration

The ACB workload orchestration mechanism comprises four stages: ① the WL generation, ② the inter-PE and ③ intra-PE Approximation Threshold (AT) generation, and ④ the approximate computing stage. The definition of AT means the quantities of approximate computation that should be introduced, which is equal to the difference between the original and optimized WL. The ④ approximate computing stage will be introduced in Section IV-D. At ① the WL generation stage, the WL of each adaptive one-sided bit-serial multiplier and PE unit is determined by the Nnz data of the processed vector as follows:

$$\begin{cases} (S_{tile})_i = (Nnz_A)_{tile} \le ((Nnz_B)_{tile})_i \\ (WL_j)_i = (S_{tile_j})_i \cdot (Nnz_A)_{tile_j} + \overline{(S_{tile_j})_i} \cdot ((Nnz_B)_{tile_j})_i \\ WL_i = max((WL_1)_i, (WL_2)_i, ..., (WL_{N_{Mult}})_i) \\ i = 1, ..., N_{col}, j = 1, ..., N_{Mult}, \end{cases} \tag{4}$$

with WL of PE determined by the maximum WL of AOBMs. Then the AT of each PE is obtained at ② inter-PE AT generation stage.

Fig. 2 (a) shows the workload orchestration analysis and optimization scheme of the ACB mechanism at the inter-PE level. In the basic case, after all the VM data are prepared, the same-row PEs must wait for the slowest PE which has the maximum WL to complete. Here, we define the orchestrated working cycles as the workload baseline, which is suitable for the description at both PE and AOBM levels. As shown in the ACB inter-PE scheme of Fig. 2 (a), we consider adjusting the workload baseline to the minimum WL and discarding the least significant bit (LSB) non-zero Booth terms in the other heavy-workload VM tasks. The AT of PE is defined as:

$$\begin{cases} WL_{PE_baseline} = min(WL_1, WL_2, ...WL_{N_{col}}) \\ AT_i = WL_i - WL_{PE_baseline}, i = 1, ..., N_{col}, \end{cases} \tag{5}$$

where AT is equal to the difference between each WL of PE and workload baseline which is adjusted to the minimum WL in the PE row. After ② the inter-PE AT generation, the analysis of ③ intra-PE AT generation will be introduced in the following subsection.

D. Optimization for Intra-PE Workload Orchestration

After each PE receives a VM task, each AOBM in the same PE is allocated with a tiled VM task, and the different workloads among the tiled VM tasks cause the intra-PE workload imbalance. For ③ the intra-PE workload orchestration analysis, we decide to apply approximate computing to the AOBM level. The key issue is to determine the AT of each AOBM in one single PE according to the current data, including the AT of PE and WL of each $AOBM$.

The basic approach shown in Fig. 2 (b) allocates the AT of PE to the AOBM with maximum WL, making the AT of this AOBM equal to that of the PE. Consequently, the workload baseline at AOBM level is calculated by subtracting the AT of PE from that maximum-WL value. Although this approach reduces workload imbalance, further optimizations can leverage the error resilience of attention-based models. Like the analysis conducted for the inter-PE case, intra-PE workload orchestration aims to find new orchestrated working cycles for all AOBMs in one PE. As depicted in Fig. 2 (b), compared to the basic approach, we choose to allocate the AT of PE to the AOBM with the minimum WL instead of the maximum-WL one, allowing the AOBM workload baseline to be determined by the difference between the minimum AOBM WL and the PE's AT. The AOBM workload baseline can be formalized as follows:

$$\begin{cases} WL_{AOBM_baseline} = min((WL_1)_i, (WL_2)_i, ..., (WL_{N_{Mult}})_i) \\ (AT_j)_i = AT_i - WL_{AOBM_baseline} + (WL_j)_i \\ i = 1, ..., N_{col}, j = 1, ..., N_{Mult}. \end{cases} \tag{6}$$

Consequently, the ATs for each AOBM can be derived through basic operations between Nnzs and WLs. The AT of each AOBM finally determines the quantities of approximate computation introduced in each tiled VM task, as well as the workload optimization quantity of each AOBM in one PE. As a result, the ACB mechanism enables the PE array to achieve both inter-PE and intra-PE workload orchestration while significantly reducing computation.

E. Leading-Booth Computing Mechanism

There is abundant redundancy in the attention mechanism, thereby allowing for aggressive optimization methods with minor accuracy loss. To leverage this characteristic, we introduce the Leading-Booth computing mechanism into attention computation. Based on the adaptive one-sided paradigm of sparse bit-serial multiplication, the Leading-Booth mechanism retains only the first non-zero Booth term of the value used for serialization, thus providing greater workload mitigation than the ACB mechanism. The computing process of Leading-Booth mechanism can be formalized as follows:

$$A_1 \times A_2 = \begin{cases} (C_1 \times 2^{S_{1i}}) \times A_2 & N_1 \le N_2 \\ A_1 \times (C_2 \times 2^{S_{2i}}) & N_1 > N_2, \end{cases} \tag{7}$$

where only the first non-zero Booth term (C_1 or C_2) is remained in bit-serial multiplication. This approach simplifies the original adaptive bit-serial multiplication into a single-cycle operation.

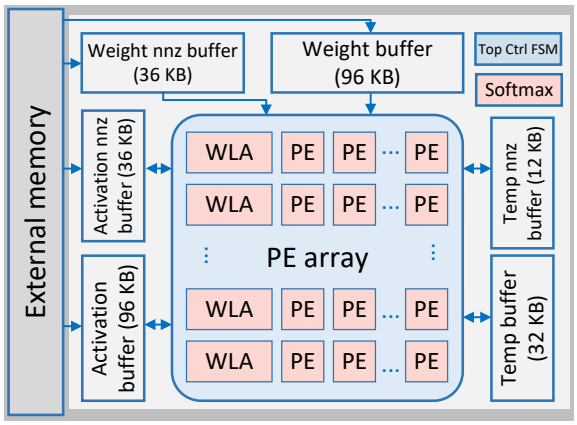


Fig. 3. The overall architecture of Blaze. The light gray part represents our memory design and corresponding size. The red part shows our computing modules and workload analyzers, as well as the Softmax unit.

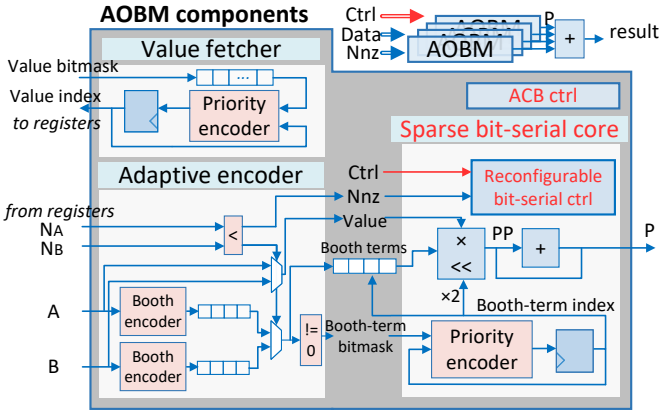


Fig. 4. Structure of the sparse bit-serial PE with four AOBMs.

In DEQ [8], recomputation is necessary after the low-complexity computation of $Q \cdot K^T$ to mitigate accuracy loss. In contrast, the Leading-Booth mechanism retains more operand information, thereby eliminating the need for recomputation. To ensure accuracy preservation, we apply the Leading-Booth mechanism to $Q \cdot K^T$ while still employing the ACB mechanism for the $P \cdot V$ and QKV generation stages. As will be presented in Section V-B, the Leading-Booth mechanism incurs only a negligible loss of accuracy. Our Leading-Booth method is integrated into the PE through a reconfigurable bit-serial computing core (Subsection IV-D), without the need for extra computing engines. As a result, the Leading-Booth mechanism achieves a hardware-friendly computing solution that balances complexity and accuracy effectively.

IV. BLAZE ARCHITECTURE

A. Overall Architecture

The overall architecture of Blaze is illustrated in Fig. 3. Before the inference phase, all embedding data are transferred from the host to the hardware platform. The processing element (PE) array consists of sparse bit-serial PEs and Workload Analyzers (WLAs), with data being transferred from the activation and weight buffers. The output data from matrix multiplication (MM) tasks are stored in temporary buffers. The WLA requires Nnz data for dynamically orchestrated workload assignments, which are computed concurrently alongside activation and weight data for the MM task. Consequently, the initial Nnz values for the embedding and weight data are obtained from the software side, while subsequent Nnz values are generated on the hardware side. The overall data flow in the Blaze architecture is managed by a top-level controlling Finite-State Machine (FSM).

B. Sparse Bit-Serial PE

In our design, each PE comprises four parallel AOBMs, allowing simultaneous processing within one single cycle and achieving an optimal balance between performance and hardware efficiency. The structure of the sparse bit-serial PE is illustrated in Fig. 4. Each AOBM inside the single PE handles one quarter of a tiled vector multiplication (VM) task, which consists of a value fetcher, an adaptive encoder, a sparse bit-serial core, and a controller for the ACB workload orchestration mechanism.

In our design, each 8-bit operand in the data vector is assigned a corresponding 3-bit input Nnz and a generated 1-bit mask, indicating bit-level and value sparsity workloads, respectively. A bitwise AND operation on the value bitmask vectors allows AOBMs to selectively fetch non-zero values via a priority encoder, using the previous output as the mask. This method efficiently obtains non-zero value indices while skipping zeros, avoiding unnecessary computations [16]. Similarly, zero-value Booth terms are excluded using Booth-term bitmask signals, directing the sparse bit-serial core to retrieve non-zero Booth terms through the priority encoder. Consequently, each AOBM's computing process is optimized by leveraging both value and bit-level sparsity in attention-based models, thereby significantly enhancing computational efficiency. To support ACB workload orchestration and Leading-Booth computing modes, we incorporate a reconfigurable sparse bit-serial core within the PE, detailed in Section IV-D. The entire PE operation is executed in a pipelined manner.

C. Workload Analyzer

As described in Section III-C, prior to the implementation of approximate computing, it is essential to acquire the workload (WL) and Approximate Threshold (AT) at both the PE and AOBM levels. The workload analyzer structure in Fig. 5 details the hardware computing process of WL and AT generation. The tile counter records the number of processed operands pairs for one single tiled VM task. The WL of each AOBM within a single PE can be obtained through the operations between tiled Nnz vectors. After the obtainment of WL, the AT at PE and AOBM levels can be computed via further operations introduced in Sections III-C and III-D.

D. Reconfigurable Bit-Serial Computing Core

Once the AT for each AOBM is obtained, these data, along with the corresponding control signals, are transferred to the AOBMs within each PE. As depicted in Fig. 4, the workload-orchestrated sparse bit-serial computing process is managed by the ACB controller and the reconfigurable bit-serial controller in the AOBM, both functioning as counters to regulate the amount of approximate computation at the levels of tiled VM task and fundamental scalar multiplication (SM), respectively. The ACB controller tracks the amount of approximate computation at the tiled VM task level. For each AOBM, with the input AT set as counter threshold, the ACB controller can decide the assignment of approximate computation for specific pairs of scalar operands. When the counter of ACB controller reaches the AT of the AOBM, or when all non-zero values in the current vector tile have been processed, no further approximate computation will be introduced into subsequent SM computations.

The sparse bit-serial controller can switch between the ACB and Leading-Booth computing modes based on the input control signals. This controller operates as a reconfigurable counter. In the Leading-Booth computing mode, the counter threshold is set to one since only the leading Booth term is needed. As illustrated in Fig. 4,

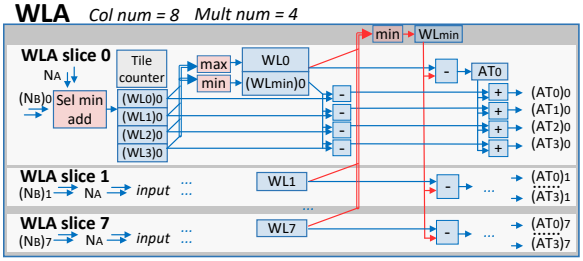


Fig. 5. The WLA structure, which serves to generate WLs and ATs according to the ACB workload orchestration mechanism.

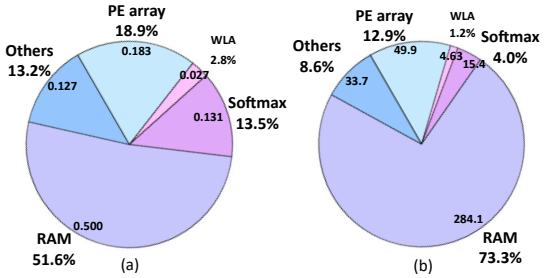


Fig. 6. (a) Area and (b) Power breakdown of Blaze.

in the ACB computing mode, the Nnz output from the adaptive encoder serves as the initial counter threshold. As introduced in Section III-B, in our approximate computing mechanism, the last Booth term from the adaptive encoder output shown in Fig. 4 is discarded, resulting in an extra reduction of the initial counter threshold by one. To prevent significant computational discrepancies and accuracy loss, we stipulate that if the Nnz output of the adaptive encoder in the AOBM is equal to one, approximate computing will not be applied to that particular operand pair. Consequently, the reconfigurable bit-serial computing core efficiently implements both Leading-Booth computing and the approximate computing stage of the ACB mechanism within our Blaze architecture.

V. EVALUATION

A. Evaluation Setup

Benchmarks. To validate the computing performance of our bit-level optimization schemes, we select the Bert-base model and fine-tune it on the SQuaD [17], CLOTH [18], and six GLUE [19] text classification tasks. The evaluation metrics are Matthews correlation for CoLA, F1 for MRPC, accuracy for QNLI, RTE, SST-2 and CLOTH, Pearson correlation for STSB, and exact-match for SQuaD. We have developed our coding experiments based on PyTorch [20] and Huggingface [21]. We fine-tune the pretrained model for 5 epochs using the AdamW optimizer. For the GLUE, SQuad and CLOTH datasets, the maximum sequence length for BERT is set to 128.

Hardware implementation. We implement the Blaze architecture in Verilog HDL and synthesize it using the Synopsys Design Compiler at 500MHz under TSMC 28nm HPC+ technology. The power consumption of SRAM-based on-chip buffers is obtained by CACTI 7 [22]. Fig. 6 provides the detailed area and power breakdown of Blaze. Our PE array is configured as 8×8 with four AOBMs placed in each PE. Since each 8-bit value can be encoded into four Booth terms and each AOBM tackles one Booth-term multiplication in a single cycle, the computing capacity of four AOBMs in each PE is equal to one 8-bit multiplier per PE. The total SRAM capacity is 308KB, which is made up of input activation, weight, temporary data and their corresponding Nnz buffers. Our accelerator has a total area of $0.968mm^2$ and a total power of $387.7mW$. We develop a cycle-accurate simulator for our proposed Blaze architecture to collect statistics on computation, buffer usage, and external memory accesses for each attention computation workload. For off-chip traffic, we follow DEQ [8] and assume 20 pJ/bit for energy consumption, with a 256 bits/cycle off-chip bandwidth.

B. Accuracy Evaluation

Table. II presents the accuracy results of Sanger [5], DTQAtten [6], DEQ [8], and Blaze. We conducted the Leading-Booth mechanism in $Q \cdot K^T$ and replaced the rest matrix multiplication in attention as the ACB workload-orchestrated matrix multiplication. The accuracy results show that Blaze achieves -0.08%, -0.58%, 0.70%, 0.03% accuracy improvement on average over baseline, Sanger, DTQAtten, and DEQ, respectively. Our method incurs at most only 0.9% accuracy degradation in the MRPC dataset, while in the QNLI dataset our methods achieve the highest accuracy results. These accuracy results indicate that, although our approaches are lossy, their impact on the model's accuracy is minimal.

Note that unlike Sanger, DTQAtten, and DEQ, our methods also apply to the linear layers in attention while these designs can only accelerate the $Q \cdot K^T$ and $P \cdot V$ stages. Although introducing more lossy components to attention without recomputation stages, our work still achieves comparable accuracy results compared with DEQ. At the $Q \cdot K^T$ stage, our Leading-Booth mechanism employs only 3-bit multiplication with no recomputation, while DEQ performs 4-bit multiplication and needs recomputation to mitigate accuracy loss. We believe that this difference arises because DEQ utilizes the 4-bit data statically, while our Leading-Booth Mechanism dynamically utilizes the first effective 3-bit (Booth term) data.

C. Performance Comparison with SOTA Attention Accelerators

We have conducted extensive experiments on SQuAD, CLOTH, and six GLUE datasets to compare our accelerator with state-of-the-art (SOTA) attention accelerators. We create cycle-accurate simulators for Sanger, DTQAtten, and DEQ to evaluate their performance statistics considering both algorithmic and hardware profits. For a fair comparison, we restrict the accelerators to the same peak computing capacity and buffer size. Fig. 7 presents performance results normalized to Sanger. Experimental results show that Blaze achieves average normalized performance gains of $6.18\times$, $5.13\times$, and $2.37\times$ over Sanger, DTQAtten and DEQ, respectively. Regarding energy efficiency, our accelerator improves efficiency by $43.96\times$ compared to Sanger and $9.69\times$ compared to DTQAtten.

The performance and energy efficiency gains of Blaze over Sanger, DTQAtten, and DEQ stem from three key aspects. First, Blaze effectively exploits bit-level sparsity. While Sanger, DTQAtten, and DEQ adopt optimization methods for data flow and processing element utilization, they overlook the abundant bit-level sparsity inherent in attention-based models. In contrast, Blaze employs an efficient sparse bit-serial PE design. Second, building on the basis of the sparse bit-serial architecture, Blaze incorporates the ACB mechanism for workload orchestration, discarding certain least significant Booth terms to optimize computation workloads. Importantly, the ACB mechanism coordinates workloads across the entire PE array at both the inter-PE and intra-PE levels, significantly reducing idle cycles and enhancing computation resource utilization.

Finally, Blaze optimizes the computation workload of $Q \cdot K^T$ using the Leading-Booth mechanism. Unlike DEQ, which requires high-precision recomputation to ensure accuracy, the Leading-Booth mechanism for $Q \cdot K^T$ requires no additional recomputation because of its strong information retention capability, thus greatly optimizing the computation workload without sacrificing accuracy. While Sanger and DTQAtten also avoid additional recomputation, their methods involve the removal of certain elements through pruning and the

TABLE II
ACCURACY RESULTS OF DIFFERENT DESIGNS ON NLP DATASETS.

Dataset	CoLA	MRPC	QNLI	RTE
Baseline	55.7	89.8	91.1	63.2
Sanger	57.2	88.7	91.0	66.4
DTQAtten	54.3	89.7	90.5	63.0
DEQ	58.3	89.6	90.8	62.4
Blaze	**55.3**	**88.9**	**91.5**	**65.1**
Dataset	**SST-2**	**STS-B**	**SQuaD**	**CLOTH**
Baseline	92.8	88.5	81.3	79.6
Sanger	93.1	88.6	81.4	79.6
DTQAtten	92.2	/	80.5	/
DEQ	92.5	88.4	80.4	78.8
Blaze	**92.3**	**87.9**	**81.1**	**79.3**

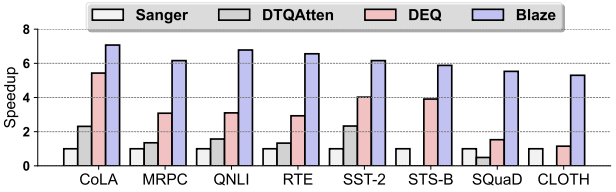


Fig. 7. Speedup of attention accelerators, with the performance of Sanger set as the normalized standard.

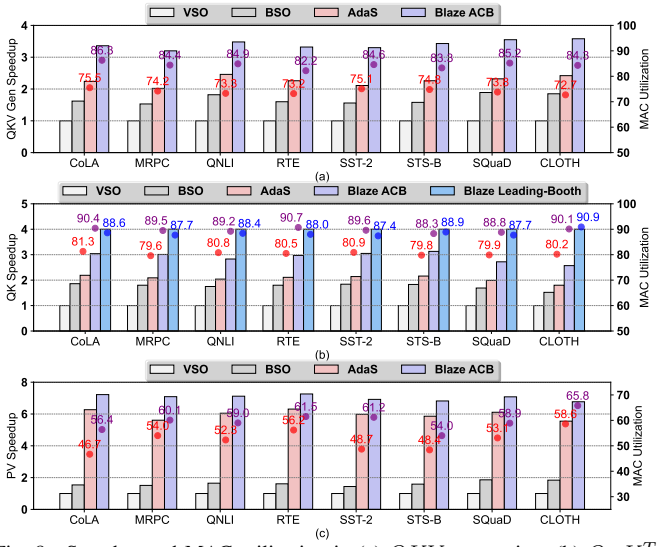


Fig. 8. Speedup and MAC utilization in (a) QKV generation, (b) $Q \cdot K^T$, and (c) $P \cdot V$ stages of different computing mechanisms (AdaS, Blaze ACB and Leading-Booth for MAC utilization), with the performance of VSO set as the normalized performance standard.

application of mixed-precision techniques. These methods, however, introduce a risk of omitting essential information, thus making it challenging to realize substantial performance gains when operating within stringent accuracy requirements.

D. Effectiveness of Workload Orchestration Optimization

We have conducted experiments to compare with the SOTA bit-sparse accelerator and evaluate the effectiveness of our workload-orchestration optimization schemes. We develop a cycle-accurate simulator to collect the performance and MAC utilization statistics of the bit-sparse accelerator AdaS on the SQuaD, CLOTH, and six GLUE datasets. Meanwhile, cycle-accurate simulators are also created for the value-sparsity optimized (VSO) and bit-sparsity optimized (BSO) multiplication mechanisms for performance comparison, which merely skip non-zero operands and non-zero Booth terms, respectively, without workload orchestration or other optimizations. All mechanisms are evaluated with the same peak computing capacity and buffer size. Fig. 8 shows the performance comparison of these computing mechanisms at different matrix multiplication stages in attention, including QKV generation, $Q \cdot K^T$, and $P \cdot V$.

According to the results, our ACB mechanism achieves $3.40\times$, $2.03\times$, and $1.51\times$ performance gains on average on the linear layers of QKV generation compared to VSO, BSO, and AdaS, respectively. Our ACB mechanism performs better in the $P \cdot V$ computation, reaching on average $7.04\times$ and $4.35\times$ performance gains compared to VSO and BSO. This is due to the higher value and bit-level sparsity at the $P \cdot V$ stage, and our ACB mechanism can make use of both kinds of sparsity to improve computing performance. Although our ACB mechanism achieves a closer performance with AdaS at the $P \cdot V$ stage, the $P \cdot V$ stage accounts for a non-dominant fraction of the computation burden among the whole attention computation process. For evaluations on $Q \cdot K^T$, our Leading-Booth mechanism achieves on average $4.00\times$, $2.28\times$, and $1.94\times$ performance gains compared to the VSO, BSO, and AdaS.

In terms of computing mechanisms, the sparse bit-serial accelerator AdaS is lossless. Recognizing that attention-based models exhibit robust performance, we develop a lossy ACB workload orchestration mechanism that enhances performance with minimal accuracy loss. While AdaS employs a group-shuffling bidirectional-innerjoin mechanism to address workload imbalance issues [11], it only addresses intra-PE imbalances, leaving the inter-PE imbalances unresolved. Our ACB mechanism effectively alleviates both the intra-PE and inter-PE workload imbalances. As illustrated in Fig. 8, our ACB mechanism achieves average increase by 10.55%, 9.20%, and 7.36% in MAC utilization compared to AdaS during QKV generation, $Q \cdot K^T$, and $P \cdot V$, respectively, thereby showing that the ACB mechanism not only optimizes computation workload, but also achieves more effective workload orchestration and greater computation resource utilization. Additionally, our Leading-Booth mechanism enhances MAC utilization by 8.08% versus AdaS. By leveraging approximate computation, we improve both workload quantities and orchestration, yielding a performance boost of $1.42\times$ to $1.58\times$ compared to AdaS across six GLUE datasets, as well as the SQuAD and CLOTH datasets. Notably, our Leading-Booth mechanism also achieves $1.38\times$ performance gains over the ACB mechanism, as it simplifies the multi-cycle sparse bit-serial multiplication in ACB to a single-cycle process.

VI. CONCLUSION

In this work, we propose Blaze, an efficient bit-sparse accelerator for attention-based models. To alleviate the commonly existing load imbalance problem, we propose an Approximate-Computing-Based workload orchestration mechanism. Besides, we develop the Leading-Booth computing mechanism to further utilize the redundancy in attention. The experimental results show that our method incurs only at most a 0.9% accuracy loss compared with the baseline. At the hardware level, our accelerator can achieve $2.37\times \sim 6.18\times$ performance gains, and $9.69\times \sim 43.96\times$ energy efficiency compared with SOTA attention accelerators. Our accelerator can achieve up to $1.58\times$ speedup in attention computing performance compared with the SOTA bit-sparse accelerator AdaS.

ACKNOWLEDGEMENTS

This work was financially supported in part by National Key Research and Development Program of China under Grant 2021YFA1003602, in part by Project of Shanghai EC under Grant 24KXZNA12, and in part by National Natural Science Foundation of China (NSFC) Research Project 92373207. The computations in this research were performed using the CFFF platform of Fudan University.

REFERENCES

[1] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[3] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[4] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 97–110, IEEE, 2021.

[5] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 977–991, 2021.

[6] T. Yang, D. Li, Z. Song, Y. Zhao, F. Liu, Z. Wang, Z. He, and L. Jiang, "Dtqatten: Leveraging dynamic token-based quantization for efficient attention architecture," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 700–705, IEEE, 2022.

[7] T. Yang, F. Ma, X. Li, F. Liu, Y. Zhao, Z. He, and L. Jiang, "Dtatrans: Leveraging dynamic token-based quantization with accuracy compensation mechanism for efficient transformer architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 509–520, 2022.

[8] X. Wang, Z. Song, Q. Huang, and X. Liang, "Deq: Dynamic element-wise quantization for efficient attention architecture," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*, pp. 623–630, 2023.

[9] J. Albericio, A. Delmás, P. Judd, S. Sharify, G. O'Leary, R. Genov, and A. Moshovos, "Bit-pragmatic deep neural network computing," in *Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture*, pp. 382–394, 2017.

[10] S. Sharify, A. D. Lascorz, M. Mahmoud, M. Nikolic, K. Siu, D. M. Stuart, Z. Poulos, and A. Moshovos, "Laconic deep learning inference acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 304–317, 2019.

[11] X. Lin, G. Li, Z. Liu, Y. Liu, F. Zhang, Z. Song, N. Jing, and X. Liang, "Adas: A fast and energy-efficient cnn accelerator exploiting bit-sparsity," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[12] W. Sun, Z. Zou, D. Liu, W. Sun, S. Chen, and Y. Kang, "Bit-balance: Model-hardware co-design for accelerating nns by exploiting bit-level sparsity," *IEEE Transactions on Computers*, 2023.

[13] A. Li, H. Mo, W. Zhu, Q. Li, S. Yin, S. Wei, and L. Liu, "Bitcluster: Fine-grained weight quantization for load-balanced bit-serial neural network accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4747–4757, 2022.

[14] G. Wang, S. Cai, W. Li, D. Lyu, and G. He, "Bsvit: A bit-serial vision transformer accelerator exploiting dynamic patch and weight bit-group quantization," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[16] A. Gondimalla, N. Chesnut, M. Thottethodi, and T. Vijaykumar, "Sparten: A sparse tensor accelerator for convolutional neural networks," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 151–165, 2019.

[17] P. Rajpurkar, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[18] Q. Xie, G. Lai, Z. Dai, and E. Hovy, "Large-scale cloze test dataset created by teachers," *arXiv preprint arXiv:1711.03225*, 2017.

[19] A. Wang, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[20] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[21] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

[22] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

LightRIM: Light Runtime Integrity Measurement for Linux Kernels in Embedded Applications

Yili Guo[1], Zhuoran Ma[1], Xiangyue Li[1], Jiajia Huang[1], Wanli Chang[1,2]
[1]Hunan University, China [2]Huawei Technologies, China

***Abstract*—Linux kernels are being widely deployed in embedded applications, such as increasingly automated vehicles and robots, due to their robust ecosystem. Security modules have been developed to enhance the integrity of Linux kernels, a critical system component. However, these modules consume substantial computational resources, making them unsuitable for embedded domains. We introduce LightRIM, a lightweight method to measure the Linux kernel's integrity during runtime, ideal for resource-limited embedded applications. We focus on major attack types and extract objects for monitoring. Our approach includes a two-stage hashing process and an event-triggered measurement algorithm tied to the security value. To mitigate Time-of-Check-to-Time-of-Use (TOCTOU) attacks, we introduce a heuristic algorithm that maximizes the attack detection rate within CPU usage constraint and randomizes the measurement intervals. Experimental results indicate that LightRIM incurs less than 0.7% performance overhead while providing extensive attack coverage.**

I. Introduction

Linux kernels are prevalent in embedded systems, particularly in intelligent and connected applications, owing to their strong ecosystem and dependability. Security is of high importance, as any breach of kernel integrity can lead to dire outcomes. Unauthorized alterations to kernel code or data structures may cause system malfunctions, vulnerabilities, or total collapse. Thus, preserving the integrity of Linux kernels during runtime is vital for the security and stability of the entire embedded system.

Linux Security Module (LSM) has been developed to enhance the integrity of the Linux kernel. Among its components, the Integrity Measurement Architecture (IMA), introduced in 2004 and based on Trusted Computing Group (TCG) standards, has driven significant advancements in both static and dynamic integrity measurement techniques [1], [2]. Nevertheless, IMA's hash algorithms incur significant computational overheads [3], and measuring large files can result in substantial storage demands [4].

Related Works: Jaeger et al. [5] proposed PRIMA, a prototype based on IMA and integrated with SELinux policies to focus on trusted subject interactions, reducing overhead. However, its reliance on PRIMA-aware applications and the required OS modifications limit its practicality. Zhang et al. [6] demonstrated that SELinux integration significantly degrades kernel performance, even with policy enforcement disabled. Then, DRIVE [7] introduced a method for detecting runtime attacks by continuously monitoring memory images of binary files and verifying their consistency with the loaded binary code. However, this approach mainly focuses on memory image measurements, overlooking other critical objects, which limits its comprehensiveness. Meanwhile, Xfilter [8] provides an LSM policy framework with package management integration but relies on extensive customization and system-wide monitoring, making it less suitable for embedded systems. Bohling et al. [9] demonstrated that IMA's security guarantees can be undermined by malicious block devices and investigated inherent Time-of-Check-to-Time-of-Use (TOCTOU) vulnerabilities within Linux's IMA framework. The problem of TOCTOU invalidity is widespread in various fields such as remote attestation, operating systems, and certifications. Numerous approaches have emerged to address TOCTOU vulnerabilities across diverse contexts [10], [11]. However, these methods mainly rely on hardware-based remote attestation to counteract TOCTOU attacks. Embedded systems frequently depend on hardware-supported secure boot mechanisms to ensure system integrity at the hardware level. These mechanisms include secure boot chains, roots of trust, and bootloaders [12]. Although these solutions provide robust security during booting, they cannot prevent run-time manipulation and often impose significant hardware resource overhead [13].

This paper presents LightRIM, a lightweight runtime integrity measurement architecture for monitoring immutable variables on Linux systems. Critical measurement objects, identified through integrity-compromising attack analysis, are hashed and stored in a baseline library. To minimize storage, a two-stage hashing mechanism aggregates these objects into a single hash. An event-triggered measurement algorithm dynamically adjusts security values for finer granularity. To mitigate TOCTOU attacks, we model the relationship between system overhead and measurement intervals, proposing an optimization algorithm to maximize detection rates under CPU constraints. Benchmarks and real-world attack evaluations validate LightRIM's efficiency and robustness.

The contributions of this paper are as follows:

- We develop a baseline library for storing critical runtime objects and propose a two-stage hashing mechanism with an event-triggered measurement algorithm.
- We formulate a Simulated Annealing (SA)-based optimization algorithm to maximize TOCTOU attack detection rates under predefined CPU usage constraints.
- We demonstrate that LightRIM incurs less than 1 ms latency in LMBench tests and increases system overhead by

This work was supported by the National Key Research and Development Program of China (2023YFB4503704).
Corresponding Author: Wanli Chang (wanli.chang.rts@gmail.com).

under 0.7% in SPEC CPU2006 tests, and exhibits robust detection capabilities against real-world attacks. Our code is available at https://github.com/ElyerGuo/LightRIM.git.

II. Background

A. Linux Kernel Integrity Attacks

The ATT&CK framework is a globally accessible knowledge base constructed on real-world attack vectors. From this framework, we select Linux integrity-related attacks, focusing on two representative examples: code injection and rootkit attacks.

Code Injection Attack. The ATT&CK library includes attacks such as Virtual Dynamic Shared Object (VDSO) Hijacking (T1055.014), Ptrace System Calls (T1055.008), and Dynamic Linker Hijacking (T1574.006). Taking VDSO as an example, it is a dynamic library in the kernel that can be mapped to the user space. Adversaries may hijack the system call interface code stubs mapped from VDSO shared objects to processes to execute system calls and open and map malicious shared objects. This attack involves modifying the kernel VDSO page with the set_memory_rw function, allowing user space to directly modify VDSO and hijack it for shellcode (kernel arbitrary code execution vulnerability). Next, it modifies the kernel VDSO data and writes shellcode to hijack functions in the user space. When a high-privileged process calls specific functions in VDSO, it triggers the shellcode (arbitrary kernel read/write vulnerability).

Rootkit Attack. The ATT&CK library provides procedural examples for various rootkit attacks (T1014). Adversaries usually manipulate the core components or system tools of the operating system using rootkit attacks, allowing them to gain unauthorized access to the system and maintain long-term control. For example, adversaries may modify the source code of the Linux kernel or load malicious kernel modules to take control and conceal the system. Additionally, they may alter the behavior of system tools such as `ps` and `ls` to conceal their activities or provide false information.

B. Threat Model

As a runtime integrity measurement solution for Linux systems, our threat model and assumptions align with prior research [14]–[16]. The primary considerations are as follows:

- We assume that the Linux system is correctly loaded, fully functional at boot, and actively running, though it may contain vulnerabilities that allow attackers to interact with the kernel post-boot.
- The attacker possesses complete control of non-root user-mode processes and can interact with the kernel through standard kernel interfaces.
- Other kernel integrity measurement methods, such as IMA and control flow integrity (CFI) techniques, are assumed to be established.
- LightRIM is assumed to be deployed in a trusted setting, like a Trusted Platform Module (TPM)-protected environment, safeguarding the integrity of the baseline library against unauthorized modifications.

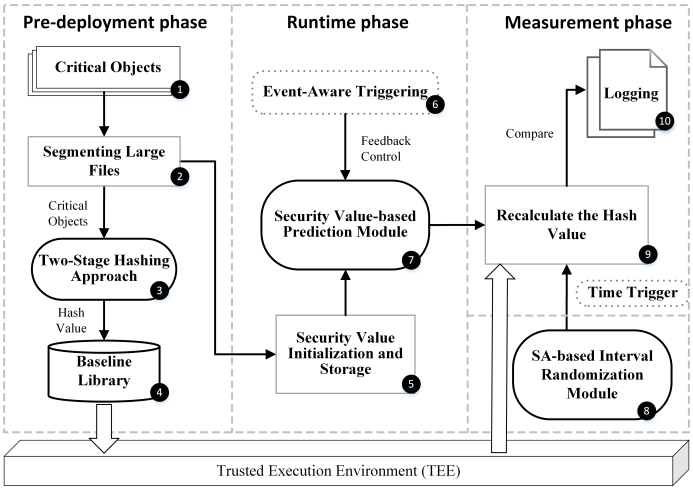


Fig. 1: Overview of LightRIM.

In this threat model, the attacker's objective is to tamper with critical system components to compromise the kernel, escalate privileges, or disrupt normal system operation.

III. Lightweight Runtime Integrity Measurement Framework Design

A. LightRIM Framework Overview

As shown in Fig. 1, the lightweight runtime integrity measurement framework consists of three phases: the Pre-deployment, Runtime, and Measurement phases. In the Pre-deployment phase, ❶ all critical objects defined by integrity attack models and immutable during runtime are extracted from the OS kernel and physical memory. ❷ Large objects, such as the kernel code and data segments, are divided into smaller sections to reduce system load. ❸ A two-stage hashing approach generates hash values for these objects, which are then ❹ stored in the baseline library. During the Runtime phase, ❺ critical object's security values are initialized and stored in the prediction module. ❻ An event-triggered mechanism evaluates measurement conditions, while ❼ security values are dynamically updated via the kernel notification chain based on system events. In the Measurement phase, ❽ an SA-based interval randomization schedules measurement operations to counter TOCTOU attacks. ❾ The current hash values of critical objects are recomputed and compared with the baseline. Any mismatch ❿ triggers an alert, signaling a possible integrity breach. This design ensures efficient integrity monitoring with minimized overhead and enhanced security.

B. Identifying and Extracting Critical Objects

In various types of integrity attacks, distinct attack targets have critical measurement objects that are vulnerable and require regular protection. Code injection attacks commonly target areas such as the kernel code segment, kernel data segment, and kernel libraries, attempting to alter the kernel's execution flow. Rootkit attacks, on the other hand, are more likely to compromise critical data structures like the system call table and interrupt descriptor table, aiming to conceal

malicious actions or establish persistent control. Additionally, user-space system processes are frequently targeted by malicious threats, further elevating the overall risk to the system. These attacks often exploit various techniques to infiltrate the system during runtime, thereby compromising the operating system's integrity. Consequently, ensuring the real-time protection of these critical objects is essential to maintain system security.

To protect critical information such as the kernel code segment, kernel data segment, and system call table, we employ the `kallsyms_lookup_name` kernel function. This function allows us to locate the addresses of kernel symbols (typically functions or variable names) within the kernel space by their symbol names. Using these addresses, we can access and extract the contents of the corresponding measurement objects for integrity protection. For kernel libraries, such as the VDSO, we directly secure the user-space VDSO library. Regarding process-related information, we access the `mm_struct` to obtain the process address space and apply protection to critical information at the corresponding addresses. Our approach specifically targets processes present at system startup, excluding any processes initiated during runtime.

C. Coarse-Grained Two-Stage Hashing Approach

This subsection describes a two-stage hashing mechanism designed to reduce the storage and comparison overhead of the baseline library. As shown in Fig. 2, we first evaluated several widely used hash algorithms within the Linux system to identify a suitable trade-off between performance and security. Among these, SHA-256 was selected due to its robustness and acceptable computational cost.

In the first stage, we apply the SHA-256 algorithm to each of the extracted measurement objects, which include kernel code and data segments, system call tables, the file system, kernel modules, and user process memory regions. For each object, SHA-256 computes a 256-bit hash value that uniquely represents its content. These individual hash values are stored in the baseline library and denoted as $HV_1, HV_2, \ldots$ in Fig. 2. To further reduce the verification overhead during runtime, a second-stage aggregation process is introduced. Specifically, only the first 32 bits of each individual hash value are selected and computed into a intermediate vector. The vector is then re-hashed using SHA-256 to generate a final aggregated hash value, referred to as `ALL_HASH_VALUE`. Both the individual hash values and this final aggregated hash value are stored in the reference database using kernel file operation functions. This two-stage approach enables fast integrity verification with minimal storage and computational costs.

D. Fine-Grained Event-Triggered Measurement Algorithm

This subsection will cover the fine-grained event-triggered measurement algorithm, consisting of the security value-based prediction and the event-aware modules.

Security Value-Based Prediction Module. The prediction module leverages dynamically adjusted security values to

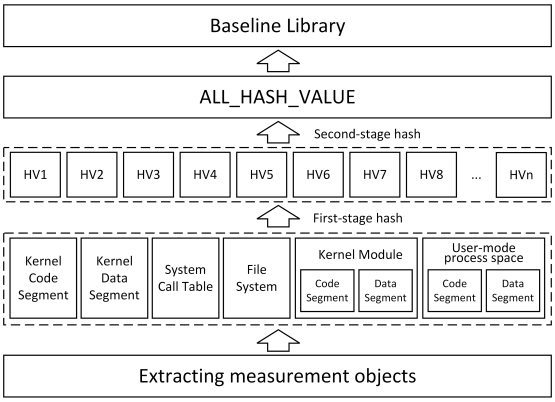


Fig. 2: The architecture of the two-stage hashing process.

determine whether a measurement object should be evaluated during each cycle. At the start of measurement, a random value is generated using a random seed and compared against the object's current security value. If the security value is lower, the object is measured, and its security value increases; otherwise, the security value decreases, raising its likelihood of measurement in subsequent cycles. Upon detecting an attack, the object's security value is significantly reduced, substantially increasing its measurement probability in future cycles. Additionally, an integrated event-triggering module dynamically adjusts security values based on system events, ensuring adaptability to runtime conditions.

Event-Aware Triggering. Deploying our architecture in kernel mode requires real-time detection of critical user-space events, including process creation/termination, kernel module loading/unloading, changes to process code segment permissions, and shell script execution. These behaviors indicate potential changes to the system's integrity in both kernel and user spaces. By monitoring these events, the architecture substantially improves overall system security. Since all these behaviors depend on system calls bridging user and kernel spaces, they can be effectively tracked by instrumenting system call functions. Algorithm 1 outlines the event-triggering mechanism that informs the measurement module and initiates necessary actions. Initially, a kernel notification chain is established, enabling kernel components to register as observers for specific events (line 2). The measurement module registers a notification function to handle event updates (lines 3–5). When an instrumented function detects a critical system call, it triggers the notification chain and passes a notification number to the measurement module (line 6). The measurement module then processes the event by adjusting security values as required (lines 7–10).

E. SA-based Attack Detection Rate Optimization Algorithm

The periodic measurement approach is vulnerable to TOCTOU attacks, as attackers can predict and exploit fixed intervals. Current solutions overlook the balance between randomized measurement intervals and system overhead, and lack a systematic model for attack detection rates. Frequent measurement triggers can lead to significant system overhead,

Algorithm 1 Event Aware Module

Input: Event i is triggered
1: $S_i \leftarrow$ The security value of object i changed
2: *RAW_NOTIFIER_HEAD(notifier_chain)* ▷ Define kernel notification chain header
3: *kprobe_trigger_function* ▷ Kprobe definition
4: *orig ← kallsyms_lookup_name(trigger_function)* ▷ Find the function address
5: *register_kprobe(&kprobe_trigger_function)* ▷ Register kprobe
6: *handler_pre_trigger ← notify_observers(i)*
7: The kernel notification, indexed by i, precedes the instrumentation function trigger
8: **if** *notifier_callback* == i **then**
9: ADD(S_i) ▷ Change the security value
10: **end if**

while infrequent measurements drastically reduce attack detection rates. Therefore, it is essential to design a randomized measurement time sequence that balances these trade-offs, maximizing attack detection rates while satisfying CPU usage constraints in limited systems. To address these problems, we developed a model that quantifies system overhead in terms of CPU usage as a function of measurement intervals and designed an SA-based heuristic algorithm to maximize detection rates within predefined CPU usage constraints, thereby reducing TOCTOU vulnerabilities.

1) Model Fit Function: To quantitatively assess the impact of measurement interval lengths on CPU usage, we conducted over 100 trials per interval on the experiment platform described in Section IV, without additional user-initiated processes. The measurement intervals ranged from 0 to 10,000 milliseconds, and average CPU usage was determined for each interval. Given the observed oscillations and exponential decay in CPU usage, we applied a logarithmic transformation to linearize the data, which followed the exponential function $y = ae^{-bx} + c$. This suggested an exponential decay model, with variations expected across different platforms. Subsequently, we employed Python's function $scipy.optimize.curve_fit$, leveraging the Levenberg-Marquardt algorithm for nonlinear least squares fitting, to fit this model to our data. The resulting fitted equation is as follows:

$$\ln(u_i) = 2.64458 \times e^{-0.00023 \times m_i} - 0.3952 \qquad (1)$$

where u_i represents the average CPU usage when the measurement module is active and m_i denotes the specified periodic measurement interval. From our calculations, the fitted function in Equation 1 achieved a coefficient of determination (R^2) of 0.99 and a Root Mean Square Error (RMSE) of 0.25, demonstrating a highly effective fit.

2) SA-based Optimization Algorithm Design: The SA-based heuristic algorithm is widely recognized for efficiently identifying optimized solutions within a search space. We propose an SA-based algorithm to generate randomized measurement time points , enhancing the detection rate of TOCTOU attacks. The algorithm is detailed in Algorithm 2, takes the vulnerability time window ω and CPU usage upper limit u_{max} as inputs, and output the optimized measurement sequence θ along with the maximum TOCTOU attack detection rate

Algorithm 2 SA-based Attack Detection Rate Optimization Algorithm

Input: ω, u_{max}
Output: θ, λ_{max}
1: T_{ini}=100, T_{ter}=0.01, δ= 0.98;
2: $m_{\text{k}} \leftarrow$ Compute from u_{max} using Equation (1);
3: $\theta_{\text{k}} \leftarrow$ `Generate_Time_Points`$(m_{\text{k}}, n \times m_{\text{k}})$;
4: $\lambda_{\text{k}} \leftarrow$ `Simulate_TOCTOU_Attack`$(\omega, \theta_{\text{k}})$;
5: $T = T_{\text{ini}}$, $\lambda_{\text{max}} = \lambda_{\text{cur}} = \lambda_{\text{k}}$;
6: **while** $T > T_{\text{ter}}$ **do**
7: $\theta_{k'} \leftarrow$ `Generate_Time_Points`(m_i, m_j) ▷ $m_i, m_j \in (m_k, n \times m_k)$
8: $\lambda_{k'} \leftarrow$ `Simulate_TOCTOU_Attack`$(\omega, \theta_{k'})$
9: **if** $\lambda_{k'} > \lambda_{\text{k}}$ **or** $e^{(\lambda_{k'} - \lambda_{\text{k}}) \div T} > random(0,1)$ **then**
10: $\theta_{\text{k}} = \theta_{k'}$, $\lambda_{\text{cur}} = \lambda_{k'}$;
11: **if** $\lambda_{\text{max}} < \lambda_{k'}$ **then**
12: $\lambda_{\text{max}} = \lambda_{k'}$, $\theta = \theta_{k'}$;
13: Save the randomly generated range (m_i, m_j);
14: **end if**
15: **end if**
16: $T = T \times \delta$
17: **end while**

λ_{max}. The algorithm first initializes the SA parameters (lines 1-5). Then it generates the minimal measurement interval m_{k} under the constraint u_{max}, and an initial measurement sequence θ_{k} using the `Generate_Time_Points` function. The initial TOCTOU detection rate is computed using the `Simulate_TOCTOU_Attack` function. We simulate discrete attack events randomly generated within a time window ω, where a modification is applied to a specific measurement object. Assuming that the attacker requires a certain amount of time to execute the operation, the attack is considered detected if the measurement point identifies the event. The attack detection rate λ is defined as the ratio of detected attacks to the total number of attacks. Then, the SA temperature iteration is performed (lines 6-17). To generate a new time sequence, the `Generate_Time_Points` function moves within the solution space from m_{k} to $n \times m_{\text{k}}$, generating a new random interval (m_i, m_j). After generating the sequence, an attack simulation yields the detection rate $\lambda_{k'}$. The new sequence is accepted if it shows a higher detection rate or meets the probability acceptance criterion (lines 9-15). If $\lambda_{k'} > \lambda_{max}$, the θ, λ_{max} are updated (lines 11-13).

IV. Evaluation

A. Implementation

LightRIM is designed to be easily integrated into the Linux kernel with root privileges, enabling straightforward deployment across diverse Linux systems. This approach significantly streamlines the deployment process in practical applications.

Baseline Library Management. Dynamic measurement objects are recalculated during each reboot to reflect state changes, while static objects are updated only after system modifications.

Adaptive Measurement Strategy. An adaptive module dynamically switches between coarse-grained and fine-grained methods based on CPU load. Coarse-grained methods reduce overhead during high system loads, while fine-grained meth-

TABLE I: Comparison of Hashing Mechanism Overhead.

Category	CPU Usage (%)	Above Baseline(%)
Baseline	0.25	-
Hashing	2.056	1.806
Two-stage hashing	1.628	1.378

ods provide stronger security assurances under lighter loads, ensuring a robust yet efficient monitoring system.

B. Experimental Setup

1) Platform: The experiments were conducted on a 64-bit Ubuntu 22.04 system running Linux kernel version 5.10.0-1057-oem, featuring an 8-core Intel i5-12400 processor and 8GB RAM.

2) Benchmarks: We evaluate the runtime performance of LightRIM by applying two benchmarks: LMBench 3.0 for detailed component analysis and the SPEC CPU2006 suite for comprehensive system evaluation, as done in previous research [15], [17]. The benchmarks are first run on a system with IMA enabled to establish baseline performance. Next, our module is loaded into the Linux kernel with the measurement functionality enabled, followed by another round of benchmark tests. The results are labeled as IMA and LightRIM+IMA, respectively.

C. Comparison of CPU Usage between Hashing and Two-Stage Hashing

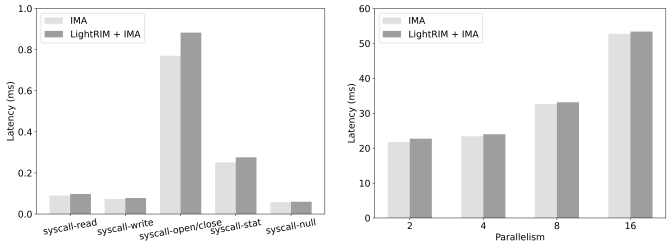
To evaluate the effectiveness of the coarse-grained measurement mechanism, we analyzed CPU usage under three scenarios: system idle, one-stage hashing, and two-stage hashing, as illustrated in Table I. The results demonstrate that the two-stage hashing mechanism reduces CPU overhead by 23.7% compared to one-stage hashing, excluding baseline overhead. This improvement is attributed to the design of the two-stage mechanism, which computes only `ALL_HASH_VALUE` during measurements. In contrast, one-stage hashing requires the extraction and comparison of multiple individual hash values, resulting in higher I/O and computational overhead.

D. Effectiveness of Attack Detection Rate Optimization

This subsection evaluates the performance of Algorithm 2 under CPU usage constraints ranging from 1% to 9%. For each constraint, 100 experiments were conducted, and average detection rates are presented in Table II. The *Initial* row shows the detection rate before optimization, while the *Optimal* row indicates the rates after optimization. The improvement (Δ) highlights the algorithm's impact, with the highest gain observed under stricter CPU constraints (e.g., a 43.6% increase at 1% CPU). However, as CPU availability rises, marginal improvement diminishes to 0.9% at 9%. Beyond this threshold, the difference between initial and optimized measurements become negligible, both approaching 100%. These results are attributed to the increased availability of system resources, which enables shorter measurement intervals, minimizing the temporal window for exploitation. However, practical systems rarely allocate excessive resources solely for runtime integrity monitoring, requiring a trade-off between security and performance. Thus, 9% is selected as a representative constraint for resource-scarce scenarios, and can be adjusted accordingly.

TABLE II: The average detection rate under CPU constraints.

Constraints	1%	2%	3%	4%	5%	6%	7%	8%	9%
Initial (%)	23.1	43.7	62.5	77.2	86.1	90.6	93.9	97.2	99.0
Optimal (%)	66.7	73.4	78.2	82.9	91.7	93.5	96.7	98.9	99.9
Δ (%)	43.6	29.7	15.7	5.7	5.6	2.9	2.8	1.7	0.9



(a) The comparison of syscall latency in LMBench tests.

(b) The comparison of context switch latency in LMBench tests.

Fig. 3: Performance overheads of LMBench.

Since all monitored objects are assumed static during runtime, any detected modifications are considered genuine compromises, resulting in a false positive rate of `0`. Conversely, true negatives represent undetected attacks, quantified as `1-detection rate`.

E. Performance Evaluation

1) LMBench: The experiments evaluated common system call operations, including read, write, and open/close, as well as context switch latency under parallel configurations of 2, 4, 8, and 16 processes. A standardized workload size of 1024 KB was used and each test was repeated 100 times to compute average latency values. To ensure consistency and minimize variability, a 10-second warm-up phase was applied before each test to stabilize system resources, particularly cache usage. The system call latency results (Fig. 3a) show that LightRIM introduces an average overhead of 8.7% compared to enabling IMA alone. The highest impact is observed for open and close operations, with a maximum additional latency of 0.113 ms due to LightRIM's measurement process, which involves file operations. Despite this, the overall overhead is minimal and acceptable for practical deployment. The context switch latency results (Fig. 3b) indicate an average overhead of 2.3%, with the largest increase of 0.91 ms occurring under single-process configurations. This impact decreases as the number of parallel processes grows, confirming that LightRIM imposes negligible performance overhead, making it suitable for resource-constrained environments.

2) SPEC CPU2006: To evaluate the impact of LightRIM on compute-intensive applications, we conducted integer (INT) and floating-point (FP) tests on the SPEC CPU 2006 benchmark suite. Normalized CPU overhead was examined, revealing minimal impact in most applications. Fig. 4 illustrates the test results for the INT suite, indicating a mere 0.49% performance decline when the measurement module was en-

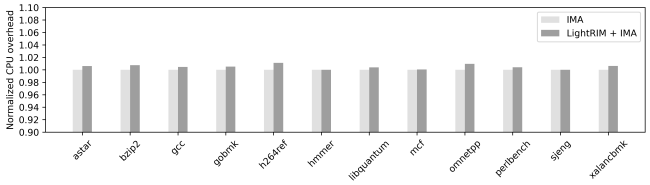


Fig. 4: Evaluation with SPEC CPU 2006 INT.

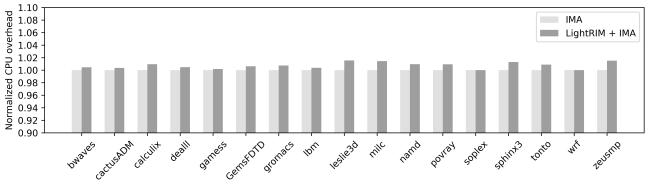


Fig. 5: Evaluation with SPEC CPU 2006 FP.

abled. Similarly, Fig. 5 presents the results for the FP suite, demonstrating only a 0.67% decrease in performance with the measurement module enabled.

F. Security Analysis

This subsection evaluates LightRIM's capability in detecting real-world attacks on immutable objects through two typical Linux integrity compromise scenarios.

1) Code Injection Attack: Attackers can compromise or control critical data by injecting malicious code, such as in VDSO hijacking attacks. These attacks involve hijacking the kernel's VDSO library to inject malicious code into processes. This allows attackers to access the process's memory, system, or network resources and potentially escalate privileges. Such attacks can evade standard security detection mechanisms as they operate within legitimate processes. Fortunately, we have discovered that monitoring the offset between VDSO functions and their starting addresses can effectively detect VDSO-based attacks. Any deviation from the baseline offset during measurement indicates potential tampering with the VDSO.

2) Rootkit Attack.: Rootkit technology manipulates kernel data structures to hide files, processes, network activity, and other critical system details. Diamorphine, a representative example, employs stealth syscall hooking by modifying the system call table, specifically `getdents` and `getdents64`, to conceal files and processes, and reuses the `kill` syscall for rootkit communication. Files with a specific prefix are hidden and processes receiving signal 31 are masked. The `diamorphine_init` module locates the syscall table address via brute force and integrates hooks, with `hacked_kill` enabling custom features. To counteract this threat, we established a baseline of syscall table addresses as a reference. By monitoring these values continuously at runtime, any unauthorized modifications to the syscall table can be promptly detected, providing effective protection against rootkit-based attacks.

V. Conclusion

LightRIM is a lightweight and adaptable integrity measurement framework tailored for resource-constrained embedded systems. This paper aims to protect unaltered critical objects during system runtime and promptly identify unauthorized modifications to maintain system security. Performance evaluations on LMBench and SPEC CPU2006 demonstrate its minimal overhead, highlighting its suitability for embedded applications. Future work will focus on deploying LightRIM in real-world embedded systems with known vulnerabilities to further assess its practicality and resilience.

References

[1] A. Apvrille, D. Gordon, S. E. Hallyn, M. Pourzandi, and V. Roy, "Digsig: Runtime authentication of binaries at kernel level." in *LISA*, vol. 4, 2004, pp. 59–66.

[2] C. Chang, X. Chen, S. Wang, Q. Xiao *et al.*, "Research on dynamic integrity measurement model based on memory paging mechanism," *Discrete Dynamics in Nature and Society*, vol. 2014, 2014.

[3] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a tcg-based integrity measurement architecture." in *USENIX Security symposium*, vol. 13, no. 2004, 2004, pp. 223–238.

[4] B. Stelte, R. Koch, and M. Ullmann, "Towards integrity measurement in virtualized environments—a hypervisor based sensory integrity measurement architecture (sima)," in *2010 IEEE International Conference on Technologies for Homeland Security (HST)*. IEEE, 2010, pp. 106–112.

[5] T. Jaeger, R. Sailer, and U. Shankar, "Prima: policy-reduced integrity measurement architecture," in *Proceedings of the eleventh ACM symposium on Access control models and technologies*, 2006, pp. 19–28.

[6] W. Zhang, P. Liu, and T. Jaeger, "Analyzing the overhead of file protection by linux security modules," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 393–406.

[7] A. Rein, "Drive: Dynamic runtime integrity verification and evaluation," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 728–742.

[8] A. Litchfield and W. Du, "Xfilter: An extension of the integrity measurement architecture based on fine-grained policies," *Applied Sciences*, vol. 13, no. 10, p. 6046, 2023.

[9] F. Bohling, T. Mueller, M. Eckel, and J. Lindemann, "Subverting linux'integrity measurement architecture," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–10.

[10] Y. Qin, J. Liu, S. Zhao, D. Feng, and W. Feng, "Ripte: runtime integrity protection based on trusted execution for iot device," *Security and Communication Networks*, vol. 2020, pp. 1–14, 2020.

[11] T. Abera, N. Asokan, L. Davi, F. Koushanfar, A. Paverd, A.-R. Sadeghi, and G. Tsudik, "Things, trouble, trust: on building trust in iot systems," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[12] R. Rashmi and A. Karthikeyan, "Secure boot of embedded applications-a review," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, 2018, pp. 291–298.

[13] N. Sklavos, I. D. Zaharakis, A. Kameas, and A. Kalapodi, "Security & trusted devices in the context of internet of things (iot)," in *2017 Euromicro Conference on Digital System Design (DSD)*. IEEE, 2017, pp. 502–509.

[14] J. Gu, Y. Ma, B. Zheng, and C. Weng, "Outlier: Enabling effective measurement of hypervisor code integrity with group detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 6, pp. 3686–3698, 2021.

[15] W. Tan, Y. Chen, Y. Li, Y. Liu, J. Wu, Y. Ding, and C. Zhang, "Ptstore: Lightweight architectural support for page table isolation," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[16] J. Criswell, N. Dautenhahn, and V. Adve, "Kcofi: Complete control-flow integrity for commodity operating system kernels," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 292–307.

[17] W. Tan, C. Li, Y. Chen, Y. Li, C. Zhang, and J. Wu, "Roload-pmp: Securing sensitive operations for kernels and bare-metal firmware," *IEEE Transactions on Computers*, 2024.

UPVSS: Jointly Managing Vector Similarity Search with Near-Memory Processing Systems

Chun-Chien Liu*, Chun-Feng Wu*, Yunho Jin†
*Department of Computer Science, National Yang Ming Chiao Tung University, Taiwan
†Department of Computer Science, Harvard University, USA
Corresponding Author: Chun-Feng Wu
E-mail: ccliu.cs12@nycu.edu.tw, cfwu417@cs.nycu.edu.tw, yjin@g.harvard.edu

***Abstract*—Vector similarity search plays a pivotal role in modern applications, including recommendation systems, image search, large language models (LLMs), and high-dimensional data retrieval. As data size scales, our research reveals that the search phase imposes substantial demands on DRAM bandwidth, leading to performance limitations in conventional von Neumann architecture with shared memory buses. This data movement bottleneck restricts the efficiency and scalability of vector similarity search due to insufficient memory bandwidth. To mitigate this issue, we leverage UPMEM, an off-the-shelf near-memory processing (NMP) system, to minimize the data movement between memory and compute units. However, UPMEM's computing engine has certain limitations and requires thorough application integration to unleash its high-parallelism capabilities. In this work, we introduce UPMEM-aware Vector Similarity Search (UPVSS), an architecture-aware system that jointly manages vector similarity search and UPMEM's NMP technology. UPVSS prioritizes offloading operations based on their strengths and capabilities, effectively alleviating the data movement bottleneck and improving overall system performance.**

***Index Terms*—Vector Similarity Search, LLM, Inverted File, FAISS, K-NN, PIM, UPMEM, HW/SW Codesign, Parallelism**

I. Introduction

Vector similarity search is a cornerstone of various applications, including large language models (LLMs) [8], [10], [26], recommendation systems [3], [33], image search [15], information retrieval [16], [24], and data retrieval in natural language processing. Instead of processing raw data like images or documents, modern systems represent them as high-dimensional vectors encoding features such as color or edges. Comparing these vectors enables efficient retrieval of similar data points. As systems and data volume scale, the number of vector dimensions has grown significantly, with models like OpenAI's GPT-3 producing embeddings with thousands to tens of thousands of dimensions [23], [34]. Our analysis indicates that running similarity searches on high-dimensional vectors is memory-bound, with cycles per instruction (CPI) values exceeding one due to frequent data movements between memory and CPU cache, which quickly saturate DRAM bandwidth. To address the immense memory bandwidth demand, this work integrates vector similarity search with a real near-memory processing system, UPMEM.

Exact similarity search requires brute-force distance calculations between all vectors, which is inefficient for large-scale, high-dimensional data [32]. To overcome this, modern applications use Approximate Nearest Neighbor (ANN) search, offering a balance between speed and accuracy. State-of-the-art ANN algorithms include cluster-based (e.g., Inverted File (IVF) [11]), graph-based (e.g., HNSW [20], NSG [7]), hash-based [38], and tree-based algorithms [27]. However, tree-based algorithms lack scalability for high-dimensional vectors [36], hash-based algorithms suffer from lower accuracy on large-scale datasets [19], and the memory consumption of graph-based algorithms grows rapidly with dataset size, making them unsuitable for large-scale applications [29]. Industrial vector similarity search libraries, like Meta's FAISS [6], [25], recommend using the IVF for managing large-scale high-dimensional vector datasets. Despite the use of ANN algorithms, the search phase remains limited by the memory bandwidth constraints of the von Neumann architecture, which struggles to meet the high DRAM bandwidth demands of vector similarity search.

Processing-in-memory (PIM) technologies are emerging as efficient solutions to the challenges of excessive data movement by bringing compute and memory units closer together. PIM can be broadly categorized into two types: processing-using-memory (PUM), where memory units perform computation, and near-memory processing (NMP), which integrates compute elements directly into memory modules. Emerging non-volatile memory technologies, such as ReRAM, enable analog-based PUM devices like ReRAM crossbars [14], [28] and TCAM [30], which are particularly effective for matrix multiplication and table look-up tasks. However, these devices rely heavily on Digital-to-Analog Converters (DAC) and Analog-to-Digital Converters (ADC), which together account for more than half of their total energy consumption and area costs [18], [39]. As a result, the significant cost associated with converting signals between analog and digital forms poses a major challenge to the widespread commercialization of analog-based PIM devices.

In contrast, UPMEM [4], [9], a recent NMP system, is the first commercially available off-the-shelf PIM device. UPMEM integrates DRAM Processing Units (DPUs) that combine general-purpose RISC processors with traditional 2D DRAM arrays on a single chip. Current UPMEM NMP systems support up to 2560 DPUs, each offering 1 GB/s of internal DRAM bandwidth, resulting in an impressive aggregate internal bandwidth of up to 2.56 TB/s. However, running applications on UPMEM naively can lead to challenges such

as frequent out-of-memory (OOM) errors due to its limited 64 MB of isolated memory per DPU and significant performance drops caused by high data transmission costs. Consequently, existing research has focused on adapting UPMEM for applications in analytics and databases [1], [13], [17], bioinformatics [2], [5], and data compression and decompression [21]. However, none of these works have explored its potential integration with vector similarity search.

Based on our investigations, running IVF-based vector similarity search directly on UPMEM encounters three significant challenges: (1) Each IVF cluster contains numerous high-dimensional vectors, often exceeding a DPU's available memory capacity (64 MB), leading to frequent out-of-memory (OOM) issues. (2) Migrating vectors between DPUs incurs substantial overhead due to costly data movement. (3) To fully utilize the DPU hardware pipeline, it is required to schedule at least 11 hardware threads (also known as tasklets) to execute concurrently. Motivated by the need to address these challenges and enable efficient vector similarity search on UPMEM, this paper introduces UPVSS, a UPMEM-aware Vector Similarity Search framework. UPVSS incorporates two key components. The first is the *DPU-aware Clusters Partition*, which evenly splits the vectors within each IVF cluster while accounting for the DPU's architectural constraints. The second is the *DPU-aware Vector Similarity Search Coordinator*, which strategically offloads vector distance calculations to DPUs and leverages task pipelining to fully exploit the high parallelism offered by the DPUs.

The rest of this paper is organized as follows: Section II presents the background, observation, and motivation. In Section III, the UPVSS is introduced to enhance query processing performance. Section IV provides the analysis and experimental results. Section V concludes this work.

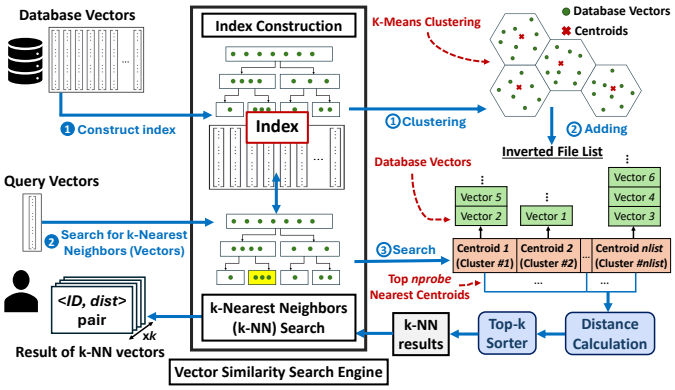


Fig. 1: The workflow of vector similarity search (left), along with the Inverted File (IVF) ANN index algorithm (right).

II. Background, Observation, and Motivation

A. Background

1) Vector Similarity Search

The typical workflow of vector similarity search consists of two steps, as illustrated in Fig. 1 (left). ❶ **Index Construction:** An index is constructed to facilitate efficient searching over the database vectors, enabling effective pruning of the search space during the query phase. This process is conceptually analogous to building traditional data structures such as B-trees or binary search trees, where the data is organized in a manner that allows the search algorithm to quickly eliminate regions unlikely contain the target items. Several indexing techniques have been developed for vector similarity search, such as KD-Tree, R-tree. ❷ **k-Nearest Neighbors (k-NN) Search:** Upon receiving a query vector, the system searches for the top-k similar vectors within the database vectors using a distance function, such as Euclidean distance, cosine similarity. This step involves extensive distance calculations and top-k sorting operations. The result of this step includes the identifiers of the k-NN in the database vectors along with their corresponding distances.

2) Approximate Nearest Neighbor (ANN) Index

Performing exact nearest neighbor search on high-dimensional vectors is computationally demanding and often impractical for large-scale applications [32]. Approximate Nearest Neighbor (ANN) methods address this challenge by reducing search time through a controlled approximation of k-NN results. This trade-off between accuracy and efficiency is suitable for many applications where near-optimal results are sufficient, such as LLMs and recommendation systems. Several ANN algorithms, including tree-based, hash-based, and graph-based methods, perform efficiently for low-dimensional applications but struggle with large-scale, high-dimensional vectors [19], [29], [36]. In contrast, the Inverted File (IVF[1]) index (or cluster-based ANN index) is commonly employed to handle large-scale, high-dimensional vectors effectively [11].

Fig. 1 (right) illustrates the workflow of the Inverted File (IVF) index. ① **Clustering Phase:** The database vectors are partitioned into *nlist* disjoint clusters by applying a clustering algorithm, typically k-means, to generate a list of centroids, where each centroid represents the center of a cluster. ② **Adding Phase:** Once the centroids are established, each database vector is assigned to its nearest centroid based on distance function and stored in the corresponding inverted file list associated with that centroid. This structure enables efficient search by restricting future queries to only the most relevant clusters. ③ **Search Phase:** The system begins by calculating the distances between the query vector and all centroids to identify the *nprobe* closest centroids, which represent the most relevant clusters for the query. Then, only the vectors within the top-*nprobe* closest centroid-associated clusters are scanned. For each vector in these clusters, the system performs distance calculations and employs a top-k sorting mechanism to determine the final k-NN results.

3) Off-The-Shelf NMP system: UPMEM

Near-memory processing (NMP) technology, which places computing units closer to memory, offers a promising solution to alleviate the data movement bottleneck across the shared bus between memory and compute units. The UPMEM NMP system is an off-the-shelf commercial solution, consisting of host CPUs, DRAM, and NMP memory modules (UPMEM DIMM modules), as shown in Fig. 2. The current system supports up to 20 UPMEM DIMMs, with each DIMM containing 128 DRAM processing units (DPUs), which are specialized

[1]IVF is included in industrial vector similarity search libraries, such as Meta FAISS.

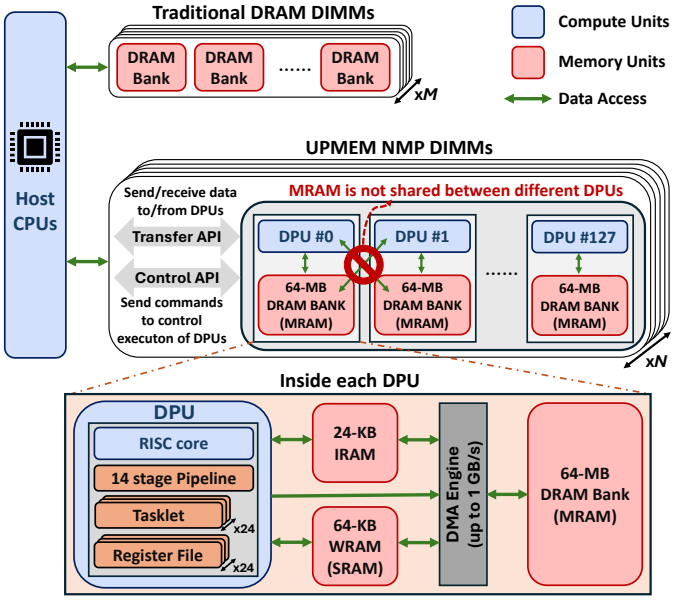


Fig. 2: The high-level architecture of UPMEM NMP system.

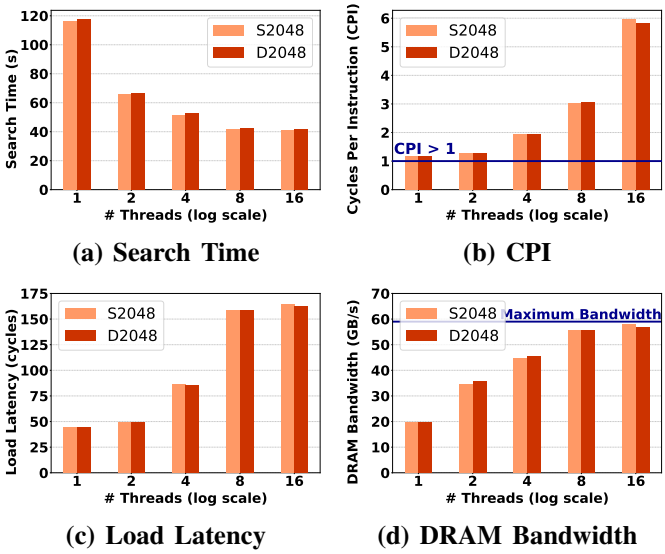


Fig. 3: Experimental evaluation of FAISS-IVF. The datasets and parameter settings are provided in Table I and Table II, respectively.

RISC processors embedded directly within standard DDR4 DRAM chips. Each DPU is connected solely to a dedicated 64 MB local DRAM bank, called MRAM, along with a 24 KB instruction memory (IRAM) and a 64 KB fast working memory (WRAM) serving as cache buffer. DPUs can only access their own MRAM, with no support for direct inter-DPU communication, thereby preventing data sharing between DPUs. Developers can implement customized WRAM memory management for their applications, optimizing resource utilization and improving overall performance. A DPU has 24 hardware threads, also known as tasklets, with each tasklet containing 24 32-bit registers. The DPU has a pipeline depth of 14 stages; however, due to hardware constraints, at least 11 tasklets must run concurrently to fully utilize the pipeline.

The UPMEM SDK [31] provides both control and transfer APIs, enabling the host CPUs to send and receive data to and from the MRAM of each DPU. This allows indirect communication between DPUs through host CPU intervention by copying data via the intermediate host DRAM. In addition, the host CPUs issue commands to the DPUs to control and manage their execution, enabling coordinated operation across multiple DPUs. Each DPU's internal DRAM bandwidth can reach up to 1 GB/s [4]. As a result, when all DPUs are active, the UPMEM NMP system achieves an aggregate internal DRAM bandwidth of up to 2.56 TB/s.

B. Observation & Motivation

Vector similarity search engines, such as Meta FAISS [6], [25], have been extensively optimized for traditional CPU architectures [22], [37]. However, these systems are fundamentally constrained by the von Neumann architecture, where CPU cores share memory and a common bus. When multiple threads perform concurrent k-Nearest Neighbors (kNN) searches, data movement between memory and CPU cache quickly saturates memory bandwidth, preventing linear performance scaling with added threads. Our analysis reveals that vector similarity search engines are memory-bound due to a low CPU cache reuse rate and low computational intensity. Most database vectors in the CPU cache are used for a single distance calculation and then flushed, limiting reuse. Additionally, each distance calculation involves few instructions yet requires moving large vectors, which fails to amortize the cost of data transfer. This issue is particularly evident in modern implementations like FAISS, which employs Intel Advanced Vector Extensions 2.0 (AVX2[2]) to process multiple data simultaneously per instruction.

We conduct a series of observational experiments, as shown in Fig. 3, to demonstrate that FAISS-IVF performance saturates as additional threads are added and to analyze the underlying causes of this saturation. We run Meta's FAISS-IVF on a system featuring an Intel i7-12700 processor and 128 GB of DDR5 DRAM, providing a theoretical peak DRAM bandwidth of 59 GB/s. We use Intel Vtune to profile the hardware usage while running FAISS-IVF. As shown in Fig. 3(a), the total search times for running FAISS-IVF over S2048 and D2048 datasets scale with the number of CPU threads, with the x-axis representing thread count and the y-axis indicating total vector search time. The results reveal that FAISS-IVF achieves limited performance gains and poor scalability as thread count increases, with overall search time plateauing beyond 8 threads.

Fig. 3(b) illustrates the cycles per instruction (CPI), with the x-axis representing the number of CPU threads and the y-axis denoting the CPI. The results indicate that CPI increases with the number of CPU threads, exceeding 1 for both datasets, which suggests the system is likely memory-bound. This is notable given that modern superscalar processors can issue up to four instructions per cycle. To investigate the high CPI,

[2]Intel AVX2 is an x86 instruction set extension enabling 256-bit wide processing for integer operations.

Fig. 3(c) shows that average load latency also rises with more CPU threads, likely due to congestion in the shared DRAM bus. This congestion causes load instructions to queue, increasing latency. Additionally, Fig. 3(d) reveals that DRAM bandwidth utilization saturates as thread count grows. By the time thread count reaches 8, DRAM utilization approaches its theoretical peak, confirming that limited performance gains stem from the shared memory bus failing to meet increased memory bandwidth demands. This challenge intensifies with high-dimensional vectors, as limited cache space can store fewer vectors. These observations highlight the need for a near-memory processing (NMP) system to address the data movement bottlenecks encountered in vector similarity search.

III. UPVSS: UPMEM-aware Vector Similarity Search

A. UPVSS System Architecture Design Overview

The primary constraints of UPMEM DPUs are *(1) each DPU's MRAM is limited to 64 MB, making efficient data placement a non-trivial task; (2) MRAM is not shared across DPUs, resulting in costly inter-DPU communication; and (3) to fully utilize the DPU hardware pipeline, at least 11 hardware threads (tasklets) must run concurrently.* This section introduces our UPMEM-aware Vector Similarity Search (UPVSS) system, designed to tightly integrate vector similarity search with UPMEM DPUs to address the data movement bottleneck encountered in von Neumann architecture. In addition, it aims to overcome the constraints of the DPU architecture and fully exploit its capabilities, particularly its high parallelism and substantial aggregate internal DRAM bandwidth, which can reach up to 2.56 TB/s.

Our UPVSS comprises two key components. First, the *DPU-aware Clusters Partition* evenly splits the vectors belonging to each IVF cluster, with considering the DPU architecture. This enables future k-NN search operations without requiring synchronization across DPUs and ensures that all DPUs are actively utilized, leveraging both the high parallelism and substantial aggregate internal DRAM bandwidth. Second, the *DPU-aware Vector Similarity Search Coordinator* significantly reduces data movement across the shared bus between host CPU and host memory during the search phase by offloading distance calculations to DPUs. Additionally, it facilitates collaboration between vector similarity search and DPUs and incorporates customized WRAM memory management to further optimize performance, considering running multiple tasklets to fully utilize the DPU hardware pipeline.

B. DPU-aware Clusters Partition

The key to implementing UPMEM-aware Vector Similarity Search lies in effectively distributing the vectors of each cluster across the DPUs. Upon receiving a query vector, it is then broadcast to each DPU that holds a portion of database vectors, allowing the DPUs to collaboratively work with the host CPUs to determine the final k-nearest neighbors (k-NN) results. To realize this, we propose *DPU-aware Clusters Partition*, which ensures a balanced distribution of clusters across all DPUs to maximize parallelism and resource utilization.

As shown in Fig. 4, our system begins with IVF index construction on the host side, where the host organizes the

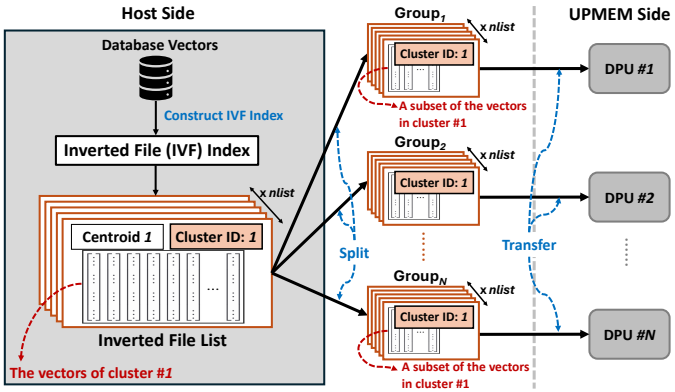


Fig. 4: DPU-aware Clusters Partition for UPVSS system.

database vectors into multiple clusters, each with an associated centroid. These database vectors are then inserted into the corresponding inverted file list of that centroid within the IVF index structure. To ensure all DPUs are activated during k-NN searches, we then evenly partition the vectors of each cluster into N groups. More precisely, the vectors within each centroid-associated cluster in the IVF index are divided into N equal segments, ensuring that each group receives a subset of every cluster. After partitioning, each of these N groups is then transferred to N distinct DPUs within the UPMEM NMP system, where N represents the maximum operational DPUs. With this partition strategy, all DPUs are actively utilized during the search phase, as each DPU contains a subset of every cluster.

C. DPU-aware Vector Similarity Search Coordinator

Fig. 5 provides a comprehensive overview of the DPU-aware Vector Similarity Search Coordinator, which encompasses the joint management and coordination between the host CPUs and DPUs. Given that the search phase primarily involves extensive distance calculations with low computational intensity, it cannot amortize the cost of data movement. Therefore, the core concept of DPU-aware Vector Similarity Search Coordinator is to offload these distance calculations between the query vector and database vectors to the DPUs. The host then collects $\langle ID, dist \rangle$ key-value pairs returned by all DPUs and merges them to obtain the final k-NN results. This approach significantly reduces data movement across the shared bus, thereby enhancing the CPU cache utilization, as each $\langle ID, dist \rangle$ pair is considerably smaller than each database vector. Furthermore, the distance calculations for vectors on each DPU are independent, eliminating the need for inter-DPU synchronization and thus effectively leveraging the high parallelism of the UPMEM system.

The workflow of the DPU-aware Vector Similarity Search Coordinator on the host side is as follows: ❶ The host receives the query vector and identifies the *nprobe* closest centroids, determining the specific *nprobe* clusters for the DPUs to perform distance calculations. The IDs of these clusters are then combined with the query vector to form the Query Metadata. ❷ The Query Metadata is broadcast to all DPUs. ❸ The host sends commands to launch each DPU, prompting them to execute distance calculations for the specified clusters

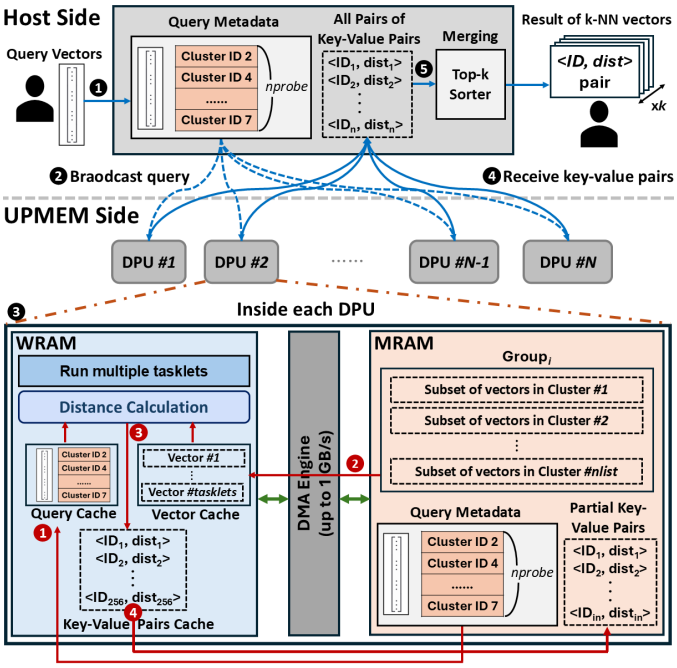


Fig. 5: The workflow of DPU-aware Vector Similarity Search Coordinator.

based on the cluster IDs in the Query Metadata. ❹ Upon completing the distance calculations, each DPU generates a set of $\langle ID, dist \rangle$ pairs as partial results, which are then sent back to the host CPU for further processing. ❺ Finally, the host merges all $\langle ID, dist \rangle$ from each DPU to determine the final k-NN results. Note that steps ❶ – ❺ are executed iteratively to continuously process incoming queries.

To further optimize performance and facilitate the efficient and concurrent execution of multiple tasklets, the workflow of the DPU-aware Vector Similarity Search Coordinator on the DPU side is organized as follows: ❶ We design a dedicated Query Cache in WRAM to store the Query Metadata, allowing it to be shared among all tasklets within the same DPU and ensuring rapid access. ❷ The subset of vectors assigned to each DPU is then evenly distributed across all active tasklets. Each tasklet has its own 2 KB Vector Cache in WRAM, allowing it to independently hold and process its assigned portion of the vectors. ❸ Subsequently, each tasklet performs distance calculations on its designated vectors, with the resulting $\langle ID, dist \rangle$ pairs stored in a shared Key-Value Pairs Cache accessible to all tasklets. ❹ The Key-Value Pairs Cache is configured with a fixed size of 2 KB, in accordance with the DPU architecture's limitation of a maximum 2 KB per WRAM-to-MRAM transfer [31]. Moreover, prior research indicates that this transfer size minimizes data movement overhead [9]. With this configuration, the cache can store up to 256 $\langle ID, dist \rangle$ pairs in WRAM. Once the Key-Value Pairs Cache reaches capacity, a designated tasklet is responsible for managing the transfer and writing the cached result back to persistent key-value storage in MRAM. The workflow and WRAM memory management strategy effectively optimize WRAM cache buffer utilization and enable concurrent multi-tasklet execution with minimal synchronization overhead, enhancing parallelism at the software level and fully utilizing the DPU pipeline at the hardware level.

IV. Experiment

A. Experiment Setup

This section evaluates the effectiveness of the proposed UPVSS system for improving the query processing performance. We compare the performance of the UPVSS with the state-of-the-art FAISS library, which is optimized using multi-threading with OpenMP, AVX2 SIMD vector instructions, and the BLAS (Basic Linear Algebra Subprograms) library to accelerate linear algebra computations. Both UPVSS and FAISS are evaluated using the IVF ANN indexing algorithm on the same system.

The experiments were conducted on a commercially available UPMEM near-memory processing (NMP) server running a Linux-based environment. The server is powered by two Intel Xeon Silver 4216 CPUs and equipped with 256 GB of DRAM, offering a theoretical peak bandwidth of 32 GB/s. Additionally, the system is equipped with 64 GB of UPMEM NMP memory distributed across 8 UPMEM DIMM modules, incorporating a total of 1024 DRAM Processing Units (DPUs). For baseline comparisons, the FAISS library was configured to run with 64 CPU threads.

The datasets used to evaluate the performance of the proposed UPVSS against FAISS are summarized in Table I. These datasets are synthesized based on the distributions of real-world SIFT [11], [12] and DEEP [35] datasets to generate high-dimensional vectors with dimensions of 2048, 3072, and 4096, named S2048, S3072, S4096, D2048, D3072, and D4096. Each dataset comprises 1 million (1M) database vectors and 1000 query vectors, with all vectors quantized to 8-bit integer format (uint8_t). Table II provides the default parameter settings used in our experiments along with a description of each parameter. Unless otherwise specified, all experiments were conducted using these default settings.

Dataset	# Dims	# Vectors	Dataset	# Dims	# Vectors
S2048	2,048	1,000,000	D2048	2,048	1,000,000
S3072	3,072	1,000,000	D3072	3,072	1,000,000
S4096	4,096	1,000,000	D4096	4,096	1,000,000

TABLE I: Datasets.

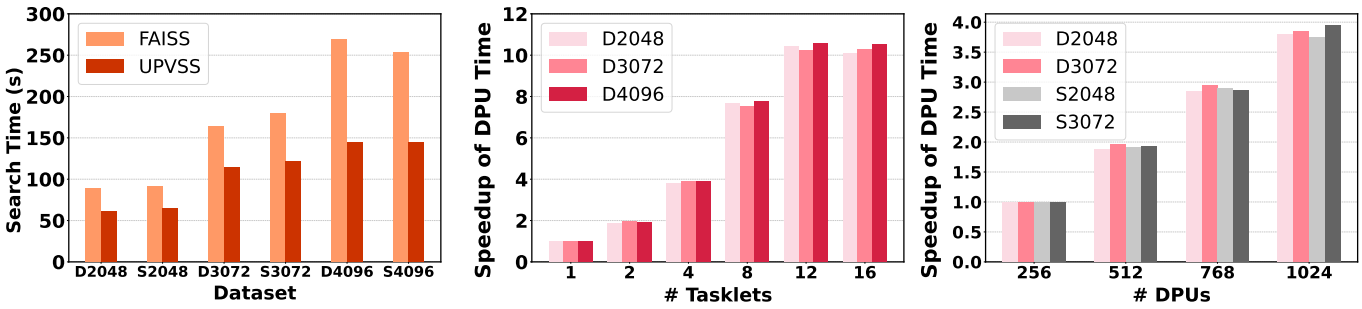
Parameters	Default value & Explanation
k	Top-k closest vectors when performing vector similarity search. **Default value: 100**
nlist	The number of clusters used to partition the database vectors. **Default value: 250**
nprobe	The number of nearest clusters explored during the k-NN search phase. **Default value: 20**

TABLE II: Parameters setting.

B. Experiment Results

1) Performance Results over State-Of-The-Art

Fig. 6(a) presents the total search time of the proposed UPVSS compared with the state-of-the-art FAISS, both running the IVF indexing algorithm across six datasets. The x-axis represents the six different datasets, while the y-axis indicates



(a) Search time comparison with FAISS. (b) Impact of tasklets on DPU time. (c) Scalability with increasing DPUs.

Fig. 6: Performance and scalability evaluation of UPVSS.

the total search time for UPVSS and FAISS on each dataset. The results show that UPVSS can reduce the overall search time by 30% to 46% and achieves a speedup of 1.42x to 1.86x compared to FAISS across all datasets.

To validate the efficiency of our WRAM memory management in DPU-aware Vector Similarity Search Coordinator under multi-tasklet execution, Fig. 6(b) presents the speedup of DPU execution time with respect to the number of tasklets per DPU across three datasets: D2048, D3072, and D4096. The x-axis represents the number of tasklets, while the y-axis denotes the speedup of DPU execution time relative to a single tasklet. The results show that the speedup of UPVSS scales linearly with the number of tasklets and saturates at 12 tasklets, consistent with the hardware constraints of the DPU. This indicates that UPVSS effectively maximizes the utilization of the DPU hardware pipeline.

2) Scalability of UPVSS

Fig. 6(c) shows the scalabiltiy of UPVSS with respect to the number of active DPUs. The x-axis represents the number of DPUs activated in the system, while the y-axis denotes the speedup of DPU execution time relative to 256 DPUs. The results show that the speedup of UPVSS scales linearly with the number of active DPUs. This scalability can be attributed to the DPU-aware Clusters Partition, which balances each cluster across DPUs, and the DPU-aware Vector Similarity Search, which offloads distance calculations to DPUs, significantly reducing data movement across the shared bus. Unlike conventional CPU architectures, UPVSS can further enhance performance by increasing the number of DPUs.

3) The impact of nlist

Fig. 7 shows the performance evaluations across six datasets for varying *nlist* values. The x-axis represents different *nlist* values, while the y-axis denotes the normalized speedup in performance relative to FAISS. The results indicate that UPVSS achieves a speedup of 1.42x to 2.62x compared to FAISS across all datasets, with an average speedup of 1.95x. Notably, as *nlist* value decreases, the number of vectors assigned to each cluster increases. This leads to more distance calculations per query, futher amplifying the memory-bound nature of FAISS. Consequently, UPVSS achieves higher speedups compared to FAISS.

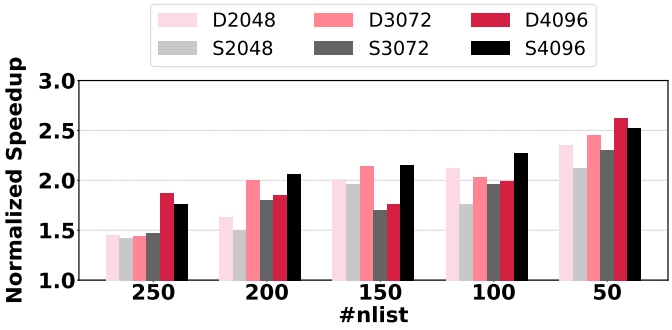


Fig. 7: Normalized speedup of UPVSS compared to FAISS for different *nlist* values.

V. Conclusion

Aiming to address the data movement bottleneck encountered in vector similarity search due to low computational intensity of distance calculations and substantial data movement across the shared bus, this paper proposes the UPMEM-aware Vector Similarity Search (UPVSS), an architecture-aware system that jointly manages vector similarity search and UPMEM's near-memory processing (NMP) technology to tackle this challenges. UPVSS comprises two key components: (1) *DPU-aware Clusters Partition*, which evenly splits vectors while accounting for the DPU's architectural constraints, unlocking its high parallelism and substantial aggregate internal DRAM bandwidth; and (2) *DPU-aware Vector Similarity Search Coordinator*, which significantly reduces data movement across the shared bus by offloading distance calculations to DPUs and incorporates customized WRAM memory management within the DPUs to fully utilize the DPU hardware resources. Evaluation results show that UPVSS achieves an average speedup of 1.95x compared to the state-of-the-art FAISS.

VI. Acknowledgement

We would like to thank Prof. David Brooks and Prof. Gu-Yeon Wei from Harvard University for their thoughtful comments and insightful suggestions. This work was supported in part by the National Science and Technology Council under grant nos. 113-2628-E-A49-021, 113-2640-E-A49-012 and Ministry of Education under Yushan Young Fellow Program.

REFERENCES

[1] A. Baumstark, M. A. Jibril, and K.-U. Sattler. Adaptive query compilation with processing-in-memory. In *2023 IEEE 39th International Conference on Data Engineering Workshops (ICDEW)*, pages 191–197, 2023.

[2] L.-C. Chen, C.-C. Ho, and Y.-H. Chang. Uppipe: A novel pipeline management on in-memory processors for rna-seq quantification. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

[3] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 271–280, New York, NY, USA, 2007. Association for Computing Machinery.

[4] F. Devaux. The true processing in memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)*, pages 1–24, 2019.

[5] S. Diab, A. Nassereldine, M. Alser, J. Gómez Luna, O. Mutlu, and I. El Hajj. A framework for high-throughput sequence alignment using real processing-in-memory systems. *Bioinformatics*, 39(5):btad155, 03 2023.

[6] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou. The faiss library. *arXiv preprint arXiv:2401.08281*, 2024.

[7] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, Jan. 2019.

[8] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M.-W. Chang. Realm: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[9] J. Gómez-Luna, I. E. Hajj, I. Fernandez, C. Giannoula, G. F. Oliveira, and O. Mutlu. Benchmarking a new paradigm: Experimental analysis and characterization of a real processing-in-memory system. *IEEE Access*, 10:52565–52608, 2022.

[10] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei. s^3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

[11] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011.

[12] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 861–864, 2011.

[13] H. Kang, Y. Zhao, G. E. Blelloch, L. Dhulipala, Y. Gu, C. McGuffey, and P. B. Gibbons. Pim-tree: A skew-resistant index for processing-in-memory. *Proc. VLDB Endow.*, 16(4):946–958, Dec. 2022.

[14] Y.-W. Kang, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, and S.-Y. Ho. On minimizing analog variation errors to resolve the scalability issue of reram-based crossbar accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11), 2020.

[15] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *2009 IEEE 12th International Conference on Computer Vision*, pages 2130–2137, 2009.

[16] P. P.-H. Kung, Z. Fan, T. Zhao, Y. Liu, Z. Lai, J. Shi, Y. Wu, J. Yu, N. Shah, and G. Venkataraman. Improving embedding-based retrieval in friend recommendation with ann query expansion. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2930–2934, New York, NY, USA, 2024. Association for Computing Machinery.

[17] C. Lim, S. Lee, J. Choi, J. Lee, S. Park, H. Kim, J. Lee, and Y. Kim. Design and analysis of a processing-in-dimm join algorithm: A case study with upmem dimms. *Proc. ACM Manag. Data*, 1(2), June 2023.

[18] T.-S. Lo, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, and W.-C. Wang. Space-efficient graph data placement to save energy of reram crossbar. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2021.

[19] K. Lu, H. Wang, W. Wang, and M. Kudo. Vhp: approximate nearest neighbor search via virtual hypersphere partitioning. *Proc. VLDB Endow.*, 13(9):1443–1455, May 2020.

[20] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2020.

[21] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, and A. Fedorova. A case study of Processing-in-Memory in off-the-Shelf systems. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 117–130. USENIX Association, July 2021.

[22] H. Ootomo, A. Naruse, C. Nolet, R. Wang, T. Feher, and Y. Wang. Cagra: Highly parallel graph construction and approximate nearest neighbor search for gpus. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 4236–4247, 2024.

[23] OpenAI. GPT-3 Embedding. https://platform.openai.com/docs/guides/embeddings, 2024.

[24] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[25] M. A. Research. FAISS: A library for efficient similarity search and clustering of dense vectors. https://github.com/facebookresearch/faiss, 2024.

[26] T. Shen, G. Long, X. Geng, C. Tao, Y. Lei, T. Zhou, M. Blumenstein, and D. Jiang. Retrieval-augmented retrieval: Large language models are strong zero-shot retriever. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15933–15946, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics.

[27] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[28] L. Song, X. Qian, H. Li, and Y. Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE international symposium on high performance computer architecture (HPCA)*, pages 541–552. IEEE, 2017.

[29] S. J. Subramanya, Devvrit, R. Kadekodi, R. Krishaswamy, and H. V. Simhadri. *DiskANN: fast accurate billion-point nearest neighbor search on a single node*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[30] C.-L. Tsai, C.-F. Wu, Y.-H. Chang, H.-W. Hu, Y.-C. Lee, H.-P. Li, and T.-W. Kuo. A digital 3d tcam accelerator for the inference phase of random forest. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[31] UPMEM. UPMEM SDK 2024.2.0 Documentation. https://sdk.upmem.com/2024.2.0/, 2024.

[32] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, page 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[33] C.-F. Wu, C.-J. Wu, G.-Y. Wei, and D. Brooks. A joint management middleware to improve training performance of deep recommendation systems with ssds. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 157–162, 2022.

[34] Y. Xu, H. Liang, J. Li, S. Xu, Q. Chen, Q. Zhang, C. Li, Z. Yang, F. Yang, Y. Yang, P. Cheng, and M. Yang. Spfresh: Incremental in-place update for billion-scale vector search. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 545–561, New York, NY, USA, 2023. Association for Computing Machinery.

[35] A. B. Yandex and V. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2055–2063, 2016.

[36] W. Yang, T. Li, G. Fang, and H. Wei. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2241–2253, New York, NY, USA, 2020. Association for Computing Machinery.

[37] B. Zheng, Z. Yue, Q. Hu, X. Yi, X. Luan, C. Xie, X. Zhou, and C. S. Jensen. Learned probing cardinality estimation for high-dimensional approximate nn search. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 3209–3221, 2023.

[38] B. Zheng, X. Zhao, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen. Pm-lsh: A fast and accurate lsh framework for high-dimensional approximate nn search. *Proc. VLDB Endow.*, 13(5):643–655, Jan. 2020.

[39] Q. Zheng, Z. Wang, Z. Feng, B. Yan, Y. Cai, R. Huang, Y. Chen, C.-L. Yang, and H. H. Li. Lattice: An adc/dac-less reram-based processing-in-memory architecture for accelerating deep convolution neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

Guarder: A Stable and Lightweight Reconfigurable RRAM-based PIM Accelerator for DNN IP Protection

Ning Lin[1,2,3,*], Yi Li[1,*], Jiankun Li[1,*], Jichang Yang[1,*], Yangu He[1], Yukui Luo[5], Dashan Shang[2], Xiaoming Chen[4,†], Xiaojuan Qi[1,†], Zhongrui Wang[3,†]

[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China; [2]State Key Lab of Fabrication Technologies for Integrated Circuits and Key Laboratory of Microelectronic Devices and Integrated Technology, Institute of Microelectronics of the Chinese Academy of Sciences (CAS), Beijing, China; [3]School of Microelectronics, Southern University of Science and Technology, Shenzhen, China; [4]Institute of Computing Technology of CAS, Beijing, China; [5]Binghamton University, State University of New York, Binghamton, NY, USA.

[*]Equal contribution to this work. [†]Corresponding author: wangzr@sustech.edu.cn;chenxiaoming@ict.ac.cn;xjqi@eee.hku.hk.

***Abstract*—Deploying deep neural networks (DNNs) on conventional digital edge devices faces significant challenges due to high energy consumption. A promising solution is the processing-in-memory (PIM) architecture with resistive random-access memory (RRAM), but RRAM-based systems suffer from imprecise weights due to programming stochasticity and cannot effectively utilize conventional weight encryption/decryption intellectual property (IP) protection schemes. To address these issues, we propose a novel software-hardware co-design Guarder. On the hardware side, we introduce 3T2R cells to achieve reliable multiply-accumulate (MAC) operations and use reconfigurable inverter operating voltages to encode keys for encrypting DNNs on RRAM. On the software side, we implement a contrastive training method that ensures high model accuracy on authorized chips while degrading performance on unauthorized ones. This approach protects DNN IP with minimal hardware overhead while significantly mitigating the effects of RRAM programming stochasticity. Extensive experiments on tasks such as image classification (using MLP, ResNet, and ViT), segmentation (using SegFormer), and image generation (using DiT) validate the effectiveness of our method. The proposed contrastive training ensures negligible performance degradation on authorized chips, while performance on unauthorized chips drops to random guessing or generation. Compared to traditional RRAM accelerators, the 3T2R-based accelerator achieves a 1.41× reduction in area overhead and a 2.28× reduction in energy consumption.**

***Index Terms*—IP Protection, PIM, Reconfigration, Contrastive training.**

I. Introduction

Deep neural network (DNN) models, particularly vision foundation models such as Vision Transformers (ViTs) [1] and generative models such as diffusion models [2], are widely deployed in edge applications like personal healthcare [3], autonomous driving and safety [4], [5], and entertainment [6]. With increasing concerns about data privacy and security [7], edge deployment of these models offers significant benefits by protecting user privacy, eliminating data communication between devices and the cloud, and enabling offline services. However, implementing these advanced DNNs on traditional von Neumann architecture edge devices is challenging due to excessive energy consumption and latency.

A promising solution to this challenge is the use of processing-in-memory (PIM) architectures with non-volatile memories (NVMs), such as RRAM. By integrating computation, memory, and storage within a single physical unit, RRAM-based PIM significantly reduces the energy overhead associated with data movement and improves parallelism. Despite its advantages, RRAM-based PIM faces two critical challenges: (1) substantial stochasticity in RRAM programming leads to imprecise weights [8], and (2) persistent data in RRAM devices makes them vulnerable to security breaches. Attackers can physically probe the memory to extract DNN weights (Fig. 1) [9], [10], and traditional encryption methods [11], [12] are difficult to implement due to both PIM architecture and programming stochasticity of RRAM.

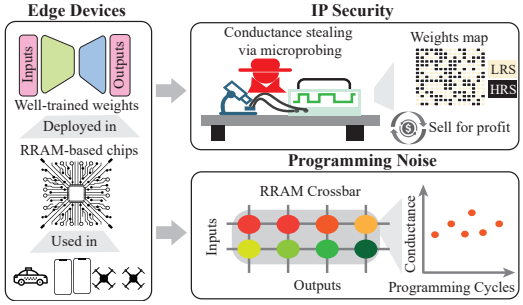


Fig. 1. Two main challenges of using RRAM-based PIM for DNN acceleration on edge devices. (1) Non-volatility makes model parameters prone to theft; (2) Programming noise can impact computation results.

To address these issues, we propose a novel software-hardware co-design framework. Our approach introduces a 3T2R PIM array with reconfigurable inverter operating voltage on the hardware side, and a contrastive training method on the software side. Together, these innovations enhance computational stability and security while creating a significant performance gap between authorized and unauthorized chips. Specifically, the contributions of this work include:

- **A secure and robust IP protection method:** We propose a software-hardware co-design Guarder that protects DNN IP on RRAM with minimal encryption overhead, while also effectively mitigating the impact of RRAM programming stochasticity.
- **Hardware innovations:** We introduce 3T2R cells to construct crossbar arrays. The inverters in the 3T2R cells eliminate weight imprecision due to programming stochasticity, and the reconfigurable inverter operating

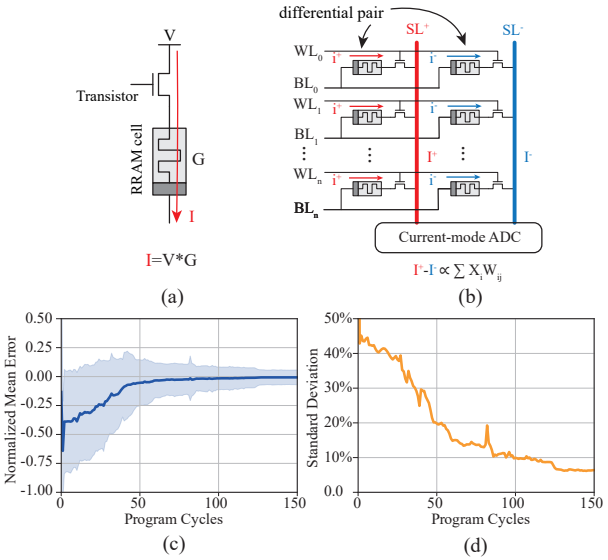


Fig. 2. RRAM-based PIM MAC principle (a) (b) and RRAM programming stochasticity (c) (d).

voltages serve as keys to encrypt weights, preventing attackers from exploiting the deployed weights.

- **Software advancements:** We develop a contrastive training method that ensures high model accuracy on authorized chips while degrading performance on unauthorized chips (e.g., general-purpose GPUs, CPUs) to near-random guessing levels.
- **Extensive validation:** We validated our approach on tasks like image classification, segmentation, and generation. Results show stable performance on authorized chips and significant degradation on unauthorized ones, ensuring effective model protection. The 3T2R-based design reduces chip area by 1.41× and energy consumption by 2.28× compared to previous RRAM designs, and withstands up to 30% RRAM programming stochasticity.

II. Preliminaries

A. RRAM-based DNN Accelerators

Computational Principle. RRAM is a promising non-volatile PIM technology, offering high integration density, low energy consumption, and reduced fabrication costs [13]–[16]. As shown in Fig. 2 (a), a single RRAM device functions as a variable resistor. When paired with a transistor, it forms the classic 1T1R structure [17], generating output current based on Ohm's law. Fig. 2 (b) illustrates that arranging these 1T1R cells in a 2D crossbar structure allows the currents from different 1T1R cells on the same source line (SL) to accumulate according to Kirchhoff's current law, or weighted-current summation (WCS). A differential pair formed by two adjacent SL vectors of 1T1R units, when processed through an ADC [18], can represent both positive and negative values, corresponding to the weights in a DNN and their MAC operations with the input. However, WCS has inherent limitations: (1) Current sensing leads to high static power consumption. (2) The accumulated MAC result reflects individual RRAM programming errors [19]–[21]. (3) It is incompatible with traditional weight storage encryption techniques [11], [12].

RRAM Device Characteristics. RRAM cells exhibit significant device-to-device and cycle-to-cycle programming stochasticity due to the underlying electrochemical reactions. Typically, a large number of programming cycles are required to approach the target resistance. Specifically, we programmed 250 real RRAM devices fabricated under 180 nm technology and analyzed the normalized mean error ($(G_p$ - $G_t)/G_t$, where G_p is the programmed conductance and G_t is the target conductance) and the standard deviation of the programming errors, as shown in Fig. 2 (c) and (d). The results indicate that when the programming cycles are fewer than 50, the mean error and standard deviation are relatively large, exceeding -0.2 and 20%, respectively. As the number of programming cycles increases, the programming error gradually decreases. This yields large time and energy overheads.

B. IP Protection on RRAM

Prior works addressing IP protection on RRAM primarily employ encryption methods and circuit obfuscation, yet these approaches seldom consider the stochastic nature of RRAM.

Encryption. Cai *et al.* [11], [12] proposed a lightweight method known as sparse fast gradient encryption. This technique involves adding perturbations to the conductance values of the RRAM during encryption, which can be removed during decryption. However, it overlooks the inherent stochasticity of RRAM. Due to programming stochasticity, RRAMs often require multiple programming cycles to accurately write a weight, the energy consumption and latency adversely affect the DNN inference.

Circuit obfuscation. Zou *et al.* [22] proposed obfuscating the connections between positive and negative crossbars. During inference, the obfuscation module must be configured with the correct keys; without these keys, the stolen DNNs becomes non-functional. This approach, however, requires complex peripheral circuits, resulting in substantial area and energy overhead, which is particularly problematic for edge devices. Furthermore, this method does not address the stochasticity of RRAM.

III. Guarder Framework

A. Threat Model

In our threat model, we focus on real-time security-sensitive applications, such as segmentation in self-driving systems or drones, and generative models on mobile devices. We consider the victim to be the model provider and hardware provider (e.g., Nvidia's Jetson NANO) [23]. The attacker is assumed to be the client user operating on the edge side. The model vendor must protect their valuable model IP before deploying the model on edge devices.

Attacker's Capabilities. We assume that the attacker aims to extract a well-trained DNN model. With physical access to RRAM-based PIM accelerators, the attacker can read memory to extract DNN weights and deduce the architecture [9], [24],

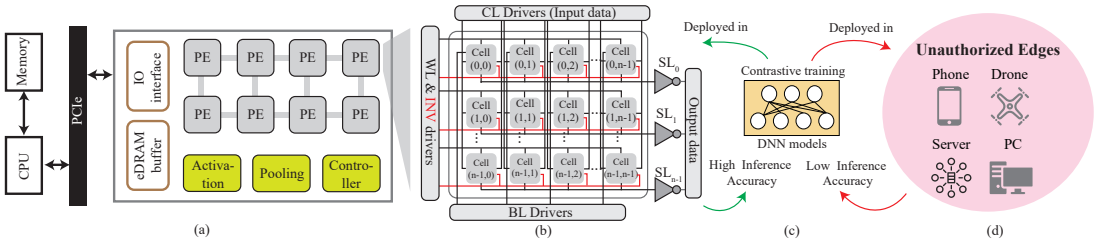
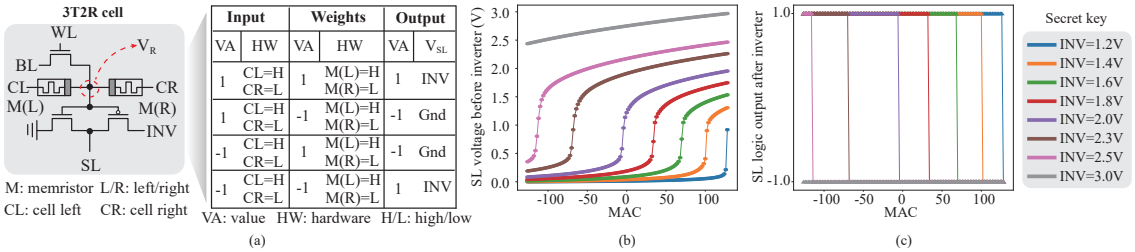


Fig. 3. Hardware-software co-design overview. (a) Hardware architecture; (b) Crossbar array; (c) Contrastive training; (d) Unauthorized edges.



Input		Weights		Output	
VA	HW	VA	HW	VA	V_{SL}
1	CL=H CR=L	1	M(L)=H M(R)=L	1	INV
1	CL=H CR=L	-1	M(L)=H M(R)=L	-1	Gnd
-1	CL=H CR=L	1	M(L)=H M(R)=L	-1	Gnd
-1	CL=H CR=L	-1	M(L)=H M(R)=L	1	INV

Fig. 4. (a) Noise-robust 3T2R cell design; (b-c) SL voltages before inverter and SL logic output after inverter as a function of MAC result at reconfigurable INV voltages, respectively.

[25]. Such extracted information could be exploited for profit, for instance, by reselling the stolen model.

Attacker's Limitations. Following existing literature (e.g., [26], [27]), we assume that attackers lack access to the vendor's proprietary training data, which is crucial for retraining the stolen DNN parameters. This training data is typically private, safeguarded by data protection laws such as the EU GDPR [28], and holds significant commercial value, making it inaccessible to the attacker.

B. Overview

To address the stochastic nature of RRAM and the challenges of IP protection, we propose a novel hardware-software co-design, as illustrated in Fig. 3. On the hardware side, the 3T2R design has been experimentally validated to withstand up to 30% of the RRAM programming stochasticity. Furthermore, the reconfigurable array effectively prevents attackers from utilizing the functions of the weights deployed on the RRAM after they have been stolen. On the algorithm side, we propose a contrastive training method, which ensures that the model achieves high accuracy and performance on authorized chips, while the accuracy on unauthorized edge chips is reduced to near-random guess levels.

C. Hardware Design

The RRAM-based PIM architecture is depicted in Fig. 3 (a). The top level of the chip contains Processing Elements (PEs), an eDRAM buffer, and neural-functional peripheries, including activation functions and pooling, and controller. Different layers of DNNs are typically mapped to different PE units. Additionally, when the length of the unrolled weights in the same layer exceeds the size of the crossbar, the weights are split and mapped onto different PEs.

Noise-Robust 3T2R Cell. Inspired by the voltage division cell design in [19]–[21], we propose an innovative 3T2R cell (Fig. 4 (a)). This cell integrates two RRAMs in complementary states, an inverter, and a programming transistor. The output of the RRAM voltage divider (referred to as V_R in Fig. 4 (a)) is binarized by the inverter, effectively performing an AND operation between the input (referred to as CL/CR in Fig. 4 (a)) and the stored conductance. The binary inverter within the 3T2R cell also mitigates RRAM programming stochasticity impact on the cell output.

More specifically, the truth table illustrates the relationship between different inputs and weights and their corresponding outputs. For instance, when the input value is logic '1', the CL voltage is high, while the CR voltage is low. When the weight is logic '1', the corresponding left RRAM (M(L)) is in a high resistance state, and the right RRAM (M(R)) is in a low resistance state. The product of the input and output can be measured at the SL, where a measured $V_{SL} = INV$ corresponds to logic '1'.

Security-Reconfigured PE Array. Within a single crossbar, binary weights are stored in the array. Input vectors are CL voltages and the MAC results are read out in parallel from the SL invectors. The WL, BL, and INV control RRAM selection, programming and encryption, respectively, as shown in Fig. 3 (b).

We reconfigure a chip with a crossbar size of 128×128. Cells on the same SL share the same inverter supply voltage (INV) (red line in Fig. 3(b)). Different SLs have different INV voltages. Each combination of INV voltages across all SLs is a key for encryption. This is becacuse, for a given SL, as the INV voltage increases from 1.2V to 3.0V, the SL voltage (before inverter) rises (Fig. 4(b)) accordingly, Consequently, the MAC threshold where the SL logic output flips right shifts. For instance, as shown in Fig. 4(c), when the INV is 2.0V, the MAC threshold is 0. When the INV reduces to 1.6V, the SL logic output is '-1' only if the SL MAC result is greater than threshold 50; otherwise, it is '1'.

As a result, the 3T2R array performs both MAC operations and reconfigurable binarizations, equivalent to a binary neural network with tunable sign activation.

RRAM-friendly Encryption. To illustrate DNN protection, we use a single MLP layer as an example. Fig. 5 (a) shows

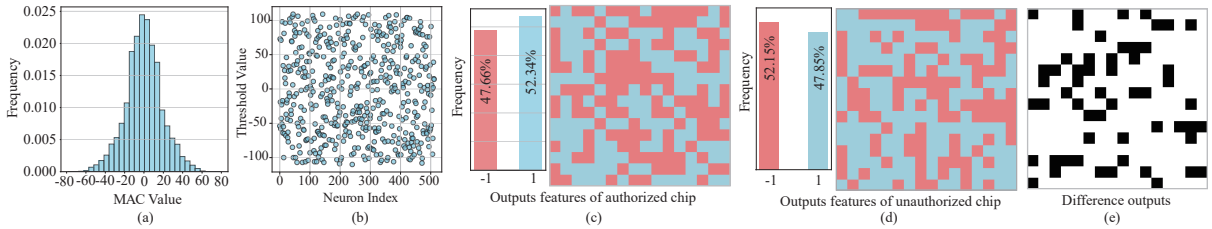


Fig. 5. Encryption comparisons. (a) Histogram of MAC results; (b) Threshold values for different output neurons; (c) Output features of the authorized chip; (d) Output features of the unauthorized chip; (e) Feature differences.

the output features the MLP layer, which follows a Gaussian distribution centered around 0. We first implement this layer on the authorized chip, with a secret key k illustrated in Fig. 5 (b). Here the key refers to the collection of MAC thresholds of different SLs, ranging from -110 to 110, which are physically implemented using different INV voltages on different SLs. The encrypted authorized chip's output features in Fig. 5 (c) show '-1' and '1' counts of 47.66% and 52.34%, respectively. In contrast, the unauthorized chip with a default key (i.e. 0) produces features shown in Fig. 5 (d), with '-1' and '1' counts of 52.15% and 47.85%. The feature map differences are revealed in Fig. 5 (e). Thus, the authorized chip's output significantly differs from the unauthorized one, thank to lightweight and fast INV settings without dedicated encryption/decryption modules.

D. Software Design

Naive Training. One intuitive method is to train the weights with the authorized chip by minimizing corresponding loss function. However, this approach cannot ensure that the unauthorized chip used by an attacker will be unusable.

Specifically, as illustrated in Fig. 6 (a), the performance gaps across various models, including shallow MLP models on MNIST dataset and deeper architectures like ResNet and ViT, on CIFAR10 dataset classification are approximately 13%, 10%, and 2%, respectively. These relatively minor differences are insufficient.

Contrastive Training. To increase this performance gap, we propose a novel contrastive training method. This approach aims to train a DNN that excels when running on the authorized chip with a secret key, but performs poorly on the unauthorized chip with the default key (i.e., INV = 2.0V, corresponding to SL MAC threshold = 0).

Let X and Y^t denote the input data and true labels of the dataset, respectively. f represents the DNN model, and $\mathcal{L}$ is the loss function (cross-entropy or mean squared error). Our contrastive training can be formulated as,

$$\begin{aligned}\min \mathcal{L} &= \min_X \mathbb{E}\mathcal{L}(f(k,X),Y^t) + \lambda \times \max_X \mathbb{E}\mathcal{L}(f(0,X),Y^t) \\ &\simeq \min_X \left(\mathbb{E}\mathcal{L}(f(k,X),Y^t) - \lambda \times \mathbb{E}\mathcal{L}(f(0,X),Y^t)\right)\end{aligned} \tag{1}$$

Here, $\mathbb{E}$ represents the expectation operator, and λ is a hyperparameter controlling the importance of accuracy gap between the authorized and unauthorized chips. k represents the INV voltage set of our chip.

This formulation aims to minimize the loss with the secret key k, while maximizing the loss with the default key 0, enhancing the performance gap between authorized and unauthorized scenarios.

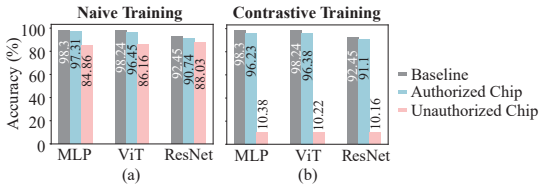


Fig. 6. The performance comparison of (a) the naive training and (b) contrastive training on classification models.

To prevent training collapse due to a negative total loss $\mathcal{L}$, we redefine the hyperparameter λ using magnitude normalization. Specifically, we compute the logarithmic difference between $\mathcal{L}(f(0,X),Y^t)$ and $\mathcal{L}(f(k,X),Y^t)$, and adjust it with α,

$$\lambda = \alpha \times 10^{\left(\log_{10} \frac{\mathcal{L}(f(0,X),Y^t)}{\mathcal{L}(f(k,X),Y^t)}\right)} \tag{2}$$

This redefinition ensures the magnitude difference between $\mathcal{L}(f(0,X),Y^t)$ and $\mathcal{L}(f(k,X),Y^t)$ is appropriately adjusted, maintaining the primary task of enhancing the accuracy of the authorized chip.

IV. Evaluation

A. Experimental Setup

Software Benchmarks. Evaluations were conducted on tasks including classification, detection, segmentation, and image generation. For classification, an MLP with four fully connected layers was validated on the MNIST dataset [29], while deeper models such as ResNet [30] and Vision Transformer (ViT) [1] were validated on the CIFAR-10 dataset [31]. Segmentation tasks utilized Segformer B0 model on the VOC dataset. For image generation, the Diffusion Transformer (DiT) B/2 [2] was validated on the Animal Faces-HQ (AFHQ) dataset [32]. In these models, the weights of the layers requiring encryption were quantized to binary weights before being mapped to 3T2R array. All models were trained using the PyTorch framework.

Hardware Accelerators. The Cadence SPICE model of the 3T2R array was simulated using UMC's 180 nm technology node. The RRAM in the 3T2R cell was modeled with low-resistance state (LRS) and high-resistance state (HRS) of 10 kΩ and 100 kΩ, respectively. In the 3T2R array, VRH and VRL were set to 0.8 V and 0.5 V, respectively. The default INV was set to 2.0 V, which means the key element or threshold (k) in Eq. (1) is 0. The array size was fixed to 128×128. As the output of the 3T2R array is binary, no peripheral ADC circuit was necessary.

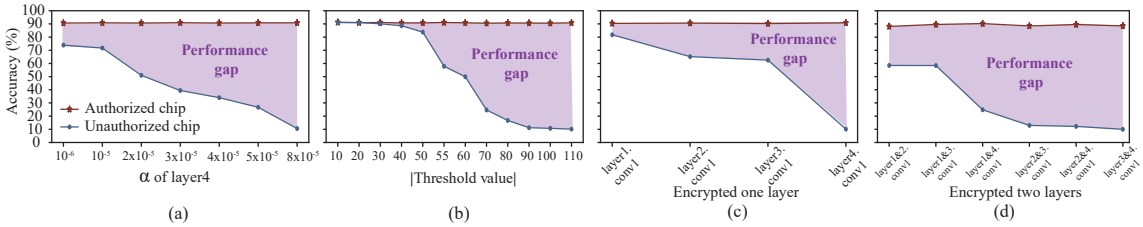


Fig. 7. IP protection effect on a ResNet model with (a) different scaling factors α; (b) The absolute value of threshold; (c) Encryption position index and (d) dual-layer encryption effect.

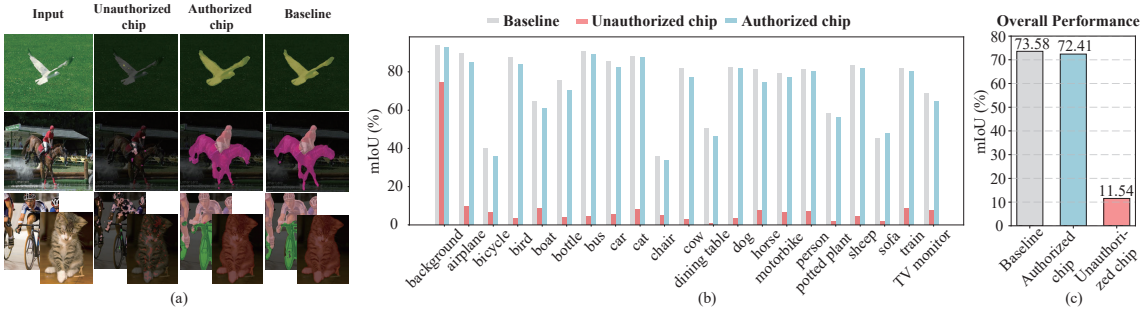


Fig. 8. IP protection effect on the semantic segmentation task. (a) Qualitative results; (b) Class-wise mean Intersection over Union (mIoU).

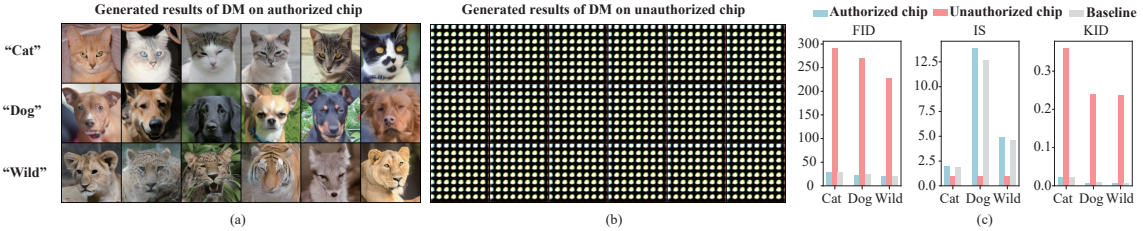


Fig. 9. IP protection effect on the image generation task. (a) Output from the authorized chip; (b) Output from the unauthorized chip; (c) Evaluation metrics, where Fréchet Inception Distance (FID) and Kernel Inception Distance (KID) are better when smaller, and Inception Score (IS) is better when larger.

B. Software IP Protection Evaluation

Object Classification. Fig. 7 analyzes the ResNet model under different hyperparameters, showing variations in the performance gap between authorized and unauthorized chips. Specifically, as illustrated in Fig. 7 (a), when α is very small (e.g., less than 10^{-5}), the performance gap is less than 20%. However, as α increases, the performance gap widens. Furthermore, the IP protection effect is proportional to the size of the secret key, or INV voltage, space. As shown in Fig. 7 (b), for the ResNet model, when the absolute value of the SL MAC threshold is less than 50 ($-50 <$ "SL MAC threshold" < 50), and the corresponding INV voltage is between 1.6V and 2.3V (see Fig. 4 (c)), the performance gap is small, failing to achieve the desired protection effect. However, when the absolute value of the SL MAC threshold is 90 ($-90 <$ "SL MAC threshold" < 90), with the corresponding INV voltage between 1.4V and 2.5V, the performance of the unauthorized chip approaches random guessing.

We also analyzed the impact of the impact of layer to be encrypted, as shown in Fig. 7 (c). When α is set to 0.0001 and the absolute value of the threshold is 110, encrypting deeper layers results in a larger performance gap. Fig. 7 (d) illustrates the effect of dual-layer encryption. Compared to single-layer encryption, dual-layer encryption significantly increases the performance gap.

Using these methods, we conducted generalization capability tests on both shallow MLPs and deep ViTs. The results are presented in Fig. 6 (b). Our proposed contrastive training method significantly enhances the performance gap between authorized and unauthorized chips. Furthermore, compared to the baseline model running on a GPU, the model deployed on the authorized chip exhibits minimal performance degradation, with only a 1-2% decrease.

Semantic Segmentation. Fig. 8 demonstrates the effectiveness of IP protection in the segmentation task. For the model on unauthorized chips, most object-related pixels are misclassified. However, on the authorized chip, the segmentation results closely match the baseline segmentation performance (Fig. 8 (a)). Quantitatively, the segmentation ability on the unauthorized chip is very poor for almost all classes, except for the background. In contrast, on the authorized chip, the segmentation performance is nearly identical to that of the software baseline (Fig. 8 (b)). The overall mIoU of the model on the authorized chip reached 72.41%, while on the unauthorized chip it was only 11.54%, proving the effectiveness of the method.

Image Generation. Fig. 9 demonstrates the effectiveness of our method when applied to generative DiT models. Fig. 9 (a) shows that our model can generate high-quality image samples on the authorized chip, while in Fig. 9 (b), no meaningful images can be generated on the unauthorized chip. Fig. 9 (c) compares the image generation metrics between the authorized chip, the unauthorized chip, and the baseline. Across all three categories, the authorized chip (average FID: 23.31, IS:

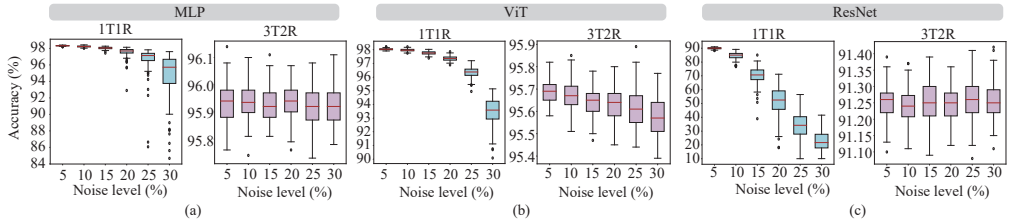


Fig. 10. Robustness test results for the 3T2R and 1T1R chip designs on (a) MLP, (b) ViT and (c) ResNet.

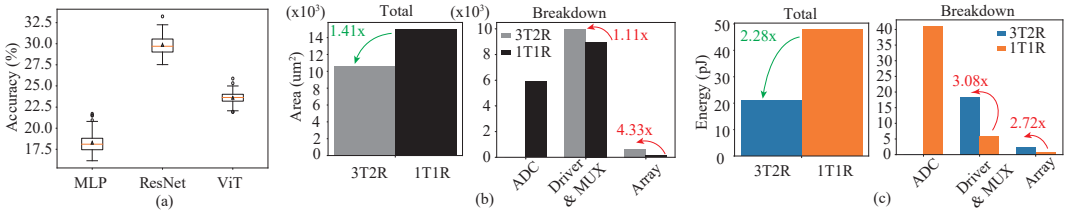


Fig. 11. Security and Area & Energy analysis. (a) Random guessing attack, and each model was tested over 100 trials. In the box, the orange line indicates the median, while the triangle denotes the mean; (b) Area and energy comparison between 3T2R and 1T1R.

6.89, KID: 0.012) achieves image quality comparable to the baseline (average FID: 24.58, IS: 6.40, KID: 0.012), while the unauthorized chip (average FID: 262.84, IS: 1.00, KID: 0.28) fails to generate any valid images.

C. Hardware Performance Evaluation

Robustness. Fig. 10 illustrates the robustness test results for the 3T2R and 1T1R chip designs under RRAM programming noise levels of 5%, 10%, 15%, 20%, 25%, and 30%, with each condition tested over 100 trials. As the noise level increases to 30%, the 3T2R chip's accuracy declines by less than 1%, while the 1T1R chip's accuracy drops significantly. Specifically, for a shallow model like MLP, as noise increases from 5% to 30%, the 3T2R chip exhibits almost no change in accuracy, whereas the 1T1R chip experiences a maximum decrease of 10%. However, for deeper models such as ViT or ResNet, the accuracy of the 1T1R chip gradually decreases with increasing noise, dropping by approximately 8% when noise reaches 30%. In contrast, the 3T2R chip remains very stable, showing a decline of less than 0.5%. This remarkable stability is primarily attributed to the INV inverter in the 3T2R design, which effectively filters out RRAM programming stochasticity compared to the 1T1R design [17].

Security. According to Kerckhoff's Principle and Shannon's Maxim [33], attackers can understand the protection method except for the secret keys. In this context, the secret key refers to the collection of INV voltages to 3T2R array. Assuming an array size of 128×128, there are 128 INV voltages to be guessed. As shown in Fig. 4 (c), during normal operation, the range of INV can span from 1.2V to 2.5V. In practice, we assume INV takes discrete values at 0.1V intervals, giving each INV 14 possibilities. If an attacker attempts to use exhaustive search to find the exact key, the key space for an array of size 128×128 would be 14^{128}, which is an astronomical number. We also assume that the attacker can make random guesses; the performance of the model with 100 random trials remains extremely low, demonstrating the ineffectiveness of random guessing attacks (see Fig. 11 (a)).

Area & Energy. Fig. 11 (b) and (c) compare the area and energy consumption of 128×128 array chips using the same software benchmarks. Compared to the 1T1R-based PIM accelerator, our 3T2R-based PIM accelerator achieves overall reductions in both area and energy consumption by 1.41× and 2.28×, respectively. This is primarily due to the elimination of the need for ADCs in the 3T2R PIM. Specifically, although the array composed of the proposed 3T2R-based PIM occupies 4.33× more area and consumes 2.72× more energy than the 1T1R-based PIM [17], and the 3T2R design incurs 1.11× higher area and 3.08× higher energy costs for the driver and MUX components, the elimination of ADCs significantly compensates for the overhead introduced by the 3T2R-based PIM accelerator in terms of array and driver and MUX components.

V. Conclusion

RRAM-based PIM accelerators are promising for addressing energy consumption issues at the edge, but often overlook IP security and RRAM's inherent stochasticity. This paper proposes a hardware-software co-design Guarder to address these challenges. We introduce a robust 3T2R cell design to mitigate RRAM stochasticity and reconfigure the array by adjusting the INV voltage, preventing attackers from exploiting stolen models. Additionally, a contrastive training method ensures that trained model parameters work only on authorized chips. We believe that the proposed methods enhance the security and robustness of RRAM in edge applications.

Acknowledgment

This research is supported by the National Key R&D Program of China (Grant No. 2022YFB3608300), National Natural Science Foundation of China under Grant 62488101, 62495104 and 62374181, Shenzhen Science and Technology Innovation Commission (Grant No. SGDX20220530111405040), Beijing Natural Science Foundation (Grant No. Z210006), Hong Kong Research Grant Council (Grant Nos. 27209621, 17205922, 17212923), Joint Laboratory of Microelectronics (JLFS/E-601/24). This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[2] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023.

[3] Pranav Rajpurkar, Awni Y Hannun, Masoumeh Haghpanahi, Codie Bourn, and Andrew Y Ng. Cardiologist-level arrhythmia detection with convolutional neural networks. *arXiv preprint arXiv:1707.01836*, 2017.

[4] AMD. Mercedes-benz accelerates innovation with amd technology. https://www.amd.com/en/resources/case-studies/mercedes.html.

[5] AMD. Subaru leverages amd technology to enhance ai-powered automotive safety systems. https://www.amd.com/en/resources/case-studies/subaru.html.

[6] Shuochao Yao, Yiran Zhao, Aston Zhang, Shaohan Hu, Huajie Shao, Chao Zhang, Lu Su, and Tarek Abdelzaher. Deep learning for the internet of things. *Computer*, 51(5):32–41, 2018.

[7] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference system for neural networks. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, pages 27–30, 2020.

[8] Sourjya Roy, Shrihari Sridharan, Shubham Jain, and Anand Raghunathan. Txsim: Modeling training of deep neural networks on resistive crossbar systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(4):730–738, 2021.

[9] Siddhartha Chhabra and Yan Solihin. i-nvmm: A secure non-volatile main memory system with incremental encryption. In *Proceedings of the 38th annual international symposium on Computer architecture*, pages 177–188, 2011.

[10] Bin Gao, Bohan Lin, Yachuan Pang, Feng Xu, Yuyao Lu, Yen-Cheng Chiu, Zhengwu Liu, Jianshi Tang, Meng-Fan Chang, He Qian, et al. Concealable physically unclonable function chip with a memristor array. *Science advances*, 8(24):eabn7753, 2022.

[11] Yi Cai, Xiaoming Chen, Lu Tian, Yu Wang, and Huazhong Yang. Enabling secure in-memory neural network computing by sparse fast gradient encryption. In *ICCAD*, pages 1–8, 2019.

[12] Yi Cai, Xiaoming Chen, Lu Tian, Yu Wang, and Huazhong Yang. Enabling secure nvm-based in-memory neural network computing by sparse fast gradient encryption. *IEEE Transactions on Computers*, 69(11):1596–1610, 2020.

[13] Ali Shafiee, Anirban Nag, Naveen Muralimanohar, Rajeev Balasubramonian, John Paul Strachan, Miao Hu, R. Stanley Williams, and Vivek Srikumar. Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 14–26, 2016.

[14] Ping Chi, Shuangchen Li, Cong Xu, Tao Zhang, Jishen Zhao, Yongpan Liu, Yu Wang, and Yuan Xie. Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 27–39, 2016.

[15] Zhongrui Wang, Saumil Joshi, Sergey E Savel'ev, Hao Jiang, Rivu Midya, Peng Lin, Miao Hu, Ning Ge, John Paul Strachan, Zhiyong Li, et al. Memristors with diffusive dynamics as synaptic emulators for neuromorphic computing. *Nature materials*, 16(1):101–108, 2017.

[16] Linghao Song, Xuehai Qian, Hai Li, and Yiran Chen. Pipelayer: A pipelined reram-based accelerator for deep learning. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 541–552, 2017.

[17] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. *Nature*, 577(7792):641–646, 2020.

[18] Yulin Zhao, Jie Yu, Donglin Zhang, Qiao Hu, Xuanzhi Liu, Haijun Jiang, Qingting Ding, Zhongze Han, Jinhui Cheng, Wenjun Zhang, et al. A 0.02% accuracy loss voltage-mode parallel sensing scheme for rram-based xnor-net application. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6):2697–2701, 2022.

[19] Linfang Wang, Wang Ye, Chunmeng Dou, Xin Si, Xiaoxin Xu, Jing Liu, Dashan Shang, Jianfeng Gao, Feng Zhang, Yongpan Liu, et al. Efficient and robust nonvolatile computing-in-memory based on voltage division in 2t2r rram with input-dependent sensing control. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1640–1644, 2021.

[20] Miguel Angel Lastras-Montano and Kwang-Ting Cheng. Resistive random-access memory based on ratioed memristors. *Nature Electronics*, 1(8):466–472, 2018.

[21] Qi Liu, Bin Gao, Peng Yao, Dong Wu, Junren Chen, Yachuan Pang, Wenqiang Zhang, Yan Liao, Cheng-Xin Xue, Wei-Hao Chen, Jianshi Tang, Yu Wang, Meng-Fan Chang, He Qian, and Huaqiang Wu. 33.2 a fully integrated analog reram based 78.4tops/w compute-in-memory chip with fully parallel mac computing. In *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 500–502, 2020.

[22] Minhui Zou, Zhenhua Zhu, Yi Cai, Junlong Zhou, Chengliang Wang, and Yu Wang. Security enhancement for rram computing system through obfuscating crossbar row connections. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 466–471. IEEE, 2020.

[23] Nvidia jetson nano developer kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/. Accessed: 2023-10-20.

[24] Weizhe Hua, Zhiru Zhang, and G Edward Suh. Reverse engineering convolutional neural networks through side-channel information leaks. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[25] Lejla Batina, Shivam Bhasin, Dirmanto Jap, and Stjepan Picek. {CSI}{NN}: Reverse engineering of neural network architectures through electromagnetic side channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 515–532, 2019.

[26] Qingli Guo, Jing Ye, Yue Gong, Yu Hu, and Xiaowei Li. Puf based pay-per-device scheme for ip protection of cnn model. In *2018 IEEE 27th Asian Test Symposium (ATS)*, pages 115–120. IEEE, 2018.

[27] Ning Lin, Xiaoming Chen, Hang Lu, and Xiaowei Li. Chaotic weights: A novel approach to protect intellectual property of deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(7):1327–1339, 2021.

[28] A. Skendžić et al. General data protection regulation — protection of personal data in an organisation. In *MIPRO*, 2018.

[29] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[32] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197, 2020.

[33] Claude E Shannon. Communication theory of secrecy systems. *The Bell system technical journal*, 1949.

Adora Compiler: End-to-End Optimization for High-Efficiency Dataflow Acceleration and Task Pipelining on CGRAs

Jiahang Lou, Qilong Zhu, Yuan Dai, Zewei Zhong, Wenbo Yin and Lingli Wang*
State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China
*Email: llwang@fudan.edu.cn

***Abstract*—To fully harness emerging computing architectures, compilers must provide intuitive input handling alongside powerful code optimization to unlock maximum performance. Coarse-Grained Reconfigurable Arrays (CGRAs) — highly energy-efficient for nested-loop applications — have lacked a compiler capable of meeting these objectives. This paper introduces the Adora compiler [1], which effectively bridges user-friendly, lightweight coding inputs with high-performance acceleration on the CGRA SoC. Adora utilizes CGRA-target loop transformations to achieve efficient data-flow level execution while optimizing data communication and task pipelining at the task-flow level. Additionally, it incorporates a comprehensive automated algorithm with a thoughtfully designed optimization sequence. A series of comprehensive experiments highlights the exceptional efficiency and scalability of the Adora compiler, demonstrating its transformative impact in leveraging CGRA capabilities for acceleration in edge computing.**

***Index Terms*—CGRA compiler, loop transformation**

I. Introduction

Compared to traditional reconfigurable architectures such as Field Programmable Gate Arrays (FPGAs), Coarse-Grained Reconfigurable Arrays (CGRAs) exploit time-multiplexing of computational resources by switching configurations at the word-level granularity, rather than the bit-level. This enables rapid reconfiguration within a few cycles, enhancing both flexibility and efficiency of limited resources. Word-level granularity allows CGRAs to achieve energy efficiency on par with that of Application-Specific Integrated Circuits (ASICs). Consequently, CGRAs are increasingly gaining prominence as a viable architecture for edge computing applications, such as AI inference and baseband processing [2].

Motivation. This paper focuses on compilation automation and automated tuning for comprehensive CGRA applications. We argue that the compiler should offer a user-friendly programming model along with powerful performance optimizations to achieve high-efficiency programming and execution on CGRAs. Although previous works have made attempts at CGRA source code tuning for optimized execution [3]–[5], several limitations remain:

Limitation 1: Difficulty in coding an optimal nested loop for CGRA. Given that CGRAs are commonly used for loop acceleration in the literature, writing code that maximizes performance by carefully considering both data locality and parallelism remains challenging for software developers without in-depth knowledge of the architecture. This necessitates the development of a highly automated compiler incorporating a suite of tailored optimizations specifically designed to enhance data-flow execution on CGRAs.

Limitation 2: Using CGRA SoCs for real-world applications requires more than just loop crafting. For a post-silicon CGRA SoC, it is essential to manage not only the accelerated kernel but also the orchestration of the entire system, including data movement, CGRA reconfiguration and execution, inter-task pipelining, and hierarchical memory management—particularly for complex applications involving large data sizes. Therefore, optimizations for task pipelining on the CGRA SoC must be integrated into an end-to-end push-button compilation framework.

Limitation 3: The vast optimization space in CGRA coding makes finding the most efficient execution method challenging. Both software loop transformations and hardware flexibility extend the vast optimization space on source code for CGRA execution. Sometimes, the optimization can be insufficient or even counterproductive, as demonstrated in Fig. 3 and Fig. 4 in Sec. III. This inspires us to thoroughly analyze the execution characteristics of CGRA, tailor loop transformations specifically for CGRA, and develop a fast algorithm with a well-crafted optimization sequence.

Therefore, comprehensive end-to-end automation and thorough optimization are crucial to reducing design complexity, thereby removing barriers for software developers and enabling broader and easier adoption of CGRA technology.

Related work. Previous works have considered several common loop transformations, like loop reordering [5], [6], fusion/fission [6], [7], unrolling [4], [5], flattening [7], etc. Some of these techniques target special CGRA architectures [6] or particular application types [5], [8]. Comprehensive loop transformations and an automated tuning mechanism make a CGRA compiler more practical, but focusing solely on CNNs limits versatility [5]. Recently, a mapper for spatial-temporal CGRA with comprehensive loop transformation exploration with a GNN-based prediction model is presented in [3].

Contributions. Our approach surpasses previous works by introducing hardware-specific optimizations that signifi-

This work is supported by National Science and Technology Major Project (2021ZD0114701).

cantly reduce the optimization space, along with task-flow level enhancements for post-silicon execution. Furthermore, an automated algorithm integrates these optimizations in a systematic manner.

The contributions of this work are:

- Customized data-flow level optimizations for efficient CGRA execution, combined with novel optimizations for data communication and effective task pipelining at the task-flow level, leveraging polyhedral-based affine transformations and dependency analysis.
- A comprehensive algorithm featuring a carefully designed optimization sequence to explore optimal source code transformations within the combined data-flow and task-flow optimization space.
- An end-to-end open-source framework [1] from user input to real execution/verification on the CGRA SoC, with transparent optimization and compiling flow to ease the afford of users.

II. Overview of Adora Compiler

A. Overall Compilation Workflow

The complete compilation flow of the Adora compiler is depicted in Fig. 1. Software designers can input algorithms written in C or PyTorch/TensorFlow models, with support provided by the MLIR infrastructure [9]–[12].

The process is followed by a seamless compilation flow that transitions from user input to final execution on the CGRA. Once transformed into an MLIR representation, the system aims to offload all loop computations onto the CGRA, maximizing opportunities to minimize reliance on the less efficient control core.

An automated optimization process from two distinct perspectives is employed: the first focuses on data-flow, maximizing loop acceleration on the target CGRA (Sec. III), while the second targets task pipelining across the entire system (Sec. IV). A fast algorithm for scheduling these optimization techniques to achieve optimal source code transformation is briefed in Sec. V.

Subsequently, the optimized kernels are mapped to the CGRA architecture, producing an executable for the CGRA SoC, which can be validated through simulation or FPGA prototyping.

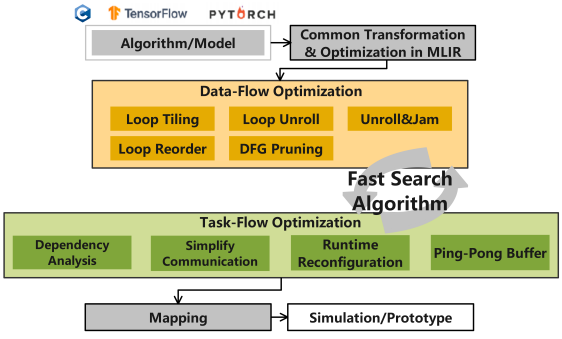


Fig. 1: Compiling Flow of Adora compiler

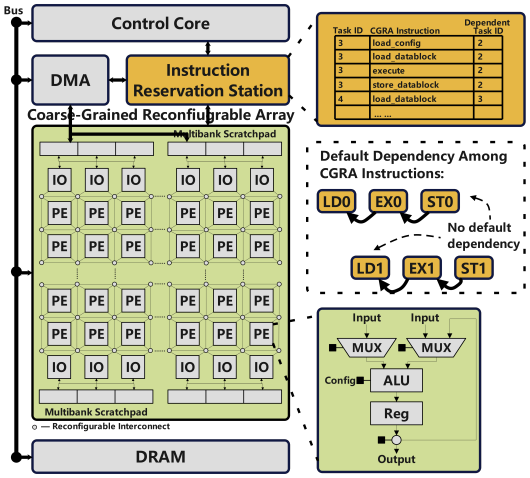


Fig. 2: Architecture of Targeted CGRA

B. Targeted CGRA Architecture

Fig. 2 shows the whole architecture of our targeted CGRA SoC [13]. The control core is a lightweight RISC-V Rocket CPU [14], managing CGRA instruction issuance, execution monitoring, and DRAM memory allocation. All computationally intensive tasks, particularly loop iteration computations, are offloaded to the CGRA whenever possible. The reservation station (RS), similar to those used in CPU design, stores CGRA instructions issued from the control core and manages out-of-order issuance by handling task dependencies. The coarse-grained reconfigurable array (CGRA) consists of processing elements (PE) and Input/Output (IO) blocks, interconnected through a reconfigurable network, supporting various computational patterns. IOs are positioned at the array's top and bottom to facilitate data transfers between the CGRA and the multi-bank scratchpad memory (SPM). The Direct Memory Access (DMA) supports broadcasting data across multiple SPM banks.

Our target CGRA is a Chisel-based architecture with user-defined parameters (e.g. array size, supported operations, memory size, and bus width). Adora incorporates architecture descriptions at each stage of the compilation process, enabling adaptability across diverse CGRA architectures.

C. Execution Model

CGRA users are relieved from managing the specifics of loop computation on the CGRA, as Adora automatically generates an execution model. This model resembles a CPU-GPU or CPU-accelerator framework, incorporating explicit data movement, CGRA configuration and execution, as shown in Fig. 3(a). The control core initiates a thread that offloads most of computing kernels to the CGRA.

The default CGRA instruction dependencies are illustrated in Fig. 2. Note that a `load_datablock`(DRAM-to-SPM), `execution` and `store_datablock`(SPM-to-DRAM) within the same task ID are inherently dependent. Intra-task dependencies can be specified by adding a dependent ID to the instruction.

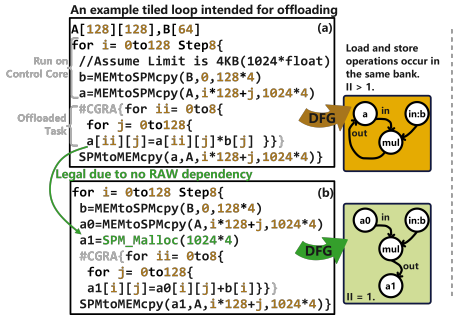


Fig. 3: (a) Tiling example, (b) Automated new bank allocation for output storage in the absence of RAW dependency.

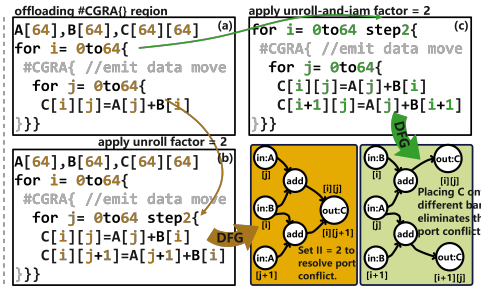


Fig. 4: (a) Original code, (b) loop unroll-and-jam as a replacement for (c) ineffective unrolling.

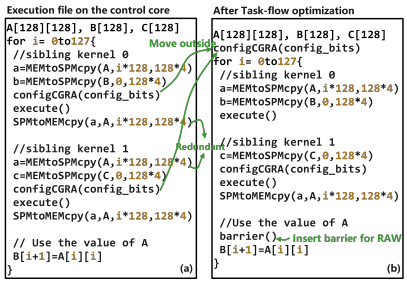


Fig. 5: Task-Flow Level Optimization Example.

Based on the RS design, the control core does not wait for CGRA execution to finish. Instead, it continues issuing DMA and CGRA instructions, halting only in response to RS back-pressure or a `barrier` requirement.

III. Data-Flow Level Optimizations

In this section, we focus on transforming nested loops into data-flow execution on CGRA using dependency analysis and affine transformations based on the polyhedral model.

For a specific memory access statement s in a nested-loop, either `load` or `store`, let the iteration vector $\vec{i}$ be composed of the indices of the corresponding loop iteration variables. The constraints for this memory access are:

$$C_{\vec{i}} \cdot \vec{i} = \begin{bmatrix} C^{\text{LB}}_{\vec{i}} \\ C^{\text{ARRAY}}_{\vec{i}} \end{bmatrix} \cdot \vec{i} \geq 0 \tag{1}$$

where C^{LB} and C^{ARRAY} represent constraints of the loop bounds and array access bounds, respectively. For two memory access statements s and t, the dependency polyhedron is:

$$\mathcal{P}_e(s,t) \equiv \left[\begin{array}{cc} C_{\vec{i_s}} & \\ & C_{\vec{i_t}} \\ \hline & R_{\vec{i_s}\to\vec{i_t}} \end{array}\right] \begin{bmatrix} \vec{i_s} \\ \vec{i_t} \\ 1 \end{bmatrix} \begin{bmatrix} \geq 0 \\ = 0 \end{bmatrix} \tag{2}$$

where $R_{\vec{i_s}\to\vec{i_t}}$ is the affine relationship from s to t. Equation (2) applies to two statements either within the same iteration or across different iterations. We solve (2) to determine whether two memory access statements exhibit flow- or output-dependency.

A. Loop Tiling and Local Memory Management

The primary objective of **loop tiling** is to create a tile of the iteration space that minimizes data communication. Unlike tiling for CPUs [15], tiling for CGRAs must adhere to the stringent constraints imposed by the SPM size in CGRAs, i.e., the accessed memory space satisfies the available local SPM capacity.

A tiled iteration space, called a tile or a task, which will be offloaded onto CGRA, can be defined by

$$\left[\begin{array}{cc} C_{\vec{i}} & \\ & C^{\text{Tile}}_{\vec{j}} \\ \hline & R_{\vec{i}\to\vec{j}} \end{array}\right] \begin{bmatrix} \vec{i} \\ \vec{j} \end{bmatrix} \begin{bmatrix} \geq 0 \\ = 0 \end{bmatrix} \tag{3}$$

where $\vec{j}$ represents the transformed iteration vector after tiling, C^{Tile} denotes the constraints for the offloaded data block, which is a sub-view of a larger memory reference, stored in CGRA local SPM. The tiled memory region defined by (4) must adhere to the SPM size limit:

$$C^{\text{Tile}}_{\vec{j}} \cdot \vec{j} \geq 0 \tag{4}$$

Adora identifies all feasible tiling methods that satisfy local memory size constraints by systematically examining each level of the nested loop, from the outermost to the innermost. The goal is to determine the tile configuration that maximizes local memory utilization.

Flow-dependencies are also analyzed with (2) for a tile intended for offloading. As illustrated in Fig. 3(a), the kernel accesses the same bank with both a read and write operation in each iteration, but without any flow-dependency. Consequently, a new local SPM bank can be allocated for write operations to prevent memory port contention and ensure the smooth execution of the pipeline.

B. Loop Unroll and Unroll-and-Jam

Given the time-multiplexed nature of CGRAs, maximizing parallelism within computing and memory constraints makes loop unrolling crucial.

Loop Unrolling has been explored by previous works [3]–[5], [8]. However, indiscriminate loop unrolling may result in SPM port conflicts among write operations (while load conflicts are mitigated through data block broadcasting), potentially negating performance gains, as shown in Fig. 4(b).

The port conflicts of `store` statement s is checked by:

$$C^{\text{Tile}}_{\vec{j}} \cdot \Delta\vec{j} > 0 \tag{5}$$

where $C^{\text{Tile}}_{\vec{j}}$ is defined in (3), $\Delta\vec{j}$ denotes the change in the iteration vector as each loop level advances by its own step. The satisfaction of (5) indicates the presence of write port conflicts, leading to the initial interation (II) equivalent to the unrolling factor, thereby providing no benefit for loop execution on the CGRA. In this context, the loop unroll-and-jam offers an effective alternative solution.

Loop unroll-and-jam unrolls the outer loop and fuses inner loops, reducing memory instruction dispatch overhead and enhancing parallelism for efficient software pipelining for CPUs. Applying this approach to CGRA creates additional optimization opportunities, as illustrated by the transformation of the green flow in Fig. 4(c). Specifically, we unroll the outer loop of the tile and fuse the resulting duplicated tiles. This strategy not only enhances data reuse but also generates an additional tiled data block, thereby preventing memory write conflicts. The resulting execution model resembles the Single-Instruction-Multiple-Thread (SIMT) paradigm. However, the presence of flow-dependency hazards renders unroll-and-jam techniques ineffective.

To fully utilize the computing resources of CGRA, the unrolling or unroll-and-jam factors should be maximized based on the hardware resource limits, effectively making unrolling a binary strategy: apply either unrolling or unroll-and-jam up to the maximum resource limit. Notably, Adora unrolls multiple levels of loops, not limited to the innermost loop, utilizing the largest feasible unrolling factors.

C. Loop Reorder

Loop reorder is a key contributor to the vast optimization space for nested loops, as each new loop permutation creates a distinct optimization space for all other possible loop transformations, leading to definitely different tile. In the context of CGRA, reordering aims to find the optimal loop permutation that maximizes data reuse and minimizes communication volume for the offloaded tile, thereby enhancing overall efficiency.

Our primary focus is on optimizing reuse within the innermost two levels of the computation, as this segment exerts the most significant impact on the data-flow execution efficiency on CGRA. Adora comprehensively evaluates both temporal and spatial group reuse for every load and store operation across all legal loop permutations. For load and store operations exhibiting flow-dependencies(RAW), Adora also analyzes self-reuse, with a preference for loop permutations where these flow-dependencies are confined to the innermost levels. This facilitates effective tiling and ensures that these operations can be efficiently executed within a single tile, thereby optimizing data locality and reducing inter-tile communication overhead.

D. Optimized Data-flow Graph Generation

In addition to loop transformations, the Data-Flow Graph (DFG) generation process for a tile refines the DFG with **rewriting** techniques by combining simpler operations into hardware-supported complex operators (such as `accumulation`). For unsupported operators, Adora substitutes them through a combination of simpler operators, ensuring flexibility in hardware usage (such as `rsqrt`,`tanh`,`exp`).

The DFG generation also accommodates more complex loop structures, such as imperfectly nested loops and branches. Furthermore, Adora supports non-constant loop boundaries through index computation or runtime reconfiguration as needed. Adora integrates these approaches seamlessly, allowing efficient execution for a wide range of scenarios on CGRA.

IV. Task-Flow Level Optimizations

This section analyzes the whole task-flow from a high-level perspective of executing each tile during iteration. Adora carefully orchestrates instructions for pipelined execution by leveraging the out-of-order instruction issuance mechanism based on thorough task dependency analysis. Additionally, Adora incorporates runtime reconfiguration to accommodate varying requirements that emerge during execution, and employs compiler-supported ping-pong buffering to effectively overlap data communication with computation.

A. Task Dependency Analysis

Since the default dependency supported by the hardware is defined as shown in Fig. 2, only output and flow-dependencies between different tasks need to be handled.

First, we extend the concept of iteration vector for data block transfer instructions (`load_datablock` and `store_datablock`):

$$\vec{d} = [\vec{i}, \vec{shape}] \tag{6}$$

where $\vec{i}$ consists of the initial indices indicating the starting positions across each array dimension, with the length of the transferred data block in each dimension specified by $\vec{shape}$.

Then the dependency polyhedron between two data block transfer instructions s and t is:

$$\mathcal{P}_e(s,t) \equiv \left[\begin{array}{cc} C_{\vec{d}_s} & \\ & C_{\vec{d}_t} \\ \hline R_{\vec{d}_s \to \vec{d}_t} & \end{array}\right] \left[\begin{array}{c} \vec{d}_s \\ \vec{d}_t \\ 1 \end{array}\right] \left[\begin{array}{c} \geq 0 \\ \hline = 0 \end{array}\right] \tag{7}$$

where the constraints $C_{\vec{d}}$ still consist of the loop bounds and the original array bounds. Adora checks dependency between two data block transfers by solving (7). For dependent tasks, Adora embeds the dependency information into the dependent task ID in instructions, as illustrated in Fig. 2.

Additionally, when needed, Adora inserts `barrier` instructions to enforce flow dependencies via software.

B. Orchestration of Communication

Adora carefully manages data communication in the task-flow through three approaches:

First, memory access patterns become non-contiguous in DRAM address space after tiling, necessitating the insertion of Scatter-Gather instructions for data transfer. Additionally, unrolling or unroll-and-jam transformations may create overlapping memory access regions, so DRAM-to-SPM broadcast is enabled to support efficient data access.

Second, Adora **eliminates redundant CGRA instructions** by reusing configurations and previously transferred data. It identifies shared configurations or `load_datablock`s across loop iterations, thereby removing redundant DMA requests and CGRA configurations.

Third, in cases where sibling tasks exist in the same loop level, Adora analyzes and eliminates redundant load-after-store of data block dependencies using (7).

An example of the above optimizations is shown in Fig. 5, where redundant configuration and data transfer instructions are eliminated, and a barrier instruction is added to handle flow-dependency.

C. CGRA runtime reconfiguration

The rapid reconfiguration capability is utilized to adapt to runtime variations in offloaded tasks, with the control core dynamically adjusting configurations through bitstream modifications. Adora statically identifies all dynamically reconfigurable parts in the bitstream, such as memory access addresses, trip counts for tasks with variable iteration spaces, or constants that are instantiated only at runtime.

D. Software Supported Ping-Pong Buffer

Ping-pong buffer can be implemented via runtime reconfiguration, with two possible strategies: the first combines two adjacent banks into a ping-pong buffer, while the second splits a single bank into two halves. At runtime, the control core continuously adjusts the SPM address offsets for data block transfers and reconfigures the starting bank address for CGRA I/O during execution. This enables pipelining of execution and data block transfers, utilizing the out-of-order issuance of RS within the CGRA SoC.

However, the ping-pong buffer is not always the optimal choice, as it introduces reconfiguration overhead and affects both the number of available SPM banks and the size of each bank during CGRA execution.

V. Optimization Algorithm For Optimal Execution

By tailoring loop transformations specifically for CGRA architecture, the optimization space is significantly reduced, enabling a more targeted and efficient optimization exploration. Algorithm 1 presents our automation algorithm with carefully designed optimization sequence for nested loops, which integrates both data-flow and task-flow optimization. This algorithm comprehensively considers the interactions among various optimizations, ensuring a coordinated application of techniques to maximize execution efficiency.

First, we select the top 25% of loop permutations with high reuse in the innermost two levels for further exploration (line 2). Then, under three ping-pong buffer strategies—two configuration types or disabling (line 3), Adora applies tiling, selects between unroll or unroll-and-jam, generates the DFG, and performs comprehensive task-flow optimizations (line 4-11). After that, the ratio of II to resource utilization (RU) and total communication volume (TCV) is used to evaluate the result (line 12). Finally, the algorithm returns the top 5 Pareto-optimal results with minimal TCV.

Algorithm 1 Adora Optimization Process for a Nested-Loop

```
1: Calculate reuse for all valid loop permutations
2: for Loop permutations with high reuse do
3:   for Three Ping-pong buffer strategies do
4:     Apply loop tiling
5:     if Write port conflicts are detected by (5) then
6:       Perform unroll-and-jam to hardware limit
7:     else
8:       Perform unrolling to hardware limit
9:     end if
10:    Generate and map the optimized DFG
11:    Perform task-flow analysis and optimization
12:    Calculate the II/RU and TCV metrics
13:    Append result to result list
14:  end for
15: end for
16: return Top 5 minimal TCV solutions from the Pareto set.
```

VI. Evaluation

Broad evaluations demonstrate Adora's optimization efficiency and scalability. The comparison objects include a high-performance Von Neumann core and a reconfigurable device.

We synthesize CGRAs with two different sizes, using TSMC 40nm at 500 MHz: For a 6×8 CGRA, there are 2×8 IOs and 4×8 PEs, with 64-bit DMA bandwidth, 8KB per SPM bank, occupying 5.4mm^2 and consuming 0.52W. For a 12×16 CGRA, there are 2×16 IOs and 10×16 PEs, with 128-bit DMA bandwidth, 16KB per SPM bank, occupying 10.1mm^2 and consuming 0.84W. All performance results are based on VCS simulations and FPGA prototypes.

We present Adora's optimization results on Polybench [16], a widely used benchmark suite for evaluating nested loop acceleration, using the `large` dataset configuration. We then compare Adora against GCC-compiled kernels running on a single core of the advanced Intel i7-14700K processor, using the `-O2` optimization level.

Our end-to-end framework is also compared with another high-level compiler for FPGA on edge AI inference. We deploy three models to our 12×16 CGRA with Adora and the Intel DE10-Agilex FPGA board with AI Suite [17]. The models used are from widely adopted computer vision and language processing applications.

Additionally, our automated optimization algorithm is compared with a heuristic-based approach in terms of search time.

A. Accelerations for loop kernels

We present the results before and after Adora's optimization for the Polybench kernels in Tab. I. Our data-flow optimizations significantly enhance resource utilization, while task-flow optimizations effectively reduce DMA requests.

Fig. 6 presents a comprehensive evaluation using Polybench, which demonstrates the extensive kernel offloading capabilities of our end-to-end compiler. Our CGRAs achieve an average performance of 0.42× and 0.99× compared with single-core of

TABLE I: Comparison of Before- and After-Optimization Results for Polybench Benchmarks

Benchmarks		gesumv	gemm	sy2k	atax	cholesky	jacobi-1d	deriche
Before	II	2	1	1,2*	2,2	1	1,1	2,2,1,2,2,1
	#IO	7	4	2,6	3,4	4,3	4,4	7,7,1,7,7,1
	#PE	8	3	1,6	2,2	3,2	3,3	8,10,3,10,11,3
	#DMA request	54000	6017000	4×10^9	87597	9995009	9000	368154
After	II	1	1	1,1	1,1	1	1,1	2,2,1,2,2,1
	#IO	11	16	2,6	11,4	4,3	8,8	7,7,2,7,7,2
	#PE	9	12	1,6	10,2	3,2	6,6	8,10,6,10,11,6
	#DMA request	36072	4817452	3×10^9	68131	9372879	8001	139296

*A single benchmark may contain multiple kernels.
Total #PE = 32, #IO = 16

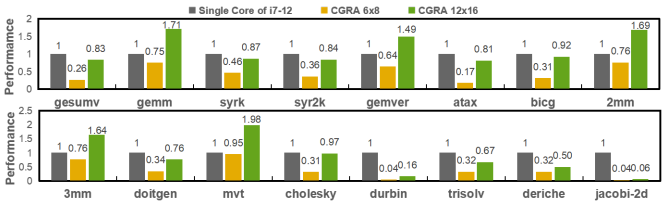


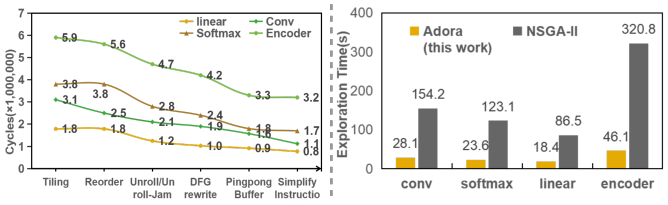
Fig. 6: Performance Comparison of CGRA Sizes vs. i7-14700K Single-Core on Polybench (i7 Peak Frequency Around 10× Higher)

TABLE II: Comparison of AI Inference Performance: Adora Compiler vs. Intel AI Suite

Model	AI Suite on Intel Agilex7 FPGA						
	Latency	Frequency	ALM	ALUT	Register	BRAM	DSP
Resnet-18	3.2ms	615MHz	23%	10%	19%	34%	13%
Alexnet	8.6ms	608MHz	23%	10%	18%	27%	11%
Bert	34ms*	600MHz	19%	9%	15%	14%	7%

Model	Adora on 12X16 CGRA				
	Latency	Frequency	PE	IO	SRAM
Resnet-18	113.8ms	500MHz	40%	59%	48%
Alexnet	450.8ms	500MHz	34%	75%	64%
Bert	192.2ms	500MHz	31%	62%	56%

*Result of `Bert` is estimated before P&R in AI Suite.



(a) Breakdown of experiments for various optimizations

(b) Exploration for optimal design: Adora vs. NSGA-II-based search

Fig. 7: (a) Breakdown of experiments for various optimizations, (b) Exploration for optimal design: Adora vs. NSGA-II-based search

i7-14700K, despite the significant technology and frequency differences (with a frequency gap of up to 10 times). Considering CGRA's much lower power consumption, the advantages of CGRA in edge computing are evident. The results indicate that CGRAs benefit from high parallelism and simple memory access patterns, such as `gemm`, `gemver`, and `mvt`, which are suitable for data-flow execution. However, extensive dependencies in benchmarks like `durbin` and `deriche` restrict optimization potential of CGRAs.

Moreover, the performance improvement observed between two CGRAs of different sizes demonstrates that Adora effectively extracts additional acceleration potential as the computational capacity of the underlying architecture increases.

B. Accelerations for AI inference

Although not specifically optimized for neural network acceleration, Adora still supports efficient edge AI inference. The comparison between our compiler for CGRA and the AI Suite for FPGA, shown in Tab. II, demonstrates that CGRA achieves comparable inference latency despite the technological differences (TSMC 40nm vs. Intel 10nm). This indicates that FPGAs are inherently suited for spatial computing, providing high parallelism and throughput. In contrast, CGRAs excel in flexible time-multiplexing, efficiently utilizing limited resources. Their low energy-latency product makes CGRAs particularly valuable for edge AI inference.

Furthermore, Adora's optimizations demonstrate more significant improvements in resource utilization for AI layers compared to the various complex loop structures in Polybench. This is due to the well-structured loop patterns and regular memory accesses in AI inference, making it well-suited for our two-level optimization approach.

C. Breakdown of Different Optimization Steps

Fig. 7(a) presents a detailed experimental analysis of Adora progressively applying various optimizations, including loop tiling, reordering, unrolling/unroll-jamming, DFG rewriting, ping-pong buffering, and instruction elimination. The selected AI layers, `conv2d` and `softmax` from `Alexnet`, along with `encoder` and `linear` from `Bert`, are used for evaluation.

D. Optimization Exploration Algorithm Speed

In Fig. 7(b), we compare Algorithm 1 with an NSGA-II-based loop optimization approach, which effectively finds globally optimal solutions by exploring all feasible loop transformations within an unrestricted search space. However, our optimization algorithm, utilizing a carefully designed sequence, finds the optimal solution, on average, six times faster across ten experiments for each case. This demonstrates the effectiveness of customized loop transformations for reducing exploration space, as well as the efficiency of our rapid exploration algorithm.

VII. Conclusion

The Adora compiler acts as a powerful bridge between user-friendly programming and CGRA SoC, minimizing barriers even for those without deep architectural knowledge. With the assistance of the polyhedral model and careful consideration of hardware features, the Adora compiler customizes a series of optimizations for CGRA source code at the data-flow level and designs high-level optimizations for effective task pipelining. Additionally, a fast automation algorithm orchestrates these optimizations, efficiently addressing all the limitations outlined in Sec. I.

References

[1] J. Lou, "adora-compiler." https://github.com/MIONkb/adora-compiler.

[2] Z. Hassan, A. Ometov, E. S. Lohan, and J. Nurmi, "Coarse-grained reconfigurable architectures for radio baseband processing: A survey," *Journal of Systems Architecture*, p. 103243, 2024.

[3] B. Shi, T. Dai, J. Zhang, X. Wei, and G. Luo, "Pt-map: Efficient program transformation optimization for cgra mapping," in *2024 61th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2024.

[4] T. Cheng, X. Chenhao, L. Ang, B. K. J, and T. Antonino, "Aurora: Automated refinement of coarse-grained reconfigurable accelerators," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1388–1393, IEEE, 2021.

[5] B. Inpyo, H. Barend, M. Hyemi, and E. Bernhard, "Auto-tuning cnns for coarse-grained reconfigurable array-based accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2301–2310, 2018.

[6] L. Huang, Y. Chen, and D. Liu, "Optimizing data reuse for loop mapping on cgras with joint affine and non-affine transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[7] L. Jongeun, S. Seongseok, L. Hongsik, and S. H. Uk, "Flattening-based mapping of imperfect loop nests for cgras," in *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, pp. 1–10, 2014.

[8] L. Jiahang, G. Xuchen, M. Yiqing, Q. Yunhui, H. Yihan, Y. Wenbo, and W. Lingli, "An agile deploying approach for large-scale workloads on cgra-cpu architecture," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, IEEE, 2024.

[9] C. Lattner, M. Amini, U. Bondhugula, A. Cohen, A. Davis, J. Pienaar, R. Riddle, T. Shpeisman, N. Vasilache, and O. Zinenko, "Mlir: Scaling compiler infrastructure for domain specific computation," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 2–14, 2021.

[10] W. S. Moses, L. Chelini, R. Zhao, and O. Zinenko, "Polygeist: Raising c to polyhedral mlir," in *2021 30th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 45–59, IEEE, 2021.

[11] LLVM, "Torch-mlir." https://github.com/llvm/torch-mlir.

[12] J. Pienaar, "Mlir in tensorflow ecosystem," 2020. Compilers For Machine Learning (C4ML) 2020, San Diego, CA, USA.

[13] Y. Qiu, Y. Mao, X. Gao, S. Chen, J. Li, W. Yin, and L. Wang, "Fdra: A framework for dynamically reconfigurable accelerator supporting multi-level parallelism," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 17, pp. 1–26, January 2024.

[14] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[15] Bondhugula U and Hartono A and Ramanujam J, et al., "Pluto: A practical and fully automatic polyhedral program optimization system," in *Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation(PLDI 08)*, 2008.

[16] L.-N. Pouchet, "Polybench: The polyhedral benchmark suite." https://www.cs.colostate.edu/~pouchet/software/polybench/.

[17] M. J. Ahmad, Jahanzeb and R. Venkata, "Intel fpgas and socs with intel fpga ai suite and openvino toolkit drive embedded/edge ai/machine learning applications." https://www.mouser.lt/pdfDocs/ai-fpga-whitepaper.pdf.

SimPhony: A Device-Circuit-Architecture Cross-Layer Modeling and Simulation Framework for Heterogeneous Electronic-Photonic AI System

Ziang Yin[1], Meng Zhang[2], Nicholas Gangi[2], Rena Huang[2], Jeff Zhang[1], Jiaqi Gu[1†]
[1]Arizona State University, [2]Rensselaer Polytechnic Institute
†*jiaqigu@asu.edu*

***Abstract*— Electronic-photonic integrated circuits (EPICs) offer transformative potential for next-generation high-performance AI, but they require interdisciplinary advances across devices, circuits, architecture, and design automation. The complexity of these hybrid systems makes it challenging even for domain experts to understand distinct behaviors and interactions across the design stack. The lack of a flexible, accurate, fast, and easy-to-use EPIC AI system simulation framework significantly limits the exploration of hardware innovations and system evaluations on common benchmarks. To address this gap, we propose SimPhony, a cross-layer modeling and simulation framework for heterogeneous electronic-photonic AI systems. SimPhony offers a platform that enables (1) generic, extensible hardware topology representation that supports heterogeneous multi-core architectures with diverse photonic tensor core designs; (2) optics-specific dataflow modeling with unique multi-dimensional parallelism and reuse beyond spatial/temporal dimensions; (3) data-aware energy modeling with realistic device responses, layout-aware area estimation, link budget analysis, and bandwidth-adaptive memory modeling; and (4) seamless integration with model training framework for hardware/software co-simulation. By providing a unified, versatile, and high-fidelity simulation platform, SimPhony enables researchers to innovate and evaluate EPIC AI hardware across multiple domains, facilitating the next leap in emerging AI hardware. Our code is open-sourced at link[1].**

I. Introduction

Heterogeneous electronic-photonic integrated circuits (EPICs) are emerging platforms for high-performance artificial intelligence (AI) computing. Demonstrated optical neural networks (ONNs) showcase breakthroughs in their performance and efficiency [1]–[9]. There is extensive research progress in materials, devices, and circuits aiming to overcome technological barriers in photonic AI. Some cross-layer co-design [10]–[14] and architecture optimization [4], [5], [15]–[19] efforts have scaled these systems for real-world AI tasks. Exploring the full stack of EPIC AI systems demands expertise across physics, devices, analog circuits, system architecture, electronic-photonic design automation (EPDA), and AI algorithms, as system-level evaluation is critical to understanding innovations at each individual design layer.

However, the complexity of such a hybrid system makes it challenging even for experts to grasp the behavior of each component and its interactions across software and hardware. Key challenges include: ❶ **Lack of Unified Representation of Distinct PTC Circuit Topology**: Photonic tensor cores (PTCs) employ diverse optical principles for matrix computation, e.g., magnitude/phase modulation, wavelength/mode/time-division multiplexing (WDM, MDM, TDM), interference, diffraction, etc. Such abundant design flexibility creates various PTC circuit topologies, e.g., weight bank [20], triangular mesh [1], [21], rectangular mesh [22], butterfly mesh [3], crossbar [2], [4], [17], single WDM link [23], etc. Prior simulators are modified from digital tools [24] that only support array-like computing architectures and cannot represent diverse PTC topologies. ❷ **Lack of Support for Optics-Specific Dataflow and Parallelism**: EPIC AI systems involve parallelism and resource sharing beyond temporal/spatial dimensions. The combination of optical broadcasting, hierarchical accumulation, and multi-dimensional reuse patterns results in a complex dataflow. The additional optical dimensions, like magnitude/phase, wavelength, polarization, and modes, further complicate the design space. Thus, a flexible framework tailored to optics-specific needs is required. ❸ **Lack of Accurate Model-Circuit-Layout Co-Modeling**: Unlike digital AI accelerators, analog systems integrate models tightly with devices/circuits, leading to inaccuracies in energy modeling due to a lack of real workload data and precise device settings. Beyond simple analytical models, there is a strong need to incorporate rigorous simulations and even chip measurements into energy analysis. Additionally, existing EPIC design tools overlook the layout when estimating the chip area. Previous methods [4], [25], [26] aggregate device footprints, leading to an underestimate of area. Therefore, a fast yet accurate layout estimator is essential for more reliable area analysis.

In this work, we present SimPhony, an open-source cross-layer modeling and simulation framework for heterogeneous EPIC AI systems. Built with a customized EPIC device library, SimPhony enables the hierarchical construction of heterogeneous photonic architectures from arbitrary PTC circuit topologies. Integrated with the ONN training library, SimPhony enables end-to-end simulation, including workload extraction, memory construction, and dataflow generation, while accurately analyzing system latency, data-aware energy, and layout-aware area.

Our main contributions are as follows,

- We introduce SimPhony, a cross-layer simulation framework for EPIC AI systems, enabling end-to-end accurate performance and efficiency analysis.
- **Unified PTC Representation**: Utilizing our customized device library, we design a hierarchical netlist-based circuit representation that generates arbitrary PTC topologies with automatically derived scaling rules and critical paths.
- **Photonics-Specific Dataflow Handling**: SimPhony accommodates multi-dimensional parallelism and hierarchical accumulation, effectively integrating the unique characteristics and dataflow of photonic computing hardware.

[1]https://github.com/ScopeX-ASU/SimPhony

TABLE I: PTC taxonomy [8]: PTC designs with distinct properties in operand range and reconfiguration frequency (*Reconfig*) with different # of forward required to obtain full-range output.

EPIC Designs	Operand A		Operand B		Forward Method	#Forwards
	Range	Reconfig	Range	Reconfig		
MZI Array [1]	$\mathbb{R}$	Dynamic	$\mathbb{R}$	Static	Direct	1
Butterfly Mesh [10]	$\mathbb{R}$	Dynamic	$\mathbb{C}$	Static	Pos-Neg	1
MRR Array [20]	$\mathbb{R}^+$	Dynamic	$\mathbb{R}$	Dynamic	Direct	2
PCM crossbar [27]	$\mathbb{R}^+$	Dynamic	$\mathbb{R}^+$	Static	Direct	4
TeMPO [17]	$\mathbb{R}$	Dynamic	$\mathbb{R}$	Dynamic	Direct	1

- **Accurate System Modeling**: `SimPhony` provides accurate energy analysis based on real workload data and realistic device power models, along with accurate area analysis featuring auto-generated layout estimation.

II. Preliminary

A. Photonic Tensor Core Taxonomy and Modeling Challenges

The diverse properties of PTC designs result in variations in circuit topology, devices, and operational principles, affecting speed and power. Table I categorizes PTCs by expressivity, computing mechanism, operand range, and reconfiguration speed [8]. For expressivity, universal PTCs can perform arbitrary matrix multiplication, e.g., micro-ring (MRR) arrays [20], while subspace PTCs support only a subset of static linear transformation [3]. Based on the numerical range of input operands, full-range PTCs compute arbitrary values in one shot, whereas subspace coherent PTCs (e.g., Butterfly meshes) require two differential computations for full-range input/output. Incoherent PTCs, e.g., MRR arrays [20], also require two computations to handle full-range inputs, while non-volatile phase change material (PCM) crossbar arrays [2] need up to four cycles. Reconfiguration speed is another key factor to determine its supported dataflow and operations. Thermo-optic MZI arrays require matrix decomposition and thermal tuning to reconfigure the weights, leading to a delay of μs to ms and limiting them to weight-stationary dataflows, unsuitable for dynamic self-attention. Dynamic PTCs [4], [17] use high-speed modulators for real-time matrix switching, enabling dynamic tensor products and output-static dataflows. Developing a modeling framework that accurately models complex system performance is challenging.

PTCs also feature varied topologies beyond conventional crossbars, such as triangular [21] and butterfly meshes [3]. Architectural features like optical broadcast, multi-dimensional sharing [4], analog-domain accumulation, and electrical-optical (E-O) interfaces further complicate system description and accurate modeling.

B. Photonic Accelerator Simulation Tools

Prior photonic AI hardware often relies on internal closed-source analytical device modeling and behavior-level modeling [3], [15], [16], [28], [29], unaware of the value in the actual workload. CimLoop is a recent analog NN simulator; its photonic version [24] has showcased one architecture Albireo [5], but it focuses on architectural parameters (e.g., memory and dataflow), and lacks flexible PTC construction with photonics-specific dataflow and parallelism. Its simplified device modeling and the complicated interface of the Timeloop-based framework make it challenging for device and circuit researchers to explore hardware optimization efficiently. Furthermore, previous efforts have not integrated ONN training at the application level, leaving no unified framework for *device/circuit-centric* system modeling, which is critical for EPIC AI efficiency and performance modeling.

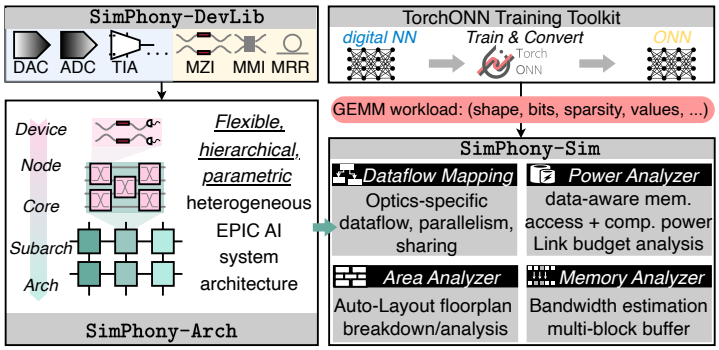


Fig. 1: Overview of our proposed `SimPhony` framework.

III. SimPhony: EPIC AI System Modeling and Simulation Framework

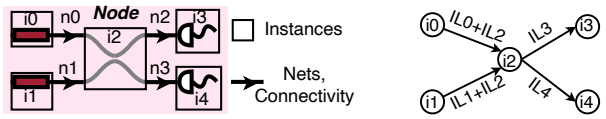
Figure 1 summarizes the simulation flow and key components of our proposed `SimPhony` framework. It contains a customized electronic-photonic device library `SimPhony-DevLib` and supports flexible, hierarchical, and parametric modeling of EPIC AI system architecture `SimPhony-Arch`. Integrated with `TorchONN` model training toolkit, our simulation system `SimPhony-Sim` handles photonics-specific dataflow with bandwidth-adaptive memory hierarchy, data-aware power estimation, link budget analysis, and layout-aware chip area analysis.

A. `SimPhony-DevLib`: Comprehensive and Customizable Electronic-Photonic Device Library

One key novelty of our simulation framework is the support of a comprehensive and customizable device library `SimPhony-DevLib`, which lays the foundation for flexible architecture construction and accurate system modeling.

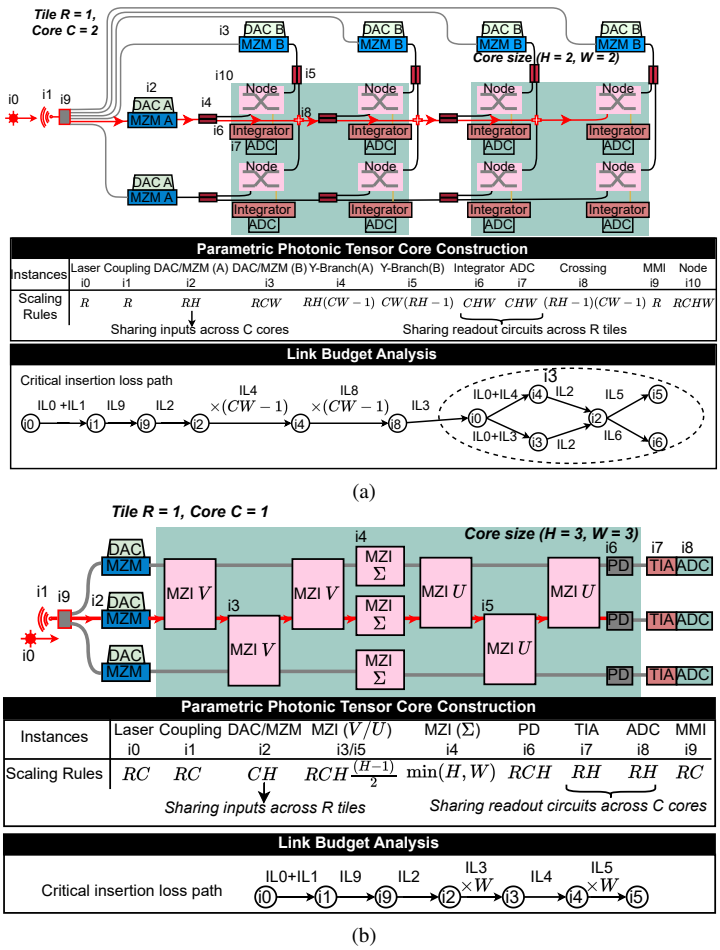
Modeling Granularity: `SimPhony-DevLib` device modeling is based on experimental data reported, ensuring an accurate representation of device characteristics. Specifically, the photonic device power models are obtained through Lumerical HEAT simulation or experimental measurements that are aware of actual device configuration, ensuring fast and accurate cost modeling.

Modeling Approach: We organize key components into electrical and optical categories, including a variety of high-performance photonic and electronic devices. Comprehensive device information is provided in detail to support accurate simulations of area, power, latency, and link budget. Devices from foundry PDKs can be plugged in. Our device library supports flexible scaling with different technology nodes, port numbers, working conditions, and more. For example, digital-to-analog converters (DACs) in our library support power scaling with customized sampling rates and bit resolutions, enabling power optimization via gating or quantization. On the photonic device front, e.g., the electro-optic Mach-Zehnder modulator (MZM) is widely used for high-speed encoding. To achieve precise performance modeling, we collect various properties such as spatial size, bandwidth, insertion loss, modulation efficiency, static power, extinction ratio, testing bitwidth, and more.



(a) (b)

Fig. 2: (a) Node-level circuit topology definition and (b) corresponding weighted direct acyclic graph (DAG) representation.



(a)

(b)

Fig. 3: Case studies of constructing parametric multi-core accelerator architecture (a) dynamic array-style TeMPO [17] and (b) static mesh-style MZI arrays [1]. The circuits can be generated by scaling up the smallest node with a user-defined scaling rule. The critical path with the highest insertion loss can be auto-derived.

B. SimPhony-Arch: Hierarchical, Parametric Heterogeneous EPIC AI System Architecture Builder

To enable flexible construction of heterogeneous multi-core architectures with diverse PTC designs and dataflows, we introduce SimPhony-Arch, a hierarchical, parametric architecture builder. Existing simulators focus on dataflow-centric architectural modeling with fixed PTCs, neglecting device/circuit details. This limits their usage for device/circuit customization, underscoring the need for a universal representation to unify diverse PTC designs.

To enable flexible PTC construction, we customize a *netlist* representation in SimPhony-Arch to describe devices as instances and port connectivity as directed 2-pin nets. Unlike electrical circuit netlists with undirected multi-pin nets, PTCs require directed 2-pin nets to capture the directional optical signal flow.

Key observations of PTC design patterns inspire us to use *modular circuit construction*, avoiding manual scripting of large netlists. This allows us to define *a minimal building block* denoted as *node*, e.g., a dot-product unit shown in Fig. 2(a), and *build the circuit according to specific scaling rules* with a user-defined node connection topology. A weighted directed acyclic graph (DAG) is generated based on the node topology, as shown in Fig. 2(b). The topology and insertion loss-based edge weights are essential in link budget analysis and layout-aware area estimation. This universal, hierarchical netlist interface also enables potential SPICE simulation and physical design as a future extension.

In the following, we provide two case studies of how to construct representative PTC architectures in a parametric style.

Case Study 1: Dynamic Array-style Tensor Cores TeMPO [4], [17]. The first case study demonstrates how to construct an array-style PTC architecture, TeMPO [4], [17], designed for dynamic time-multiplexed tensor products, as shown in Fig. 3(a). We first define the *architecture parameters*: R tiles, each containing C cores, with $H \times W$ dot-product nodes per core performing parallel computations. Then, we define the structure of the minimum building block, i.e., the dot-product unit, denoted as a *node*. A *node netlist* is used to describe the 6-device circuit topology using *directed* 2-pin nets to represent the waveguide connections and signal flow, shown in Fig. 2(a). To efficiently span the multi-core architecture without manually detailing every connection, we define *scaling rules* applied to each node and describe inter-node connections. This approach supports **parametric generation** of the architecture, enabling **automatic analysis** of area, power, and link budget. For example, the device area/power estimation engine will trace the netlist to count the number of devices considering *hardware sharing*. There are $RCHW$ total nodes for parallel dot-product. As the output of C cores in a tile are *in-situ* accumulated, integrators/ADCs can be shared and thus scaled by CHW. MZM group A encodes one matrix operand and can be broadcast to R tiles. Thus, the input encoders, i.e., DAC A and MZM A, are scaled by RH. These scaling rules are expressed as *customizable symbolic expressions* in circuit description files, enabling user-defined reuse styles to suit specific designs. The link budget analyzer will trace the netlist and auto-extract the longest path from the weighted DAG of the hierarchical netlist. Edge weights are automatically assigned to the insertion loss of incident vertices to match the parametric architecture settings, e.g., edge pointing to $i4$ stores $(CW-1)\times$ the loss of device $i4$.

Case Study 2: Static Mesh-style MZI Array [1], [22]. Besides array style, SimPhony-Arch can handle *more challenging mesh-style* PTCs, e.g., Clements-style MZI meshes in Fig. 3(b). The matrix is encoded by singular value decomposition ($U\Sigma V$), where unitary matrices U/V are decomposed to interleaved MZI meshes. To parametrically generate MZI meshes, we follow the *node* description to define a single-MZI node-U, node-Σ, and node-V. Our flexible *scaling rule* allows users to independently build unitary matrices by scaling node-U/V by $RCH(H-1)/2$ times and build diagonal by scaling node-Σ by $RC\min(H,W)$ times, which is *not representable by prior simulators that are based on arrays*. Details of the critical path and hardware sharing are omitted here, as they follow similar principles to Case Study 1.

C. SimPhony-Sim: EPIC AI System Simulation Flow

SimPhony-Sim is an end-to-end simulation flow, including NN model conversion and workload extraction to memory simulation and analysis of latency, area, and energy cost.

1) ONN Model Conversion and Workload Extraction: For digital DNN accelerator simulation, model training and hardware mapping are sufficiently decoupled. In contrast, analog mixed-signal EPIC AI systems require cross-layer co-design, resulting in closely coupled model training, conversion, mapping, and architecture simulation processes. Our SimPhony-Sim system can seamlessly interface with an open-source ONN training library, TorchONN [30], that supports various customized ONN types. A digital DNN will first be **converted to its analog optical version with layer-wise conversion**, e.g., Conv2d to TeMPOConv2d. The converted model will be trained with device non-ideality, quantization, pruning, and various co-design methodologies to ensure accuracy, robustness, and energy efficiency. The converted ONN model will be parsed by SimPhony-Sim to extract the detailed workload configuration for each layer, including input/weight size, input/weight/output bitwidth, pruning mask, scaling factors, **actual weight values**, etc. Weight values can have different modes, e.g., matrix values, normalized device transmissions, phase shifts, or even control voltages, which are useful for **precise value-aware power modeling**. Convolution, linear, and attention layers will be converted to general matrix multiplication (GEMM) representations. Other less computation-intensive layers are offloaded to electrical processors and omitted here for simplicity. With a layer-to-arch mapping configuration, we enable the flexibility to **map different layers to different types of sub-architectures** based on their compatibility and efficiency considerations, enabling heterogeneous computing paradigms.

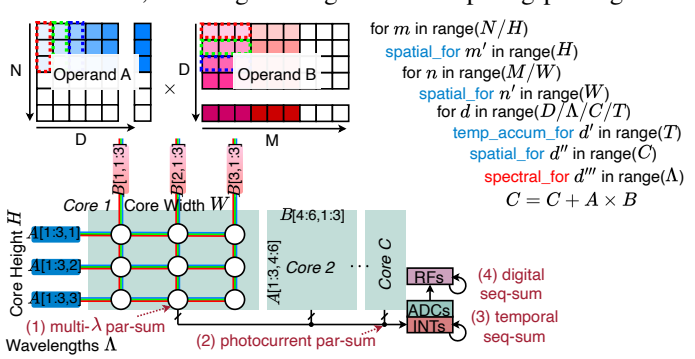


Fig. 4: Mapping blocking GEMM to output-stationary PTC with spatial/spectral parallelism and hierarchical accumulation.

2) Photonics-Specific Dataflow and Latency Analysis: Besides the support for standard dataflow for GEMM, e.g., weight/input/output stationary, here, we emphasize unique photonic-specific mapping and parallelism in SimPhony-Sim.

Multi-Dimension Parallelism and Hierarchical Accumulation. Beyond the spatial and temporal dimensions of electrical hardware, optical systems have more physical dimensions for parallel computing and hardware sharing, e.g., spectral, polarization, modes, etc. Figure 4 illustrates a partitioned GEMM workload mapped to a multi-core TeMPO architecture with multi-dimensional parallelism and hierarchical accumulation. As shown in the nested loop representation, multiple wavelengths are used for spectral parallel summation, photocurrents from C cores are aggregated for analog-domain parallel summation, spatial parallelism is used for parallel outer product, temporal integration is used for analog sequential summation, and the partial sum is further sequentially accumulated digitally in the local buffer. Based on the mapping and parallelism, we will *derive the system execution latency in units of cycles*.

Latency Penalty for Range-Restricted PTCs. As highlighted in Section II and Table I, PTCs are constrained in their *numerical range representation due to device modulation features*, leading to *varying processing times to complete full-range computations*. We denote the number of iterations to acquire full-range output as I. Especially for PTCs that can only encode positive inputs/weights, four times the execution cost is required to realize full-range output, i.e., I=4. SimPhony will automatically analyze the tensor core property based on input/weight/output encoding properties and generate the corresponding dataflow with $I\times$ latency penalty.

PTC Reconfiguration Latency Penalty. In weight-stationary PTCs, loading new weights is sometimes bottlenecked by slow device reprogramming rather than memory loading. For instance, thermo-optic (TO) devices have a thermal time constant of ~10 µs, and writing to phase change material (PCM) cells incurs a delay of over 100 ns. SimPhony-Sim automatically analyzes reprogramming latency and **applies corresponding cycle penalty whenever weight loading causes circuit reconfiguration delays exceeding one clock cycle**, e.g., 500 cycles per switch for 100 ns reconfiguration delay at 5 GHz. The total latency of mapping one layer is $\tau_{tot} = \tau_{\text{load-input/weight}} + \tau_{\text{write-out}} + I(\tau_{\text{comp}} + \tau_{\text{reconfig}})$.

3) Bandwidth-Adaptive Memory Hierarchy Modeling: One of the key points to enabling photonics advantage in AI computing is *sufficient data movement bandwidth and latency-hiding* techniques. Note that we focus on memory bandwidth analysis and assume on-chip and cross-chiplet interconnects provide sufficient data transaction bandwidth, especially when optical interconnects are available for multi-Tbps signal fanout/broadcast bandwidth [31]. SimPhony-Arch adopts a four-level memory hierarchy consisting of off-chip High Bandwidth Memory (HBM), Global Buffer (GLB), Local Buffer (LB), and Register File (RF). Each memory level stores operands A, B, and the output in progressively smaller sizes:i.e., the entire model at the HBM level, a single layer at the GLB level, the processing matrix dimensions at the LB level, and data for a single cycle at the RF level. The bandwidth of LB (BW_{LB}) and RF (BW_{RF}) must accommodate the architecture's single-cycle processing throughput, calculated as $BW_{LB}, BW_{RF} \geq BytesPerCycle \times f$, where f is the PTC operating clock frequency. GLB's bandwidth (BW_{GLB}) is calculated as $BW_{GLB} = MaxLayerSize \cdot f/(N_p \cdot D_P \cdot M_p)$, where the partitioned matrix dimensions are N_p, D_p, and M_p. We first profile the **maximum bandwidth requirement** ($\widehat{BW}$) **for all sub-architectures** based on the GLB demand per cycle, considering data sharing and broadcast in a specific dataflow. To enable full utilization of the computing cores *without memory bottleneck*, we adopt a SoTA **multi-block SRAM design to meet the bandwidth demand** — A multi-block GLB can boost its bandwidth proportional to the number of blocks [4], [17].

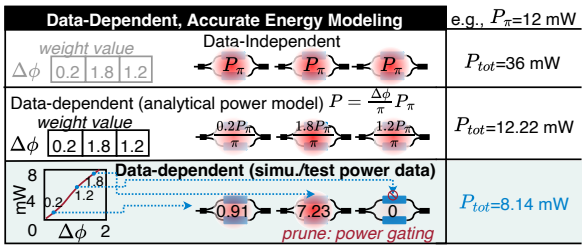


Fig. 5: SimPhony-Sim supports accurate, data-dependent energy modeling with simulated/measured device power data.

SimPhony-Sim automatically searches for the minimum number of required GLB blocks, # of Blocks $= \tau_{GLB} \cdot \widehat{BW}/(b_{\text{bus}} \cdot 8)$, where τ_{GLB} is the fastest cycle simulated by CACTI [32], and b_{bus} is the buswidth in bits.

4) Link Budget Analysis: Link budget analysis is important for photonic systems to profile the critical-path insertion loss and derive the laser source power requirement and optical signal-to-noise ratio (SNR). To obtain the IL on the critical path, we leverage our constructed hierarchical, weighted DAG representation of the architecture in Fig. 3(a), 3(b). From a given source node, i.e., laser, to a destination node, i.e., photodetector (PD), we will use the graph's longest path to get the critical path IL. Given the PD sensitivity S, to differentiate b_{in}-bit input levels with a target bit error rate, we can derive the lowest laser power required [3], [4],

$$P_{laser} = \frac{10^{(S+IL)/10} \cdot 2^{b_{in}}}{\eta_{WPE}} \cdot \frac{1}{1 - 0.1^{ER/10}}, \quad (1)$$

where η_{WPE} is the laser wall plug efficiency. Non-ideal modulation extinction ratio ER is also considered a power penalty to recover the full modulation range [20].

5) Data-Dependent Device-Response-Aware Energy Analysis: To accurately model the energy of EPIC AI systems, SimPhony-Sim captures data access energy via CACTI-simulated memory energy and computing energy based on *actual operand values and realistic device power modeling*.

For data access cost, we derive the memory access size (D_{mem}) for off-chip HBM, GLB, LB, and RF based on the dataflow analysis and accumulate the energy cost with CACTI-simulated access energy per bit e_{mem}, i.e., $E_{mem} = \sum_{mem \sim \{HBM,GLB,LB,RF\}} e_{mem} D_{mem}$.

For computing cost, SimPhony-Sim supports **data-dependent** energy analysis with **accurate device power modeling**, shown in Fig. 5. For analog hardware, the encoded data determines the device configuration, significantly impacting its power. Hence, SimPhony **accumulates the energy over cycles based on the values of the real operands**. This approach enables accurate energy profiling with *fine-grained power gating from ONN pruning* [14]. Device power modeling is also critical. Default library power references, like P_π for phase shifters (PSs), often overestimate actual power, while analytical models can be overly ideal. As shown in Fig. 5, SimPhony-Sim supports **customized power models** using analytical, simulation, or chip testing data for power estimation with different fidelity.

6) Layout-Aware Chip Area Analysis: The chip area is crucial for fabrication and packaging costs and guiding design optimization. SimPhony-Sim supports fast, realistic layout-aware area estimation. Unlike previous methods that simply sum all

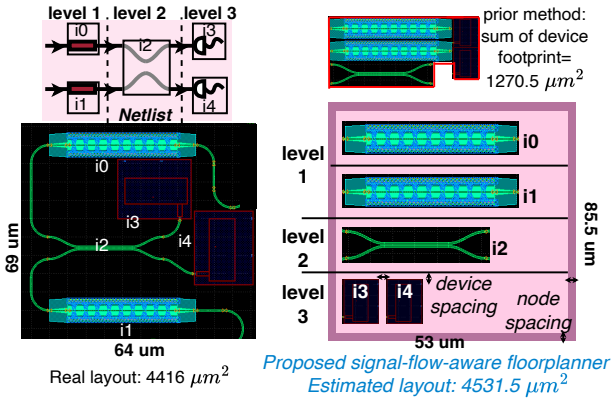
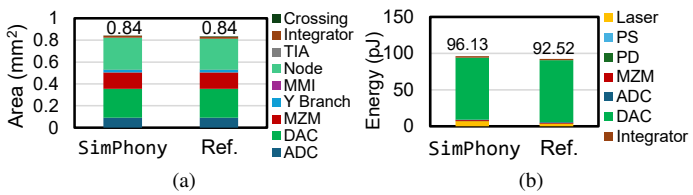


Fig. 6: Proposed signal flow-aware row-based floorplan for accurate layout-aware area estimation.



(a) (b)

Fig. 7: SimPhony results validated on a (280×28)×(28×280) GEMM task w/ TeMPO [17]. (a) Area and (b) energy breakdown.

device footprints, SimPhony-Sim either takes in a user-defined bounding box or **automatically generates a signal-flow-aware floorplan**, as shown in Fig. 6. SimPhony sets the placement site width to fit the longest device and attempts to *hide* other devices beneath it. The **floorplan follows the device's topological order** from the netlist to adhere to the minimum bending rule in PIC placement, accounting for user-defined device/node spacing. This approach closely matches the real layout area and can be potentially extended to interface with PIC placement tools.

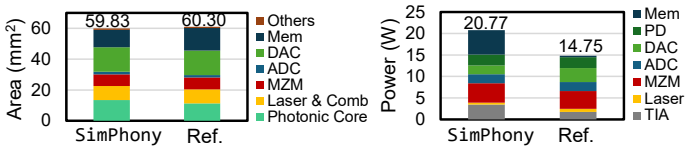
IV. Evaluation Results

A. SimPhony Validation

We validate our simulation results by comparing them with architectural evaluation results reported in previous work. It is important to clarify that, in the absence of system-level experimental demonstrations of multi-core photonic AI chips, we compare our results with prior architecture simulation results. We re-emphasize that individual *component data used are backed by experimental measurements*, and area estimates are based on real chip layouts. The functionality and accuracy of the ONN model are ensured by the TorchONN training framework, which is beyond the scope of our architecture performance modeling tool.

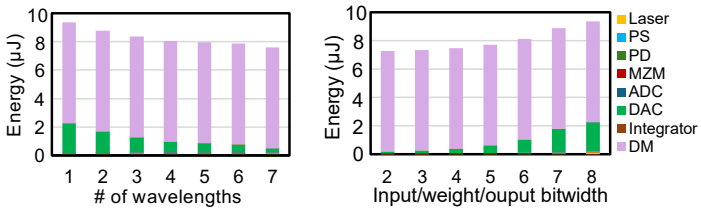
GEMM Workloads: To validate the simulation results of SimPhony for GEMM, we compare the simulated area and energy metrics on a (280×28)×(28×280) GEMM task against the reference values reported in the original TeMPO paper [17] in Fig. 7 with the following architecture settings. We set the core width and height to 4 and the number of tiles and cores per tile to 2. The area 7(a) and energy 7(b) breakdown from SimPhony match the reference results from TeMPO paper.

Dynamic Transformer Workloads: For Transformers with self-attention operations, we validate SimPhony's area and energy results against Lightening-Transformer [4] (LT), shown in Fig. 8.



(a) (b)

Fig. 8: SimPhony results validated on BERT-Base(ImageNet) w/ Lightening-Transformer [4]. (a) Area and (b) power breakdown.



(a) (b)

Fig. 9: Sweep (a) # of wavelengths (b) bitwidth on TeMPO arch [17] and (280×28)×(28×280) GEMM. *DM*: data movement.

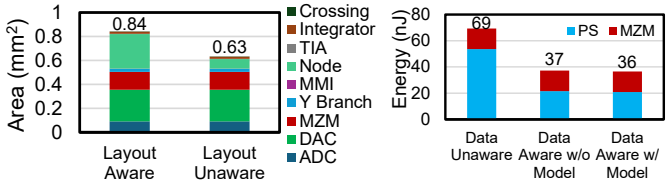
we simulate BERT-Base [33] with a single (224×224) ImageNet-1K [34] image. We adopt LT's settings: 4 tiles, 2 cores per tile, each core sized 12×12, with 12 wavelengths operating at 5 GHz. Since LT provides only power breakdowns, we report power instead of energy. SimPhony accurately reproduces the chip area when appropriate scaling factors are applied, matching LT's reported values. Minor deviations in photonic core and memory areas stem from differences in core area calculations, memory simulations, and device spacing. Our layout-aware estimator effectively generates realistic core areas. Device power aligns with LT's breakdown except for memory, where deviations result from different technology nodes used in memory simulation (SimPhony uses CACTI-45 nm [32]; LT uses PCACTI-14 nm [35]) and SRAM port/bus differences.

B. SimPhony Use Cases

To show the capability of SimPhony, we study multiple design examples by sweeping PTC architectural parameters in SimPhony to gain design insights and demonstrate their impacts on system performance and efficiency. All simulations discussed below have a 4×4 core size with 2 tiles and 2 cores per tile.

1) Multi-Wavelength Parallelism: As discussed in Section III-C2, optical systems allow parallelism beyond spatial and temporal dimensions. However, data encoding in these dimensions requires extra power. Figure 9(a) examines TeMPO [17] under varying wavelength settings on a (280×28)×(28×280) GEMM task while scaling MZM and laser sources with the number of wavelengths. Increased wavelengths enhance parallelism, speeding up computation and reducing energy for components that do not scale with wavelength. However, the MZM's energy remains constant as the number of MZMs scale with # of wavelengths.

2) Bitwidth Representation vs. Energy Consumption: To investigate how the ADC/DAC bit precision impacts the system power, in Fig. 9(b), we sweep the energy for different tensor bitwidths. The results show a clear trend of increasing energy with higher bits. Users can leverage the bitwidth simulation results to find the sweet spot for optimal efficiency.



(a) (b)

Fig. 10: (a) TeMPO [17] area breakdown with layout awareness. (b) SCATTER [14] energy breakdown with data awareness.

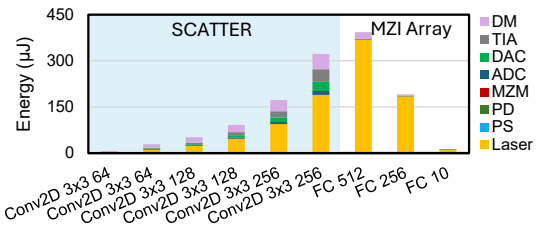


Fig. 11: Layer energy breakdown with heterogeneous layer mapping of VGG-8(CIFAR10) [36]. Convolutions are mapped to SCATTER [14], and Linear layers are mapped to MZI meshes [1].

3) Layout-Aware and Data-Dependent Modeling: The exploration of layout-aware area estimation and data-dependent modeling is shown in Fig. 10. The layout-unaware method underestimates the node area by 72%, while our floorplan estimation enables accurate area estimation compared to the real layout. In data-awareness evaluation, we focus on a weight-static PTC SCATTER [14] where weight values impact the phase shifter's power. By counting PS power based on real weight values, the PS energy decreases from 0.0537 μJ to 0.0215 μJ with an approximate power model. If a rigorous device power model is used, the energy is further reduced to 0.0209 μJ with a substantial 60% reduction.

4) Heterogeneous Mapping: Lastly, we show SimPhony's capability for heterogeneous architecture modeling with layer-to-sub-architecture mapping in Fig. 11. The convolutional layers of VGG-8 are mapped to SCATTER [14], while the linear layers are mapped to MZI meshes [1], while two sub-architectures share the same on-chip memory hierarchy. This hybrid architecture modeling showcases SimPhony's flexibility in integrating hybrid systems and enabling fine-grained workload-to-hardware mapping. As a future extension, SimPhony can be extended to enable automated design space exploration that combines the strengths of different photonic computing architectures in heterogeneous systems for diverse AI workloads.

V. Conclusion

This paper presents SimPhony, a cross-layer modeling and simulation framework for EPIC AI hardware, enabling photonic-specific design evaluation and fair comparisons across implementations. By emphasizing accurate device/circuit-level modeling with generic and extensible representations, SimPhony bridges hardware and software stacks to support flexible hardware construction, validation, and architecture exploration with multi-dimensional metric trade-offs. We have validated SimPhony's accuracy against prior work and will continue expanding its capabilities to help researchers uncover insights and drive innovation in high-performance, energy-efficient photonic computing systems.

REFERENCES

[1] Y. Shen, N. C. Harris, S. Skirlo *et al.*, "Deep learning with coherent nanophotonic circuits," *Nature Photonics*, 2017.

[2] J. Feldmann, N. Youngblood, M. Karpov, H. Gehring, X. Li, M. Stappers, M. L. Gallo, X. Fu, A. Lukashchuk, A. Raja, J. Liu, D. Wright, A. Sebastian, T. Kippenberg, W. Pernice, and H. Bhaskaran, "Parallel convolutional processing using an integrated photonic tensor core," *Nature*, 2021.

[3] C. Feng, J. Gu, H. Zhu, Z. Ying, Z. Zhao *et al.*, "A compact butterfly-style silicon photonic–electronic neural chip for hardware-efficient deep learning," *ACS Photonics*, vol. 9, no. 12, pp. 3906–3916, 2022.

[4] H. Zhu, J. Gu, H. Wang, Z. Jiang, Z. Zhang, R. Tang, C. Feng, S. Han *et al.*, "Lightening-transformer: A dynamically-operated photonic tensor core for energy-efficient transformer accelerator," in *Proc. HPCA*, 2024.

[5] K. Shiflett, A. Karanth, R. Bunescu, and A. Louri, "Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics," in *Proc. ISCA*, 2021, pp. 860–873.

[6] B. J. Shastri, A. N. Tait *et al.*, "Photonics for Artificial Intelligence and Neuromorphic Computing," *Nature Photonics*, 2021.

[7] Z. Xu, T. Zhou, M. Ma, C. Deng, Q. Dai, and L. Fang, "Large-scale photonic chiplet taichi empowers 160-tops/w artificial general intelligence," *Science*, vol. 384, no. 6692, pp. 202–209, 2024.

[8] S. Ning, H. Zhu, C. Feng, J. Gu, Z. Jiang, Z. Ying, J. Midkiff, S. Jain, M. H. Hlaing, D. Z. Pan, and R. T. Chen, "Photonic-electronic integrated circuits for high-performance computing and ai accelerators," *Journal of Lightwave Technology*, pp. 1–26, 2024.

[9] A. R. Totović, G. Dabos, N. Passalis, A. Tefas, and N. Pleros, "Femtojoule per MAC Neuromorphic Photonics: An Energy and Technology Roadmap," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 26, no. 5, pp. 1–15, 2020.

[10] J. Gu, Z. Zhao, C. Feng *et al.*, "Towards Hardware-Efficient Optical Neural Networks: Beyond FFT Architecture via Joint Learnability," *IEEE TCAD*, 2020.

[11] J. Gu, Z. Zhao, C. Feng, W. Li, R. T. Chen, and D. Z. Pan, "FLOPS: Efficient On-Chip Learning for Optical Neural Networks Through Stochastic Zeroth-Order Optimization," in *Proc. DAC*, 2020.

[12] J. Gu, Z. Zhao, C. Feng, H. Zhu, R. T. Chen, and D. Z. Pan, "ROQ: A noise-aware quantization scheme towards robust optical neural networks with low-bit controls," in *Proc. DATE*, 2020.

[13] S. Banerjee, M. Nikdast, S. Pasricha, and K. Chakrabarty, "Pruning Coherent Integrated Photonic Neural Networks Using the Lottery Ticket Hypothesis," in *2022 IEEE Comput. Soc. Annu. Symp. VLSI ISVLSI*, Jul. 2022, pp. 128–133.

[14] Z. Yin, N. Gangi, M. Zhang, J. Zhang, R. Huang, and J. Gu, "Scatter: Algorithm-circuit co-sparse photonic accelerator with thermal-tolerant, power-efficient in-situ light redistribution," in *International Conference on Computer-Aided Design (ICCAD)*, 2024. [Online]. Available: https://arxiv.org/abs/2407.05510

[15] W. Liu, W. Liu, Y. Ye, Q. Lou, Y. Xie, and L. Jiang, "Holylight: A nanophotonic accelerator for deep learning in data centers," in *Proc. DATE*, 2019.

[16] F. Zokaee, Q. Lou, N. Youngblood *et al.*, "LightBulb: A Photonic-Nonvolatile-Memory-based Accelerator for Binarized Convolutional Neural Networks," in *Proc. DATE*, 2020.

[17] M. Zhang, D. Yin, N. Gangi, A. Begović, A. Chen, Z. R. Huang, and J. Gu, "Tempo: Efficient time-multiplexed dynamic photonic tensor core for edge ai with compact slow-light electro-optic modulator," *J. Appl. Phys*, 2024.

[18] K. Shiflett, D. Wright, A. Karanth, and A. Louri, "PIXEL: Photonic Neural Network Accelerator," in *Proc. HPCA*, 2020, pp. 474–487.

[19] C. Demirkiran, G. Yang, D. Bunandar, and A. Joshi, "Mirage: An rns-based photonic accelerator for dnn training," in *Proc. ISCA*, 2024, pp. 73–87.

[20] A. N. Tait, T. F. de Lima, E. Zhou *et al.*, "Neuromorphic photonic networks using silicon photonic weight banks," *Sci. Rep.*, 2017.

[21] M. Reck, A. Zeilinger, H. Bernstein *et al.*, "Experimental realization of any discrete unitary operator," *Physical review letters*, 1994.

[22] W. R. Clements, P. C. Humphreys, B. J. Metcalf *et al.*, "Optimal Design for Universal Multiport Interferometers," *Optica*, 2018.

[23] X. Xu, M. Tan, B. Corcoran, J. Wu, A. Boes, T. G. Nguyen, S. T. Chu, B. E. Little, D. G. Hicks, R. Morandotti, A. Mitchell, and D. J. Moss, "11 TOPS photonic convolutional accelerator for optical neural networks," *Nature*, 2021.

[24] T. Andrulis, G. I. Chaudhry, V. M. Suriyakumar, J. S. Emer, and V. Sze, "Architecture-level modeling of photonic deep neural network accelerators," *arXiv preprint arXiv:2405.07266*, 2024.

[25] J. Gu, Z. Zhao, C. Feng *et al.*, "Towards area-efficient optical neural networks: an FFT-based architecture," in *Proc. ASPDAC*, 2020.

[26] J. Gu, H. Zhu, C. Feng, Z. Jiang, R. T. Chen, and D. Z. Pan, "M3ICRO: Machine Learning-Enabled Compact Photonic Tensor Core based on PRogrammable Multi-Operand Multimode Interference," *APL Machine Learning*, 2023.

[27] M. Miscuglio and V. J. Sorger, "Photonic Tensor Cores for Machine Learning," *Applied Physics Review*, 2020.

[28] J. Gu, H. Zhu, C. Feng, Z. Jiang, M. Liu, S. Zhang, R. T. Chen, and D. Z. Pan, "Adept: Automatic differentiable design of photonic tensor cores," in *Proc. DAC*, 2022.

[29] M. Li, Z. Yu, Y. Zhang, Y. Fu, and Y. Lin, "O-has: Optical hardware accelerator search for boosting both acceleration performance and development speed," in *Proc. ICCAD*, 2021.

[30] J. Gu, H. Zhu, C. Feng, Z. Jiang, R. T. Chen, and D. Z. Pan, "L2ight: Enabling On-Chip Learning for Optical Neural Networks via Efficient in-situ Subspace Optimization," in *Proc. NeurIPS*, 2021.

[31] M. Khani, M. Ghobadi, M. Alizadeh, Z. Zhu, M. Glick, K. Bergman, A. Vahdat, B. Klenk, and E. Ebrahimi, "Sip-ml: high-bandwidth optical network interconnects for machine learning training," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, ser. SIGCOMM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 657–675. [Online]. Available: https://doi.org/10.1145/3452296.3472900

[32] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, jun 2017.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009, pp. 248–255.

[35] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *2014 IEEE Computer Society Annual Symposium on VLSI*, 2014, pp. 290–295.

[36] N. Simon, J. Friedman, T. Hastie, and R. Tibshirani, "A sparse-group lasso," *Journal of Computational and Graphical Statistics*, 2013.

Precon: A Precision-Convertible Architecture for Accelerating Quantized Deep Learning Models across Various Domains Including LLMs

Jongwoo Park*, Hyeonseong Kim*, Jiyun Han, and Seungkyu Choi†
Kyung Hee University, Republic of Korea
{pmk0638, hsyoun2222, hjy0013, seungkc}@khu.ac.kr

***Abstract*—The sensitivity of LLMs to quantization has driven the development of hardware accelerators tailored for specific low-precision configurations such as weight-only quantization and mixed-precision, which can introduce inefficiencies in dedicated hardware architecture. In this work, we propose *Precon*, a precision-convertible architecture designed to accelerate various quantized deep learning models, particularly LLMs, through a unified processing unit. By enabling on-the-fly switching between half-float (FP16) decoding and integer (INT) decomposition, the design effectively supports INT4-FP16, INT4-INT4, and INT4-INT8 arithmetic within shared logic. *Precon* achieves up to 4.1× speedup and 81.4% reduction in energy consumption compared to the baseline across various domains, including the support of both accurate and efficient acceleration of quantized LLMs.**

***Index Terms*—large language models, quantization, systolic array, floating-point arithmetic, mixed-precision.**

I. Introduction

Deep learning models have shown exceptional performance across various AI domains, including computer vision [1], natural language processing (NLP) [2], robotics [3], and autonomous systems [4]. In particular, recent years have seen large language models (LLMs) achieve state-of-the-art performance in many NLP tasks. LLMs have become deeply integrated into daily life, accessible through applications such as text generation, chatbots, and machine translation [5]–[7]. However, the vast number of parameters in LLMs results in significant memory and computational demands, which limits their broader applicability [8].

One of the promising solutions to alleviate the addressed burden is data quantization [9], [10]. While deep learning models in various AI domains such as computer vision (e.g. image classification, object detection, super-resolution, etc.) or simple language processing tasks (e.g. task classification, question answering, multi-modal, etc.) can adopt ultra-low-precision computing using sub-8-bit integer (INT) formats [11]–[13], typical language processing tasks based on LLMs suffer from accuracy degradation in low-precision computing. Consequently, various research targeting post-training quantization (PTQ) of LLMs have been actively studied to support accurate low-precision LLMs.

Recent studies have shown that quantized LLMs face accuracy challenges due to the broad data distribution observed in activation tensors [14]. Consequently, weight-only quantization approaches, such as applying aggressive INT4 quantization to weights while keeping activations in half-precision (FP16), have become one of the mainstreams in LLM quantization [15]–[18]. Recent research also explores INT-only quantization methods employing mixed-precision techniques, such as 4-bit weights and 8-bit activations [19], [20], or aggressive full 4-bit quantization techniques [19], [21]. However, these methods require intensive calibration processes to reshape data distributions and often result in noticeable accuracy degradation, posing challenges for widespread adoption in practical applications.

Given this context, specialized processing unit designs have become essential on the hardware side to accelerate inference using quantized LLMs. A recent approach proposes an INT-FP arithmetic-based accelerator designed to support weight-only quantized LLM models [22]. It substitutes FP arithmetic with INT arithmetic in the processing array, allowing approximation through mantissa truncation within a shared exponent group. However, this approach lacks support for lower-precision INT-only arithmetic, restricting its applicability to a broader range of quantized models beyond weight-only quantization. On the other hand, there are several designs of precision-scalable processing units that are available to support mixed- or low-precision INT-based quantized models [23], [24]. While effective for INT-based computations, these designs lack support for high-precision FP arithmetic, limiting their utility for high-accuracy LLMs. Thus, flexible support of both INT and FP arithmetic is essential to fully accommodate quantized deep learning models, including reliable LLM serving.

In this work, we introduce *Precon*, a precision-convertible processing unit architecture designed to accelerate various quantized deep learning models, including LLMs. To support floating-point-based arithmetic required for reliable LLM inference, *Precon* incorporates on-the-fly FP16 to INT conversion within the processing unit, enabling both INT-INT and INT-FP arithmetic across multiple precision levels. This unified processing architecture supports low-precision inference with INT4-INT4/8 for error-resilient models—including LLMs, convolutional neural networks (CNNs), and transformer encoders—as well as high-precision inference with INT4-FP16 for high-accuracy LLMs. We evaluate the hardware efficiency of our systolic array architecture on various quantized models,

*J. Park and H. Kim contributed equally to this work.
†S. Choi is the corresponding author.

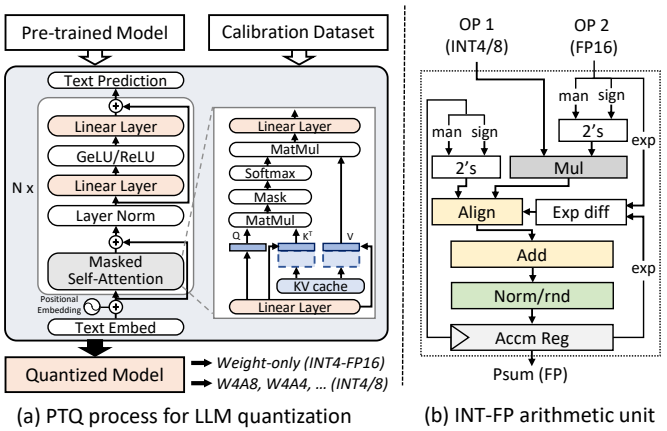


Fig. 1. (a) An overview of PTQ process for LLMs, (b) an INT-FP arithmetic unit for weight-only quantized LLMs.

demonstrating its effectiveness by comparing it to both the FP16 baseline and the state-of-the-art INT-FP accelerator [22].

Our contributions are summarized as follows.

- A decomposable FP16 encoding method is proposed to effectively decode the data into four INT4 bricks, enabling separate multiplication and addition with minimal logic overhead from precision conversion.
- We design a unified processing unit maximally leveraging the same logic components for computing different precision levels.
- We analyze the area and power efficiency of the processing unit design, followed by an evaluation of the speedup and energy efficiency of the systolic array, demonstrating improved hardware efficiency and versatility.

II. Background

A. Low-Precision Computing for LLMs

With constraints on training data and high computational demands, quantizing LLMs often relies on PTQ. Fig. 1(a) describes the overall PTQ process to obtain low-precision LLM serving. Given the extensive number of weight parameters, which dominates the memory usage, and the challenges in reducing activation bit-width posed by outliers, weight-only quantization is a commonly adopted approach in LLM quantization [15]–[18]. This approach quantizes only the weights to extremely low precision, such as INT4, while preserving high-precision computation with the activation data. It offers reduced memory consumption while maintaining near full-precision accuracy across various LLM tasks.

Recent research has shown that optimization techniques addressing outliers during calibration can enable low-precision INT arithmetic for LLM inference with acceptable accuracy. Various methods such as data redistribution through channel assembly [14], [19] and rotation-based redistribution [21], [25], have been proposed to mitigate the outliers observed in activation channels. These state-of-the-art techniques demonstrate that mixed-precision configuration of INT4 for weights and INT8 for activations yields reliable LLM performance with minimal accuracy drop. Furthermore, aggressive INT4-only quantization has been explored, accepting a certain accuracy drop. Despite the accuracy degradation, these recent attempts have triggered the possibility of INT-only arithmetic for LLM inference.

B. INT-FP Arithmetic for Quantized LLMs

LLM inference requires processing through a sequence of Transformer decoder blocks, with linear layers comprising the majority of the computational workload [22]. To optimize this workload, dedicated arithmetic units have been developed to accelerate the matrix multiplications in these linear layers, tailored to the precision configuration of the operands, namely weights and activations. While INT-only configurations can be efficiently managed with straightforward INT arithmetic, weight-only quantization introduces complexity by necessitating specialized arithmetic that handles INT-FP16 multiplication for the mixed-precision operands and FP-based accumulation.

The conventional approach to supporting weight-only quantization utilizes FP16 arithmetic with on-the-fly format conversion from INT to FP16. While this method optimizes memory usage, it fails to address computational inefficiencies. To fully leverage the advantages of quantized weights, a prototype of multiply-and-accumulate (MAC) unit specially designed for INT4-FP16 operations can be employed, as illustrated in Fig. 1(b). This unit directly accepts the INT operand and multiplies it with the mantissa of the FP16 operand, thereby significantly simplifying the FP-based multiplier. FP-based addition is then used to accumulate partial sums in FP16 format.

While such design can significantly reduce area and power consumption compared to the FP16-based design, it has a critical limitation in that it cannot fully leverage hardware efficiency for INT-INT low-precision computing. Given the need for INT-FP arithmetic to maintain accuracy in LLM inference, alongside INT-INT arithmetic support for models that allow further quantization, a reconfigurable MAC unit architecture is essential. This would enable effective handling of low-precision configurations commonly employed across diverse AI domains.

III. Precon Architecture

A. Convertible FP16 Data Format

The key requirement for supporting both INT and FP formats in the same processing engine is the ability to decompose an FP representation into uniform INT data bricks. If FP16 format is segmented into four 4-bit bricks, each capable of participating in INT-based arithmetic, efficient hardware-level switching between INT and FP arithmetic modes becomes feasible. Accordingly, we slightly modify the standard FP16 format, organizing it into one 4-bit brick for the exponent and three 4-bit mantissa bricks, including the sign bit.

Referring to the Bit Fusion technique [23], the decomposed bricks in a single data represent different bit positions, requiring the need for 4-bit shifts to align position for proper arithmetic operations. Similarly, FP addition involves a shift operation for alignment based on the exponent difference between operands. To streamline these processes, we encode the exponent as a

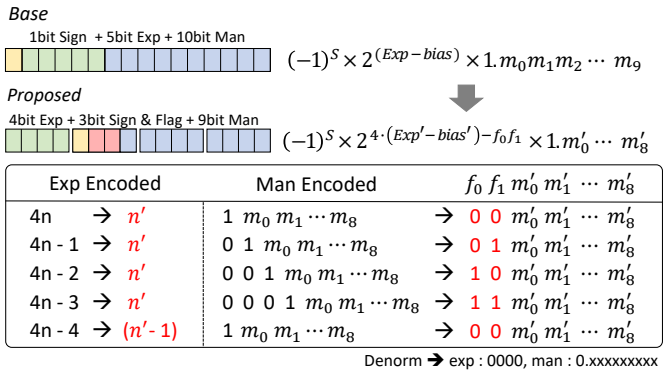


Fig. 2. The decomposable fourfold exponent-encoded FP16 format.

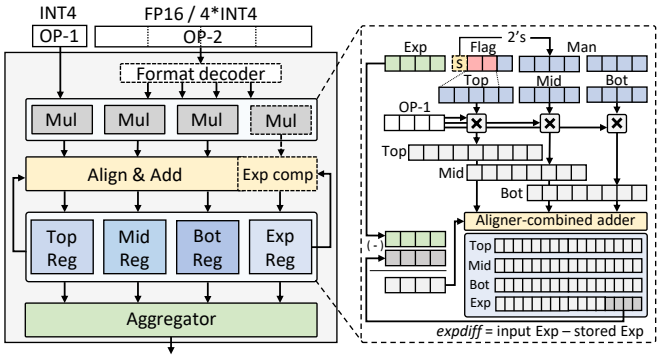


Fig. 3. The processing unit architecture with operation details.

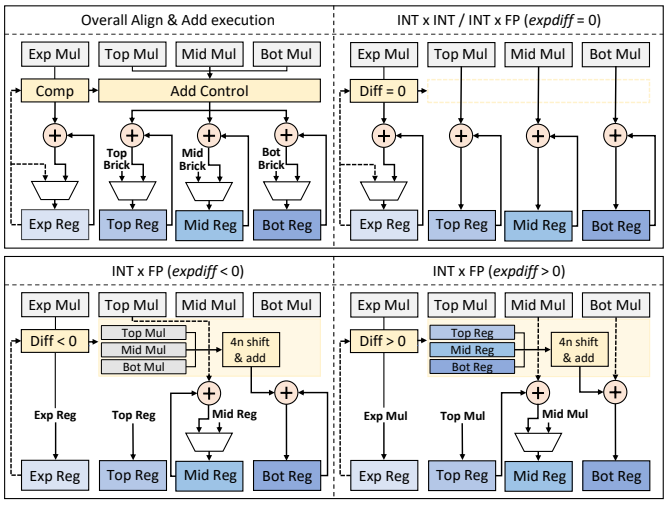


Fig. 4. The logical datapath of the proposed aligner-combined addition.

multiple of four, enabling the use of 4-bit shifters to perform alignment in both FP addition and bit-brick fusion efficiently.

Fig. 2 illustrates the proposed FP16 format, which employs a fourfold encoding for the exponent. Although this approach reduces the exponent field to 4 bits, it effectively offers an expanded range nearly twice as wide as the standard FP16's 5-bit exponent, demonstrating feasibility for a broader range of values. As each consecutive exponent value differs by a factor of four, the mantissa must represent intermediate values within the normalized range. Therefore, a leading-one must be embedded within the mantissa bits to represent the full normalized range. As shown in the left table of Fig. 2, the position of the leading-one varies based on the remainder of the exponent divided by four. This results in four possible cases, corresponding to remainders of 0 through 3, each of which is encoded using a 2-bit flag to compactly represent the leading-one position.

While a similar approach of FP16 decomposition has been studied [26], it fails to capture the leading-one value effectively inside the mantissa representation leading to a significant error produced by unavoidable mantissa truncation. On the other hand, the proposed 2-bit flag encoded inside the mantissa can effectively represent by only giving up a single bit in mantissa from the standard FP16 in every representation. Besides, the wider exponent range can even enhance the accuracy of LLM inference with possessing excessive data range [27].

B. Processing Unit Design

Given the proposed format as one of the MAC operands, we introduce a format-convertible processing unit that executes both INT4-FP16 and INT4-INT4×4 MAC operations by maximizing the shared use of computing logic. Fig. 3 presents a simplified block diagram of the MAC unit. The unit comprises four INT4 multipliers, an aligner-combined adder module, four distinct position registers, and an aggregator unit that collects data from each register to reformulate the output appropriately.

Multiplication After the data decomposition, all the input operands are represented as 4-bit bricks to engage in INT-based multiplication. Due to the encoded mantissa using the flag bits, a format decoder is required to produce three bricks, including those representing leading-one. In INT-INT mode, four separate multiplied data bricks are generated and accumulated in each register. In INT-FP mode, three multipliers handle the mantissa portion, producing three multiplied data bricks, while the other single brick conveys exponent data to its designated register.

Aligner-Combined Addition Precise control logic is implemented to add the components and store them in designated position registers. These registers, labeled as exponent, top, middle, and bottom, serve dual roles: acting as standard, independent accumulators in INT mode and as designated position accumulators required for FP16 reconstruction in the final step. As shown in the middle part of Fig. 3, the aligner needed for FP16 addition is integrated into the adder unit, facilitating both INT-based and FP16-based accumulation by aligning the candidates for addition to their appropriate positions. Specifically, all the data bricks including the multiplied results and the position registers participate as potential addends and are aligned according to the calculated exponent difference (*expdiff*), which serves as the control signal, ensuring accurate placement for addition.

Fig. 4 details the logical path of the proposed accumulation method, in which four adders perform up to four separate accumulations. The adder control unit supports flexible operand positioning, allowing multiplied results and register-stored values to align with their designated positions. In INT-INT mode, addition proceeds by directly accumulating multiplied results into the corresponding registers. For INT-FP mode, the

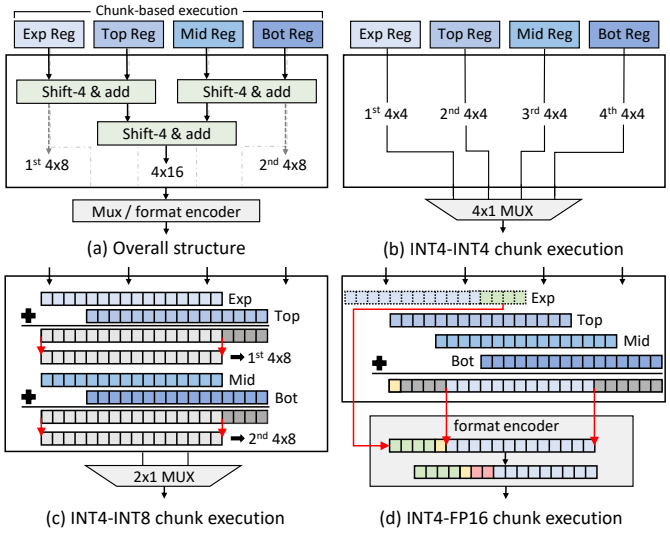


Fig. 5. Operation details of the precision-convertible aggregator.

operation branches into three cases based on the *expdiff* value between adjacent operands to be added.

When *expdiff* is zero, the operation mimics INT-INT mode but with the exponent register simply retaining the same value. For *expdiff* <0, it indicates that the register-stored values act as base values while incoming multiplied data must be aligned. Therefore, the multiplied results serve as addend bricks and are added into registers at a position lower by *expdiff* levels. Bricks unable to lower further are shifted by 4 bits and added to the lowest, which is the bottom register. Conversely, for *expdiff* >0, where the incoming data have a larger exponent, the multiplied results become the base values, and stored register values act as addends. The rest of the adding process follows the same rule as in the *expdiff* <0 case. Due to the quite congested logic within the addend controls, the critical path of our design lies here; however, it remains considerably shorter than that of unpipelined general FP16 addition, as will be discussed in Section IV-C.

Aggregation The position registers can accumulate values independently until the output is requested. At that point, the aggregator unit collects the stored values across all registers to formulate the output based on the operating mode. The detailed aggregation process is shown in Fig. 5, illustrating the overall structure, which includes three 4-bit shift-and-add units to manage precision reformatting. The aggregator is triggered once per accumulation chunk, where the chunk size can vary—from a single data point to a larger grouping, such as a quantization group or a size specific to the dataflow requirements.

Fig. 5 (b)-(d) outlines the aggregation process for each mode. For INT4-INT4, the aggregator functions as a selector, simply forwarding the chosen output. For INT4-INT8, each of the 1st-stage shift-and-add operators produces an accumulated 4×8 output and sends it to the selector. Lastly, in INT4-FP16 mode, the aggregator performs a dedicated normalization process. A two-stage shift-and-add operation combines all position register values into a single representation and trims the mantissa bits as needed, outputting the final 16-bit FP format.

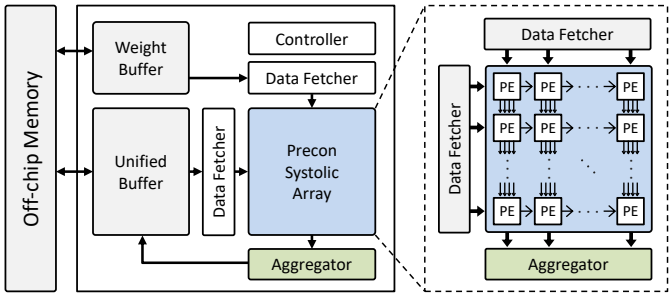


Fig. 6. High-level overview of *Precon* architecture.

C. Overall Architecture

Fig. 6 presents the overall architecture of Precon. A 2D systolic array is implemented with output stationary dataflow, utilizing the proposed MAC units. Data fetchers are implemented between the buffer and array to feed input activations and weights in proper timing. The unified buffer stores both input and output activation data and streams them into the array, while the weight buffer stores quantized weights. Unlike the structure of a single MAC unit, the aggregator is positioned only at the end of each column in the array, where stored data in the registers are transferred down the column to the aggregator unit, which then finalizes the output. *Precon* offers three precision modes—INT4-FP16, INT4-INT4, and INT4-INT8—enabling flexible format conversion to meet the precision requirements of various quantized deep learning models.

IV. Evaluation

A. Experimental Setup

Quantized Models We adopt state-of-the-art PTQ frameworks of weight-only quantization [15] and INT4/8 quantization [19]. To ensure a fair comparison of accuracy across different precision levels, we use the same baseline model and reproduce PTQ models using a consistent small calibration set from the Pile dataset [28]. For the analysis of both accuracy and hardware performance, we use quantized LLM models from the OPT family [7], as well as LLaMA1-7B and LLaMA2-7B models [6]. Zero-shot accuracy of the LLMs is evaluated on six benchmarks: PIQA, BoolQ, Arc-Easy, Arc-Challenge, HellaSwag, and WinoGrande. Additionally, we evaluate and compare perplexity on WikiText-2 [29] to further assess LLM stability at different precision levels.

To broaden the evaluation across various AI domains, we also analyze the hardware performance of deep learning models under their respective state-of-the-art PTQ settings, including CNNs [30]–[32], Transformer-encoder models such as BERT [33], and Vision Transformers (ViTs) [34], [35].

Hardware Configuration A Verilog RTL model of a 32×32 systolic array of *Precon* is modeled and synthesized using Synopsys Design Compiler with a 65nm library. Power consumption is analyzed with Primetime PX (PTPX) based on post-synthesis simulations with real test vectors. On-chip scratchpad buffers are modeled using CACTI [36], and off-chip memory access is considered based on [37]. To evaluate latency

TABLE I
LLM ACCURACY ANALYSIS

	OPT 125M	OPT 1.3B	OPT 2.7B	OPT 6.7B	OPT 13B	LLaMA1 7B	LLaMA2 7B
	Average zero-shot accuracy on six benchmarks* ↑						
Baseline	43.41	51.83	54.76	59.19	59.87	63.43	63.79
INT4-FP16	40.96	50.38	55.46	59.05	60.75	63.02	64.05
INT4-INT8	42.64	51.00	54.68	56.41	47.95	62.52	64.06
INT4-INT4	36.47	35.24	35.94	34.91	35.61	48.61	48.43
	WikiText-2 perplexity (PPL) ↓						
Baseline	27.76	14.63	12.50	10.86	10.12	5.69	5.49
INT4-FP16	29.22	14.87	12.65	10.83	10.15	5.81	5.61
INT4-INT8	30.34	15.66	13.49	12.48	14.02	5.90	5.83
INT4-INT4	-	-	-	-	-	13.76	-

** PIQA, BoolQ, Arc-Easy, Arc-Challenge, HellaSwag, and WinoGrande*

and memory access counts, a cycle-accurate simulator with weight- and output-stationary dataflow is employed. We first conduct a detailed analysis at the single processing unit level, followed by a comparison of our accelerator with the baseline FP16 architecture and the state-of-the-art accelerator featuring weight-only quantization [22].

B. Accuracy Analysis

We evaluate the accuracy and perplexity of various LLMs tested under different precision levels to assess their applicability. Considering the slight precision variations in FP16, which offers a wider exponent range but incurs a 1-bit mantissa truncation, these effects are incorporated into our weight-only quantized LLM PTQ simulation. Additionally, it is important to note that our processing unit module was tested on 10,000 actual test vectors, and an accumulation error of only 0.078% was observed during 128 accumulations per output using the proposed accumulation method. To account for these minor errors, random errors are injected based on a quantization group size of 128. By reproducing the accuracy of INT-based models under the same PTQ settings of INT4-FP16, we ensure a fair comparison of the reliability of various quantized LLMs.

Table I presents the LLM accuracy results using different precision levels for linear layers. As shown, the INT4-FP16 configuration demonstrates accuracy levels comparable to the baseline across a range of LLM sizes. While the zero-shot accuracy for the smallest OPT model exhibits noticeable degradation, INT4-FP16 consistently achieves the smallest accuracy drop and the lowest perplexity across all models. Furthermore, the stability of INT4-FP16 improves with increasing model size, categorizing this weight-only quantized LLMs as *accurate LLMs*. In the INT4-INT8 mode, which uses 4-bit weights and 8-bit activations, reasonable accuracy is preserved in most models. Several models even outperform INT4-FP16 in accuracy, but generally incurring slightly higher perplexity than INT4-FP16. With the integration of the data redistribution technique from [19], INT4-INT8 achieves sufficient accuracy for quantized LLMs, classifying it as *efficient LLMs*.

For INT4-INT4, the accuracy results are suboptimal in most LLMs, with accuracy drops exceeding 10% and unstable perplexity performance. Despite these poor outcomes, we do not apply additional techniques to improve accuracy, ensuring a fair comparison across all configurations. Note that recent studies have highlighted the potential of rotation-based PTQ methods to enhance the performance of INT4-only models [21], [25]. Therefore, we include INT4-INT4 in our hardware evaluation for quantized LLM inference, considering as a prospective area for future adoption, contingent on further optimizations.

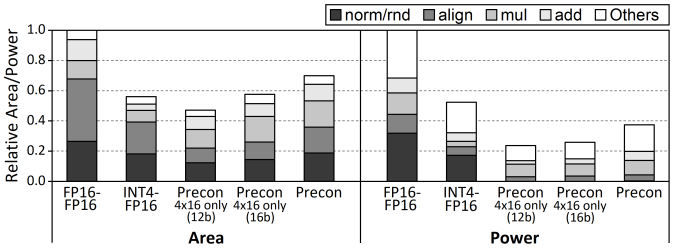


Fig. 7. Area and power comparison of a single MAC unit with breakdowns.

C. Processing Unit Efficiency

We first evaluate the performance of a single MAC unit to demonstrate the hardware efficiency of *Precon*. Comparison models of FP16 and INT4-FP16 baseline units are configured to match the same accumulation error as Precon by adjusting the adder bit-width, ensuring a fair evaluation. A higher clock frequency of 500 MHz is set for the single unit evaluation to highlight the efficacy of our design in terms of critical path delay compared to traditional FP arithmetic units. Additionally, we evaluate different versions of *Precon*, including an INT4-FP16 fixed mode (denoted as "4×16 only") and a configuration with 12-bit registers instead of the default 16-bit registers.

Fig. 7 depicts the relative area and power consumption of different MAC units, including a breakdown of individual modules. FP baseline units exhibit high area occupation in the aligner and normalizer modules. Due to reduced multiplication complexity, the INT4-FP16 baseline achieves significant area and power savings compared to the FP16 baseline. *Precon*, leveraging brick-level MAC operations, replaces high-cost barrel shifters in the aligner and normalizer with fixed 4-bit shift-and-add circuits and position controls. This substitution allows additional area reductions, with the 4×16 only mode achieving a 52.9% area reduction. To accommodate separate INT accumulations and cover all precision levels supported by *Precon*, 16-bit position registers are used by default. Although this design requires additional logic for precision conversion, *Precon* remains 30.1% smaller than the FP16 baseline.

Power consumption breakdowns reveal that a significant portion of power in FP arithmetic units is consumed by the normalizer. The aggregator in *Precon*, which manages normalization and precision conversion, operates on a chunk-based approach, requiring only a single operation per chunk size. This approach enables substantial power savings, as the aggregator remains inactive during computation, consuming only a few microwatts. Consequently, *Precon* achieves power reductions of 62.6% and 28.6% compared to the FP16 baseline and INT4-FP16 baseline units, respectively.

D. Speedup and Energy Efficiency

To evaluate the overall performance and energy efficiency of deep learning models, a 32×32 systolic array of *Precon* is

TABLE II
PRECON IMPLEMENTATION SUMMARY

	Area (mm^2)	Power (W) - 4×16	Power (W) - 4×4/8
32×32 Array	4.024	0.849	0.750
Aggregators	0.038	0.001	0.002
Data Fetchers	0.268	0.101	0.115

	Accurate LLMs	Efficient LLMs	ResNet-18/50	Mobile-V2	Efficient-B0	BERT Models	ViT Models
Precision Level	INT4-FP16	INT4-INT8	INT4-INT4	INT4-INT8	INT4-INT8	INT4-INT4	INT4-INT8

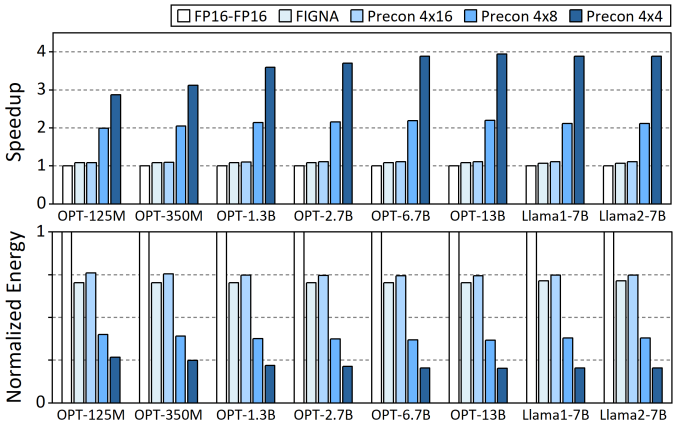


Fig. 8. Analysis of speedup and energy consumption in various LLMs.

implemented. Detailed area and power consumption are summarized in Table II, which shows slightly reduced power usage in INT modes. Two comparison architectures are prepared: the FP16-baseline and FIGNA [22], which represents state-of-the-art INT4-FP16 acceleration by using INT-only MAC units with an approximated block-based FP pre-alignment process.

Fig. 8 illustrates the speedup and normalized energy consumption of various quantized LLMs. All precision levels supported by *Precon* are evaluated, irrespective of accuracy, as discussed in Section IV-B. The throughput is consistent across the comparison models, but FIGNA and *Precon* 4×16 exhibit slightly lower latency compared to the FP16 baseline due to increased loading latency observed in the latter. As the model size increases, *Precon* achieves up to 2.2× and 3.9× speedup compared to the baseline in 4×8 and 4×4 modes, respectively. In smaller models, the limited size of linear-layer channels prevents full exploitation of parallel processing of up to four tokens in low-precision modes, resulting in lower speedup. Energy consumption follows a similar trend, with additional reductions attributed to decreased data access and chip power usage compared to the FP16 baseline. *Precon*'s INT4-FP16 mode exhibits slightly higher energy consumption than FIGNA due to higher chip power demands. While FIGNA benefits from fully INT-based MAC arrays, *Precon* provides the flexibility to support single-MAC FP arithmetic, enabling potential improvements in scalability for implementation.

To demonstrate *Precon*'s versatility, various deep learning models across AI domains are evaluated using their optimal precision levels. For models in other domains like CNNs and Transformer encoders, which are more error-resilient than LLMs, aggressively quantized inference can be considered. Based on prior PTQ studies, the lowest precision ensuring

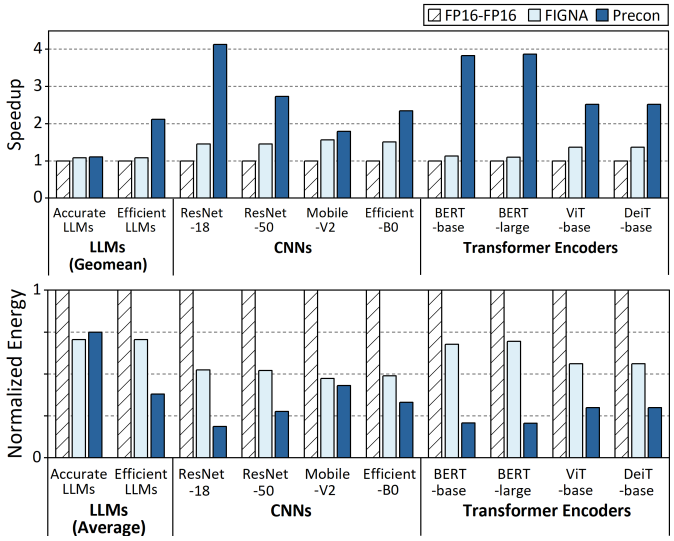


Fig. 9. Speedup and energy savings in various AI domains.

reliable accuracy is chosen for each model, as outlined in Table II. LLM results are categorized into previously defined *accurate* and *efficient* LLMs, with average values reported. Fig. 9 presents the overall speedup and energy efficiency across these models. While similar performance to FIGNA is observed in INT4-FP16 mode, *Precon* outperforms in lower precision modes, such as INT4-INT8 and INT4-INT4. This results in up to 4.1× speedup and 81.4% reduction in energy consumption compared to the baseline. These findings highlight the capability of *Precon* to adapt to diverse quantized inference requirements, spanning error-resilient CNNs to error-sensitive LLMs, through effective precision conversion in both bit-width and data format.

V. CONCLUSION

This paper presents *Precon*, a precision-convertible processing unit architecture designed to support quantized inference across a spectrum of models, ranging from error-resilient to error-sensitive, through mode conversions in both bit-width and data format. By leveraging the exponent-encoded FP16 format, which can be decomposed into 4-bit INT data bricks, the proposed MAC unit integrates an aligner-combined adder and aggregator to enable INT4-FP16, INT4-INT4, and INT4-INT8 arithmetic. *Precon* demonstrates superior hardware performance compared to existing processing unit designs, achieving up to 4.1× speedup and 82.8% reduction in energy consumption across diverse AI domains compared to the baseline systolic array architecture.

ACKNOWLEDGMENT

This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (2024-00352468); and in part by Korea Collaborative and High-Tech Initiative for Prospective Semiconductor Research (K-CHIPS) funded by the Ministry of Trade, Industry and Energy (MOTIE), South Korea, under Grant RS-2024-00405946. The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[2] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[3] C. Tang, B. Abbatematteo, J. Hu, R. Chandra, R. Martín-Martín, and P. Stone, "Deep reinforcement learning for robotics: A survey of real-world successes," *arXiv preprint arXiv:2408.03539*, 2024.

[4] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: insights of machine learning and deep learning algorithms," *Machine Learning with Applications*, vol. 6, p. 100164, 2021.

[5] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[6] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[7] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[8] C.-J. Wu, R. Raghavendra, U. Gupta, B. Acun, N. Ardalani, K. Maeng, G. Chang, F. Aga, J. Huang, C. Bai *et al.*, "Sustainable ai: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[9] S. Hashemi, N. Anthony, H. Tann, R. I. Bahar, and S. Reda, "Understanding the impact of precision quantization on the accuracy and energy of neural networks," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017, pp. 1474–1479.

[10] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, "Compensated-dnn: Energy efficient low-precision deep neural networks by compensating quantization errors," in *Proceedings of the 55th annual design automation conference*, 2018, pp. 1–6.

[11] R. Banner, Y. Nahshan, and D. Soudry, "Post training 4-bit quantization of convolutional networks for rapid-deployment," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[12] H. Tang, X. Zhang, K. Liu, J. Zhu, and Z. Kang, "Mkq-bert: Quantized bert with 4-bits weights and activations," *arXiv preprint arXiv:2203.13483*, 2022.

[13] Z. Yuan, C. Xue, Y. Chen, Q. Wu, and G. Sun, "Ptq4vit: Post-training quantization for vision transformers with twin uniform quantization," in *European conference on computer vision*. Springer, 2022, pp. 191–207.

[14] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[15] J. Lin, J. Tang, H. Tang, S. Yang, W.-M. Chen, W.-C. Wang, G. Xiao, X. Dang, C. Gan, and S. Han, "Awq: Activation-aware weight quantization for on-device llm compression and acceleration," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 87–100, 2024.

[16] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Optq: Accurate quantization for generative pre-trained transformers," in *The Eleventh International Conference on Learning Representations*, 2022.

[17] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[18] G. Park, B. Park, M. Kim, S. Lee, J. Kim, B. Kwon, S. J. Kwon, B. Kim, Y. Lee, and D. Lee, "Lut-gemm: Quantized matrix multiplication based on luts for efficient inference in large-scale generative language models," *arXiv preprint arXiv:2206.09557*, 2022.

[19] J. Liu, R. Gong, X. Wei, Z. Dong, J. Cai, and B. Zhuang, "Qllm: Accurate and efficient low-bitwidth quantization for large language models," *arXiv preprint arXiv:2310.08041*, 2023.

[20] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," *arXiv preprint arXiv:2405.04532*, 2024.

[21] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," *arXiv preprint arXiv:2404.00456*, 2024.

[22] J. Jang, Y. Kim, J. Lee, and J.-J. Kim, "Figna: Integer unit-based accelerator design for fp-int gemm preserving numerical accuracy," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 760–773.

[23] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, J. K. Kim, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 764–775.

[24] D. Im, G. Park, Z. Li, J. Ryu, and H.-J. Yoo, "Sibia: Signed bit-slice architecture for dense dnn acceleration with slice-level sparsity exploitation," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 69–80.

[25] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "Spinquant–llm quantization with learned rotations," *arXiv preprint arXiv:2405.16406*, 2024.

[26] S. Choi, J. Shin, and L.-S. Kim, "A deep neural network training architecture with inference-aware heterogeneous data-type," *IEEE Transactions on Computers*, vol. 71, no. 5, pp. 1216–1229, 2021.

[27] D. Kalamkar, D. Mudigere, N. Mellempudi, D. Das, K. Banerjee, S. Avancha, D. T. Vooturi, N. Jammalamadaka, J. Huang, H. Yuen *et al.*, "A study of bfloat16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[28] L. Gao, S. Biderman, S. Black, L. Golding, T. Hoppe, C. Foster, J. Phang, H. He, A. Thite, N. Nabeshima *et al.*, "The pile: An 800gb dataset of diverse text for language modeling," *arXiv preprint arXiv:2101.00027*, 2020.

[29] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[31] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019.

[32] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[33] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[35] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[36] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[37] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

DuoQ: A DSP Utilization-aware and Outlier-free Quantization for FPGA-based LLMs Acceleration

Zhuoquan Yu*, Huidong Ji*, Yue Cao, Junfu Wu, Xiaoze Yan, Lirong Zheng†, and Zhuo Zou†
State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China
Email: {zqyu21, hdji21, caoyue23, junfuwu23, xzyan23}@m.fudan.edu.cn, {lrzheng, zhuo}@fudan.edu.cn

***Abstract*—Quantization enables efficient deployment of large language models (LLMs) on FPGAs, but its presence of outliers affects the accuracy of the quantized model. Existing methods mainly deal with outliers through channel-wise or token-wise isolation and encoding, which leads to expensive dynamic quantization. To address this problem, we introduce DuoQ, an FPGA-oriented algorithm-hardware co-design framework. DuoQ effectively eliminates outliers through learnable equivalent transformations and low-semantic token awareness in the quantization scheme part, facilitating per-tensor quantization with 4-bits. We co-design the quantization algorithm and hardware architecture. Specifically, DuoQ accelerates end-to-end LLM through a novel DSP-aware PE unit design and encoder design. In addition, two types of post-processing units assist in the realization of nonlinear functions and dynamic token awareness. Experimental results show that compared with platforms with different architectures, DuoQ's computational efficiency and energy efficiency are improved by up to 8.8× and 23.45×. In addition, DuoQ has achieved accuracy improvements compared to other outlier-aware software and hardware works.**

***Index Terms*—FPGA, W4A4 quantization, LLMs, DSP utilization, algorithm-hardware co-design**

I. Introduction

Transformer-based large language models (LLMs) are gaining attention for their excellent performance in natural language processing (NLP) scenarios such as text understanding and generation. However, the construction and processing of models involve a large number of parameters and operations, which also pose challenges [1], [2]. For example, the GPT3-175B (FP16) requires 326 GB of memory, and the amount of computation required to complete an inference is approximately 660 TOPS.

Field-programmable gate array (FPGA) is a potential hardware platform with flexible resource scheduling and tailored architecture design to accelerate LLM inference and explore the benefits of optimization algorithms [3]–[5]. However, its computational capabilities remain largely underutilized. Efficiently utilizing DSPs, core FPGA units optimized for arithmetic tasks like multiplication and accumulation, can support more parallel operations, reducing computational load and boosting throughput in LLM tasks.

Quantizing those parameters to a low bit-width is a powerful technique to achieve optimal utilization for better computational throughput [6]. It can also greatly reduce memory footprint and bandwidth pressure. However, quantizing activations to low bit-widths, like 4-bit, in LLM operations can cause outliers, significantly degrading model accuracy. Previous work has proposed many types of methods to address this problem [7]–[11], such as channel-wise scaling, mixed-precision quantization with dedicated hardware support, Hadamard rotation, and channel-wise assembly. These methods all heuristically mark outliers and dynamically quantize the channels and tokens where the outliers are located, and cannot eliminate the impact of outliers. This paper aims to eliminate outliers at the tensor level while minimizing hardware overhead. We leverage the observation that pervasive channel outliers manifest across blocks and layers, as shown in Fig. 2(a) and Fig. 2(b), and that tokens with low semantic specificity exhibit significant outlier behavior.

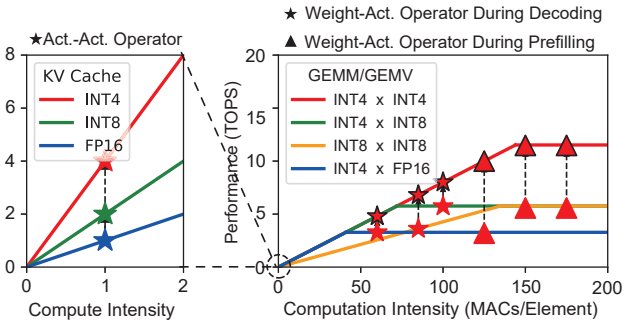
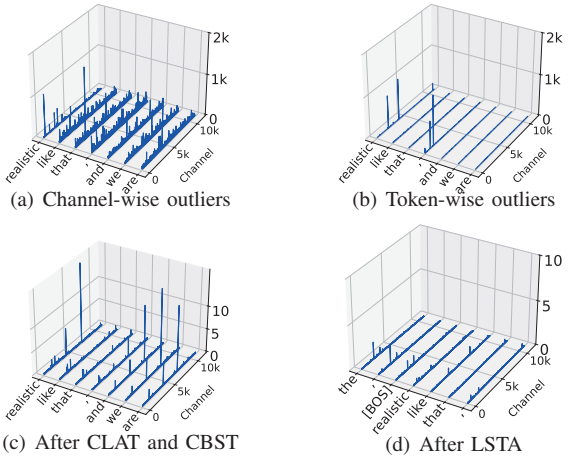


Fig. 1. Roofline model of DSP-driven FPGA platform.

We introduce DuoQ, a complete mapping and outliers-free algorithm-hardware co-design framework for LLM implementation on FPGAs, which ensures a balance between accuracy and hardware efficiency. We first reveal the potential optimization space of current FPGA accelerators for LLM, and analyze the distribution of outliers and corresponding optimization schemes. Then, to maximize the elimination of the impact of outliers, we propose a W4A4KV4 (4-bit weight, 4-bit activation, and 4-bit KV cache) per-tensor quantization framework. We build and implement an FPGA-based accelerator architecture compatible with the quantization framework. DuoQ achieves efficient computation while ensuring accuracy through comprehensive algorithm and hardware collaboration. The main contributions of this paper are listed as follows:

- We propose DuoQ, an FPGA-oriented algorithm-hardware co-design framework, as shown in Fig. 3, which hosts an outlier-free quantization scheme and a corresponding FPGA architecture design.
- We introduce an outlier-free quantization scheme jointly optimized by the equivalent transformations and the low-

This work was supported in part by the National Key Research and Development Program of China under Grant 2023YFB4704101. (*Equal contribution, †Corresponding author.)



(a) Channel-wise outliers (b) Token-wise outliers

(c) After CLAT and CBST (d) After LSTA

Fig. 2. Statistical chart of LLaMA2-7B intermediate activation values.

semantic-token awareness methods to ensure accuracy.

- A co-designed FPGA-based architecture is built to improve the DSP's resource utilization and computation efficiency.
- We implement the FPGA-based architecture from the framework. The experimental results in LLaMA-7B and LLaMA-13B show that DuoQ reduces WikiText2 perplexity by 5.56 and 2.31, respectively, compared to other state-of-the-art (SOTA) outlier-aware hardware works. Moreover, compared to platforms with different architectures, DuoQ improves computational efficiency and energy efficiency by a maximum of 8.8× and 23.45×.

II. Background and Motivation

A. Related Accelerators for LLMs

Recently, transformer-based LLM accelerators have been proposed, but they face three key limitations. a) [3], [12]–[14] mainly focus on accelerating the attention mechanism by pruning or quantization. They do not implement end-to-end LLM deployment and ignore the equally important linear layer computation with computational bottlenecks. b) Although FPGA-based accelerators [15]–[17] provide comprehensive acceleration, they often overlook software-driven algorithmic enhancements for performance. For example, low-bit quantization, as shown in Fig. 1, INT4-INT4 quantization compared to other precisions has obtained different levels of performance improvements in the prefill and decode stages of the model. c) [6], [18], [19] consider advanced quantization algorithms that include outliers and are co-designed with the hardware. However, their methods for encoding and isolating outliers are limited by fixed quantization strategies and input data and are not optimal.

Moreover, FPGA-based accelerators such as [16] and [18] do not fully utilize the important DSP resources, and the optimization potential has not been fully explored. In FPGAs, DSP-based MAC operations have higher resource utilization than LUTs, with a computational efficiency of about 90 times that of LUTs, as shown in [20] and [21]. Therefore, it is important to make efficient use of DSP resources. Current DSPs support INT8 multiplication, however, the extended support for INT4 is limited by the finite bit length and cannot

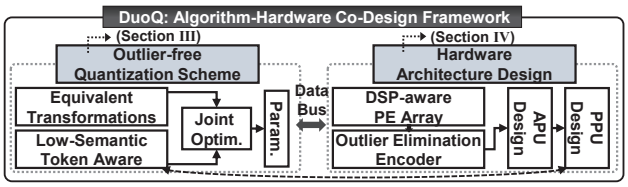


Fig. 3. The proposed algorithm-hardware co-design framework.

describe the complete information. [16], [18], [22] claim the INT4 quantization support, but the W4A8-based mixed precision method proves no computation efficiency improvement, compared to the INT8-based DSP packing. [23] attempts to deploy a converter with INT4 multiplication on an FPGA, but since it cannot support signed operands as inputs, the INT4 implementation is limited to the post-nonlinearity layer. [24] utilizes the ability to run six multiplications in parallel, but still requires significant external hardware resources to help recover the required information and accuracy.

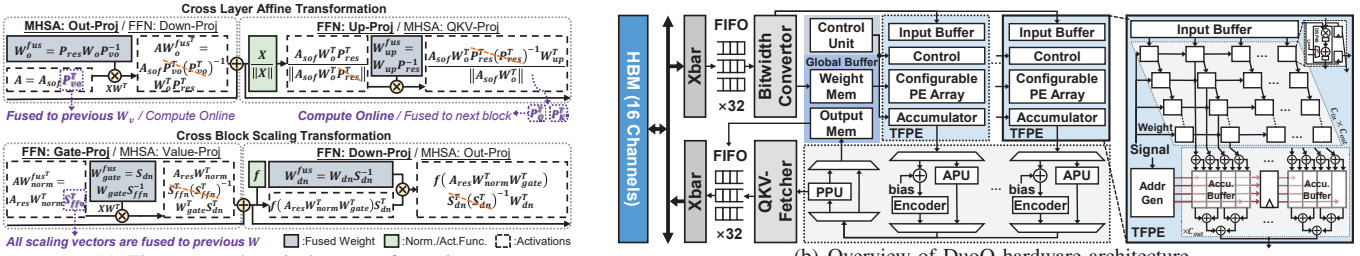
B. W4A4KV4 Quantization and Outliers Analysis

Fig. 2 shows the activation distribution of a representative linear layer in LLaMA2-7B. It contains many outliers as mentioned in the works [7], [8], as shown in Fig. 2(a), prevalent across channels and manifest across blocks and layers. More accurate encoding of these activations and outlier isolation can alleviate this problem, as shown in [6], [14], [19], but at the cost of mixed-precision methods and overheads such as reordering. More importantly, this approach cannot adapt to diverse input sequences, as the token-wise anomalies are input-dependent. Other works try to balance the weight and activation anomalies by adjusting the per-channel scaling of the diagonal elements in the matrix multiplication view. Recent works such as hadamard transformation [10], [11] have considered the features across layers and individual features, but the transformed activations are not smooth enough. They also do not take into account tokens that cause a large number of outliers. In addition, consistent with the observations in [25], the starting token and the first special token in Fig. 2(b) will exhibit large activation values. In this paper, they are collectively referred to as low-semantic tokens. It should be noted that the distribution law is not consistent among different models, so we will use the joint optimization method proposed in Section III-C for targeted optimization.

C. An Algorithm-Hardware Co-Design for INT4 LLM's Highly Efficient Implementation on FPGA

In response to these challenges concerning the quantization and hardware design, we propose the algorithm-hardware co-design framework, as displayed in Fig. 3. Concerning the quantization scheme, equivalent transformations across layers and blocks are designed to optimize weight and activation distributions, effectively mitigating outliers. The Low-semantic-awareness attention method is used to flatten the token-wise outliers, shown in Fig. 2(c), to reduce the accuracy loss, exhibited in Fig. 2(d).

An FPGA-based architecture is built to enable the quantized LLM deployment with improved computation and energy efficiency following the listed optimizations: 1) A novel



(a) The proposed equivalent transformations

(b) Overview of DuoQ hardware architecture

Fig. 4. DuoQ's framework overview.

DSP-aware and reconfigurable PE; 2) An outlier-elimination encoder; 3) A carefully designed auxiliary processing unit (APU); and 4) A post-processing unit (PPU) realizing efficient nonlinear functions and dynamic awareness of tokens.

III. Outlier-free Quantization Scheme

In transformer-based LLM, each decoder layer contains Multi-Head Self-Attention (MHSA) and Feed-Forward Network (FFN), and both of them include input normalization. Specifically, for linear layer computation, MHSA includes attention query/key/value projection block and output projection block, which are denoted as $\mathcal{B}_M = \{q, k, v, o\}$. The up/gate projection block and down projection block contained in the FFN are represented as $\mathcal{B}_F = \{up, gate, down\}$. The calculation of each block is expressed as $\mathbf{A}_b\mathbf{W}_b^\top$, where $\mathbf{A} \in \mathbb{R}^{N\times C_{in}}$ represents the activation of the input, and $\mathbf{W} \in \mathbb{R}^{C_{out}\times C_{in}}$ represents the weight matrix, and $b \in (\mathcal{B}_M \cup \mathcal{B}_F)$ represents the subscript corresponding to the above block.

A. Equivalent Transformations Across Layers and Blocks

As shown in Fig. 4(a), the proposed equivalent transformations include the cross-layer affine transformation (CLAT) and the cross-block scaling transformation (CBST). Inspired by [10], the CLAT transforms the channels' activations and weights to re-distribute the outliers and effectively eliminate them. This results in more homogeneous values facilitating the quantization. Crucially, the quantization error is also reduced across layers in the network. First, CLAT constructs an affine transformation pair with the inverse of the transformation being the identity matrix, i.e. $\mathbf{P}\mathbf{P}^{-1} = \mathbf{I}$, including the transformation pairs of the decoder layers, i.e.$\{\mathbf{P}_q, \mathbf{P}_k, \mathbf{P}_{vo}, \mathbf{P}_{res}, \mathbf{P}_{dn}\}$, where $\mathbf{P}_q = \mathbf{P}_k$. This makes the equation $\mathbf{Y} = Q(\mathbf{A}\mathbf{P}^\top)Q((\mathbf{P}^\top)^{-1}\mathbf{W}^\top)$ hold in the standard linear layer calculation. Its forward process is mathematically equivalent to the original model, ignoring the effect of quantization.

For the initial input to the MHSA and FFN, the commutation property of RMSNorm can be used to obtain that $\mathbf{X}\mathbf{P}/\|\mathbf{X}\mathbf{P}\| = (\mathbf{X}/\|\mathbf{X}\|)\mathbf{P}$, with $\mathbf{X}$ denoting the input to the block. Then, the output is denoted by $\mathbf{Y}$, and for the residual connection between the computation blocks, taking the MHSA as an example, after taking into account $\mathbf{P}$, we get: $\mathrm{MHSA}(\mathbf{X} \mid \mathbf{W}_b, b \in \mathcal{B}_M)\mathbf{P}_{res}$ where the remaining $\mathbf{P}_{res}$ is eliminated equivalently and the original $\mathbf{W}_b$ is retained. The connection is then constructed via $\mathbf{W}_o\mathbf{P}_{res}$ as

$$\mathbf{X}\mathbf{P}_{res}+\mathrm{MHSA}(\mathbf{X}\mathbf{P}_{res} \mid \mathbf{W}_b\mathbf{P}_{res}, b{=}o) = (\mathbf{X}{+}\mathbf{Y})\mathbf{P}_{res}, \quad (1)$$

where the activation value $\mathbf{X}\mathbf{P}_{res}$ of $\mathbf{P}_{res}$ is fused to the previous layer's $\mathbf{W}_{dn}$ through a cross-layer. Similarly, $\mathbf{P}_{vo}$ performs fusion to $\mathbf{W}_v$ to avoid online computation.

In particular, to minimize the gap between activations, we jointly optimize the $\mathbf{P}$s fused into $\mathbf{W}$ via Section III-C. For the online running $\{\mathbf{P}_q, \mathbf{P}_k, \mathbf{P}_{dn}\}$, they are formed by special affine transformations of the Hadamard orthogonal matrix. For CBST, it introduces a scaling vector $\vec{\mathbf{s}}$ between each block of $\mathcal{B}_M\mathcal{B}_F$. Then, by using $\vec{\mathbf{s}}$, a diagonal scaling matrix $\mathbf{S} = \mathrm{diag}(\vec{\mathbf{s}}) \in \mathbb{R}^{C_{in}}$ is constructed to be applied to the input/output across the blocks. Unlike CLAT, all $\mathbf{S}$ are fused into $\mathbf{W}$.

B. Low-Semantic Token Aware Attention Method

We design a low-semantic token-aware attention method (LSTA) that works in the attention module of MHSA. The query vector obtained by the projection block of MHSA is represented as $\vec{\mathbf{q}} \in \mathbb{R}^{C_{out}}$, and the KV caches are represented as $\mathbf{K}, \mathbf{V} \in \mathbb{R}^{C_{out}\times N}$. Their quantization errors are defined as $\mathrm{E}_\mathbf{K} = \mathbf{K} - \bar{\mathbf{K}}$ and $\mathrm{E}_\mathbf{V} = \mathbf{V} - \bar{\mathbf{V}}$. We first evaluate the error of the attention score $\mathbf{s} = \mathrm{softmax}(\vec{\mathbf{q}}\mathbf{K}^\top/\sqrt{d_k})$. Using the Lipschitz continuity of the softmax function, for inputs $\mathbf{x}$ and $\bar{\mathbf{x}}$, we have $\|\mathrm{softmax}(\mathbf{x}) - \mathrm{softmax}(\bar{\mathbf{x}})\|_1 \le \|\mathbf{x} - \bar{\mathbf{x}}\|_1$. The error of $\vec{\mathbf{q}}$ is relaxed to $\|\vec{\mathbf{q}}\|_\infty \le \gamma_{\vec{\mathbf{q}}}$, so the upper bound of the total error of the attention head $\mathbf{o} = \mathbf{s}\mathbf{V}$ is estimated as

$$\|\mathbf{o} - \bar{\mathbf{o}}\| \le \|\mathrm{E}_\mathbf{V}\|_\infty + \frac{\gamma_{\vec{\mathbf{q}}}}{\sqrt{d_k}}\|\mathrm{E}_\mathbf{K}\|_1\|\mathbf{V}\|_\infty. \quad (2)$$

It can be seen that $\|\mathrm{E}_\mathbf{V}\|_\infty$ directly controls the quantization error on the output, while $\|\mathrm{E}_\mathbf{K}\|_1$ determines the propagation of the attention score error. Therefore, driven by the experimental phenomenon in Fig. 2, we decompose the error by the low-semantic token awareness in such a way that $\mathrm{E}_\mathbf{V} = \mathrm{E}_{\mathbf{V}^{\mathrm{LST}}} + \mathrm{E}_{\mathbf{V}^{\mathrm{NOR}}}$, i.e., the error $\mathrm{E}_\mathbf{V}$ is decomposed into the error caused by low-semantic tokens and normal tokens $\mathbf{V}^{\mathbf{LST}}, \mathbf{V}^{\mathbf{NOR}} \subseteq \mathbf{V}$ and satisfies $\|\mathrm{E}_\mathbf{V}\|_\infty = \max(\|\mathrm{E}_{\mathbf{V}^{\mathrm{LST}}}\|_\infty, \|\mathrm{E}_{\mathbf{V}^{\mathrm{NOR}}}\|_\infty)$. Since $\|\mathrm{E}_{\mathbf{V}^{\mathrm{LST}}}\|$ is excluded, the contribution of $\mathrm{E}_{\mathbf{V}^{\mathrm{NOR}}}$ to the outliers is small, this upper bound is smaller, effectively controlling the attention output error. Finally, LSTA finds these tokens τ and embeds them in the KV caches using the joint optimization method in Section III-C to constrain the outliers in $\mathbf{V}^{\mathbf{LST}}$ and $\mathbf{K}^{\mathbf{LST}}$.

C. Joint Optimization Method

Our goal is to minimize the mean square error of the output of the transformer layer between the full-precision LLM and the quantized LLM. Therefore, the following loss function is considered as

$$\mathcal{L}(\theta, \pi, \tau) = \mathbb{E}_\mathbf{x}[\|\mathcal{F}(\mathbf{x}) - \hat{\mathcal{F}}_{\theta,\tau}(\mathcal{T}(\mathbf{x}); \pi)\|_F^2], \quad (3)$$

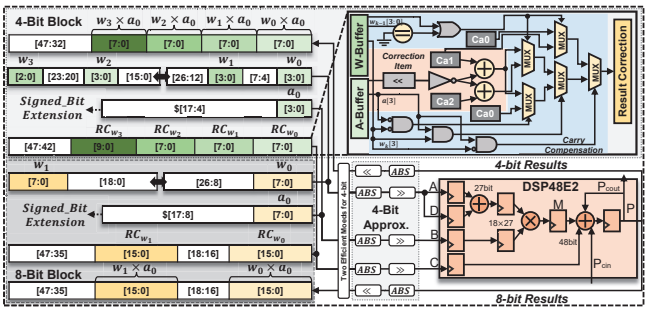


Fig. 5. Efficient packing of operands on a single DSP48E2.

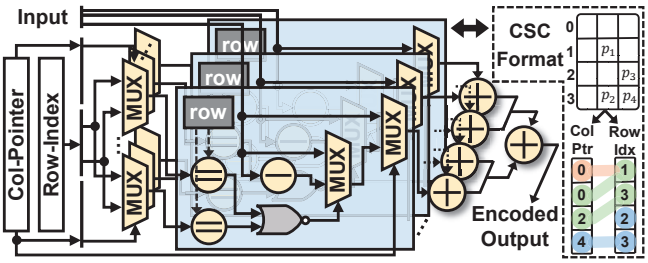


Fig. 6. Encoding the intermediate activation to eliminate outliers.

where $\pi = \{\mathbf{P}, \vec{\mathbf{s}}\}$ and $\|\cdot\|_F$ is the Frobenius norm. Given the quantization parameter θ and the token τ embedded in the model KV caches, $\mathcal{F}(\cdot)$ and $\hat{\mathcal{F}}_{\theta,\tau}(\cdot)$ denote the outputs before and after quantization of the transformer layer under the input tensor $\mathbf{x}$, respectively. $\mathcal{T}(\mathbf{x})$ is the equivalent transformation in Section III-A. It is therefore solved for

$$\min_{\theta,\pi,\tau} \mathcal{L}(\theta, \pi, \tau). \tag{4}$$

Similar to the case of [26], this formulation is analogous to the k-means clustering process. Consider θ and τ as the cluster centers, which define the mapping before and after quantization. π is then considered as the mapping from the sample points to the cluster centers. This problem is then solved using an alternating optimization strategy. First, π is fixed and a subproblem about θ and τ is solved, i.e., $\theta^t, \tau^t \leftarrow \arg\min_{\theta,\tau} \mathcal{L}(\theta, \pi^{t-1}, \tau)$, where t is the alternating index. They are optimized by stochastic gradient descent. Since they are fixed, the gradients are not backpropagated to π^{t-1}. Next, fixing θ and τ, the subproblem about π is solved, i.e., $\pi^t \leftarrow \arg\min_{\pi} \mathcal{L}(\theta^t, \pi, \tau^t)$. The key to this optimization is the optimization of $\mathcal{T}(\mathbf{x})$, i.e.,

$$\min_{\mathbf{P}} \|\mathbf{xP} - Q(\mathbf{xP})\|_F^2. \tag{5}$$

This is an equilibrium Procrustes problem subject to the constraint that $\mathbf{P} \in \mathcal{S}$ is an orthogonal stiefel manifold. We can obtain an explicit solution by singular value decomposition of the matrix $\mathbf{x}^\top Q(\mathbf{xP})$. Then, we perform an alternating approximation step, substitute the approximate π^t into the subproblems of θ^t and τ^t, and continue the alternating solution until convergence.

IV. Hardware Architecture Design

A. Overall Architecture

Fig. 4(b) shows the overall architecture of DuoQ. It adopts a five-stage pipeline structure to ensure hardware utilization: **1) Preloading:** Due to the large read latency of HBM, the weight blocks are preloaded into 32 64-depth×16B FIFOs and converted by the bitwidth convector. **2) Loading off-chip data:** After the control unit has propagated the control signals to the bitwidth converter, the weights and input data are read from the FIFOs into the buffer. The buffer consists of a BRAM array distributed across the input buffers of each token feature processing element (TFPE) and a global buffer. **3) MAC operations:** The data corresponding to C_{in} and C_{out} flows through the PE array in each TFPE and is multiplied and accumulated. In particular, the PE units are designed to efficiently calculate signed 4-bit numbers from the input via DSP-packing. **4) Post-processing operations:** The auxiliary processing unit (APU) performs non-linear operations on the intermediate results. For the encoder, it ensures that the output values are returned to storage and the KV caches are accurately quantized without outliers. **5) Writing data back to off-chip memory:** Finally, the output buffer is also written back to HBM via 32 FIFOs. The qkv-fetcher before the output buffer writes the address of the corresponding vector back with the result of the token importance of the post processing unit (PPU), to combine Section III-B.

B. DSP-aware and Reconfigurable PE Unit Design

As shown in Fig. 5, the DSP's A and D input interfaces support $K \in \{2, 4, 8\}$ w_k packings in different calculation modes. They share the same activation a_0 and input to B port. The label of each w_k is denoted as $k \in \mathcal{K} = \{0, 1, \cdots, K\}$. Then, the n-th bit of w_k is denoted as $n \in \mathcal{N} = \{0, 1, \cdots, N\}$, where $N \in \{4, 8\}$ denotes the support of the maximum bit width of our design. Therefore, w_k of N bits is represented as $w_k[N-1:0] = \{w_k[N-1], \cdots, w_k[n], \cdots, w_k[0]\}$, where $[\cdot]$ is used to indicate the index.

*1) **Activation Shared Signed 8-bit Packed Calculation**:* When $N = 8$ and $K = 2$, since the sign bit of $w_0 (k = 0)$ is not on the most significant bit (MSB), the actual result obtained is not the target $a_0 \cdot w_0$, but the unsigned $a_0 \cdot \hat{w}_0$ generated by $a_0 \cdot w_0$. $\hat{w}_0$ can be expressed as $\hat{w}_0 = \sum_{n=0}^{N-2} w_0[n] \cdot 2^n + w_0[N-1] \cdot 2^{N-1}$. Then, we transform the expression of $\hat{w}_0$ to

$$\hat{w}_0 = w_0 + w_0[N-1] \cdot 2^N. \tag{6}$$

From this, we can get $a_0 \cdot \hat{w}_0 = a_0 \cdot w_0 + a_0 \cdot w_o[N-1] \cdot 2^N$. In this case, the result after multiplication, $a_0 \cdot \hat{w}_0$, needs to be corrected by subtracting the correction term $Ct_{w_0} = a_0 \cdot w_o[N-1] \cdot 2^N$ to obtain the correct result. And when a_0 is signed, the result $a_0 \cdot w_1$ needs to be corrected by adding the correction term $Ct_{w_1} = 1$.

*2) **Improved Signed 4-bit Packed calculation**:* When $N = 4$ and $K = 4$, the DSP48E2 specifications do not support packing the MSB of w_{K-1}. Instead of the desired result, a_0 and w_{K-1} are fed into the DSP calculation, resulting in $a_0 \cdot \tilde{w}_{K-1}$ with the 3-bit signed number $\tilde{w}_{K-1}[2:0] = \{w[2], w[1], w[0]\}$. From $\tilde{w}_{K-1} = \sum_{n=0}^{N-3} w_{K-1}[n] \cdot 2^n - w_{K-1}[2] \cdot 2^2$, by splitting we get

$$\tilde{w}_{K-1} = w_{K-1} - w_{K-1}[2] \cdot 2^3 + w_{K-1}[3] \cdot 2^3. \tag{7}$$

Therefore, for $a_0 \cdot \tilde{w}_{K-1}$, it can be re-expressed as $a_0 \cdot \tilde{w}_{K-1} = a_0 \cdot w_{K-1} - a_0 \cdot (w_{K-1}[2] - w_{K-1}[3]) \cdot 2^3$. Similarly, we get

Algorithm 1 Integer-Only PWL, $k_i x + t_i (s_i \leq x \leq s_{i+1})$.

Input: x_q, ε: quantized input and scaling factor.
Output: y_q, ε_y: quantized output and scaling factor.

Function I-PWL(x_q, ε) ▷ $x_q \varepsilon = x$
  $s_q \leftarrow \lfloor s/\varepsilon \rfloor, k_q \leftarrow \lfloor k/\varepsilon \rfloor$
  $t_q \leftarrow \lfloor t/\varepsilon^2 \rfloor, \varepsilon_y = \varepsilon^2, y_q = k_q x_q + t_q$
  return y_q, ε_y ▷ $y_q \varepsilon_y \approx kx + t$
end Function

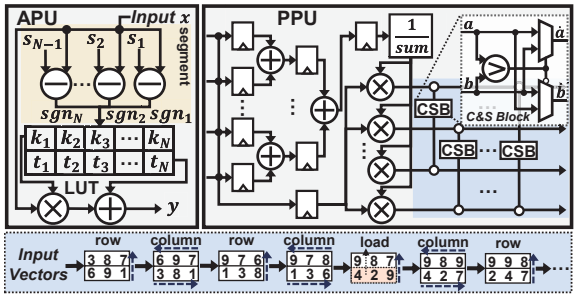


Fig. 7. Hardware microarchitecture of APU and PPU.

the correction term for w_{K-1} as $Ct_{w_{T-1}} = a_0 \cdot (w_{K-1}[2] - w_{K-1}[3]) \cdot 2^3$, which is implemented using only shifters, adders and bitwise AND units. The result of $a_0 \cdot \hat{w}_k$ obtained by packing the low-order bits of $\{w_k \mid k = 0, 1, 2\}$ in the input A, D is also corrected using the method in (6). In addition, signed-signed multiplications will generate carry between adjacent results after packing. Therefore, in addition to w_0 being unaffected, $\{w_k \mid k = 1, 2, 3\}$ requires additional carry compensation ($Ca0$ to $Ca2$) in addition to the respective correction terms. Fig. 5 shows the compensation terms constructed for w_1 and w_2, and w_3 is constructed similarly.

In addition, based on the advantage of the designed structure that expands the calculation using the common operand a, we have designed an approximation technique that can be applied to signed-unsigned calculations where most probabilities are close to 0, such as after softmax, with almost no loss of precision. Specifically, on the basis of the above complete 4-bit packing, an active bit scanner (ABS) and a shifter are added to the DSP input. The number for each port is first entered into the ABS by accumulation, and then for positive numbers, the ABS is indicated by detecting the first active bit equal to 1 (0 for negative), and then shifting right to 4 bits. Finally, the output is restored by shifting left. Thus, each port supports the packing of up to 16 unsigned numbers.

C. Encoder Design for Outlier Elimination

For the equivalent transformation in Section III-A, we design an encoder to be applied in the intermediate activation and eliminate outliers. The properties of $\mathbf{P}_h, h \in \{q, k, dn\}$ are similar to sparsity, and the distribution of $p = -1$ in the matrix is similar to the distribution of non-zero values in sparsity. Therefore, we consider formatting and storing $\mathbf{P}$ in a lookup table first by using the compressed sparse column (CSC) format [27]. Then the encoder in Fig. 6 is used to encode the input activations. Benefiting from this, the proposed encoder can be configured to work in sparse mode, which is applied to the sparse values in $\mathbf{A}_b \mathbf{W}_b^\top$ in addition to the $\mathbf{P}$ transformation. In the encoder design, input activation first selects the column containing p via Col_Ptr, and then uses Row_Idx to perform complement and accumulation operations for each set of activations. In addition, when the constructed $\mathbf{P}$ does not satisfy the size of 2^n, CLAT uses the Kronecker product to decompose it.

TABLE I
PERPLEXITY COMPARISON ON WIKITEXT WITH W4A4 QUANTIZATION

Model	L-7B	L-13B	L-30B	L-65B	O-6.7B	O-13B	O-30B	O-66B
FP16	5.68	5.09	4.1	3.53	10.86	10.12	9.56	9.34
Omniquant [7]	11.26	10.87	10.33	9.17	12.24	11.65	10.6	10.29
Olive [19]	144.75	42.24	36.55	1.4e7	107.15	416.57	334.7	4058.83
Oltron [6]	11.67	8.2	6.68	5.82	11.99	11.61	10.64	10.5
HotaQ [18]	10.72	9.07	7.39	6.02	13.84	13.9	11.02	11.29
DuoQ	6.11	5.89	5.14	4.23	11.31	11.60	10.44	10.21

*L: LLaMA Model, *O: OPT Model, Perplexity: the lower the better

D. Auxiliary Processing Unit

The APU is designed to be used with non-linear functions to support the softmax function, normalization function and activation functions such as SiLU in LLMs. To implement them efficiently without loss of precision, the APU uses a piecewise linear (PWL) approximation. To efficiently implement this approximation, we use a neural network approach to approximation, as in [28]. The parameters of each segment $k_i x + t_i$ if $(s_i < x < s_{i+1})$ are implemented using LUTs. The APU makes two important improvements. The first is to construct an integer-only PWL function as shown in Algorithm 1 to improve execution efficiency. The second is to achieve the best approximation by fine-tuning a piece of data. The structure of the APU is shown on the left side of Fig. 7. It can achieve an MAE of 2.07×10^{-3} for the sigmoid function in experiments, which is better than previous work.

E. Post Processing Unit

The PPU first performs the post-processing operations in Section IV-D. For example, in the case of softmax, the input values after the exponential transformation are summed using an addition tree and the reciprocal is calculated, and then multiplied with the input one by one to obtain a normalized probability distribution. They are then passed to our carefully designed real-time window-based sorter (RTWS) for maximum value set filtering. Due to the pipelined architecture described in Section IV-A and the on-chip memory constraints, the sorter cannot receive all the data in one clock cycle. and requires a lot of on-chip storage when processing large-scale computations of LLM, so a similar bitonic sorting network is not feasible in our scenario. In RTWS, it is also composed of basic modules, the compare and swap block (CSB). Its window comparison process is shown in the lower part of Fig. 7. This effectively saves on-chip storage resources.

V. EVALUATION

A. Methodology

For the accuracy evaluation of the quantization, our method is evaluated on LLaMA [2] and OPT [1] families with W4A4. We evaluate the perplexity on WikiText [29] and accuracy on four datasets [30]–[33]. DuoQ is compared with outlier-aware hardware methods [6], [7], [18], [19], as shown in Table I. We also compared it with the SOTA works of the algorithm design [8]–[11], as shown in Table II. And 128 random sampled sequences from [29] are used for calibration.

TABLE II
ZERO-SHOT ACCURACY OF QUANTIZED LLAMA MODELS

Model	Method	Accuracy↑					
		piqa	hella	wino	arc-e	arc-c	Avg.
LLaMA2-7B	fp16	79.11	76.00	69.53	74.33	46.25	69.04
	SmoothQuant [9]	51.90	27.05	48.38	25.63	26.37	35.87
	Atom [8]	75.14	69.37	69.37	52.99	38.40	59.73
	QuaRot [10]	76.17	72.3	64.40	68.06	41.47	64.48
	SpinQuant [11]	76.12	73.15	66.54	69.32	41.64	65.35
	DuoQ	76.87	73.12	69.38	71.11	42.23	66.54
LLaMA2-13B	fp16	80.52	79.37	72.22	77.48	49.06	71.73
	SmoothQuant	53.81	30.70	51.07	33.00	24.49	38.61
	Atom	76.50	73.84	67.40	57.49	42.32	63.51
	QuaRot	77.48	75.28	67.88	72.35	45.65	67.73
	SpinQuant	78.56	77.01	67.88	75.97	46.76	69.24
	DuoQ	79.33	76.91	71.13	74.88	47.11	69.87
LLaMA2-70B	fp16	82.54	84.31	79.48	80.30	56.91	76.71
	Atom	79.92	79.06	74.27	58.25	46.08	67.52
	QuaRot	82.43	81.82	76.24	80.43	56.23	75.43
	SpinQuant	81.61	82.36	75.85	79.17	56.31	75.05
	DuoQ	81.91	83.01	77.87	79.33	56.37	75.69
LLaMA3-8B	fp16	80.69	79.17	72.61	77.69	53.41	72.71
	SmoothQuant	51.09	28.90	50.36	28.96	23.55	36.57
	QuaRot	75.63	72.38	65.98	67.30	44.45	65.15
	SpinQuant	77.04	69.22	74.83	74.07	45.99	68.23
	DuoQ	79.12	74.31	73.29	77.91	51.48	71.22
LLaMA3-70B	fp16	84.49	84.90	80.51	85.90	64.33	80.03
	QuaRot	77.37	76.75	68.51	72.31	47.01	68.39
	SpinQuant	80.80	77.30	56.00	80.90	76.40	74.28
	DuoQ	84.04	83.77	76.77	80.92	70.15	79.13

For the hardware design part, we simulated and synthesized the RTL-based architecture on AMD Xilinx Vivado 2020.1 and implemented it on the Xilinx Alveo U280. Furthermore, Xilinx Power Estimator and switching activity interchange format file (SAIF) are jointly utilized to estimate the runtime power consumption of the design. SOTA-related FPGA accelerators [15], [16], [18] and other platforms are used as the baseline.

B. Accuracy Evaluation

We first report a comparison of DuoQ with other outlier-aware hardware on perplexity. As shown in Table I, DuoQ outperforms existing methods on various models. Olive and Oltron's methods for encoding outliers make it difficult to accurately identify all, which does not best preserve the model's accuracy. Omniquant smoothes the activation distribution but does not perform well on large models. HotaQ, while taking into account the perception of attention activation, has limited smoothing of the overall activation.

We conduct another software evaluation of the zero-shot accuracy of quantized LLMs in Table II. For the LLaMA2 (7B-70B) model, our model shows the highest accuracy in almost all datasets, with a maximum accuracy loss of only 1.86% compared to FP16. For the latest LLaMA3 (8B-70B), we have achieved an accuracy improvement of 2.99% and 4.85% compared to the SOTA working SpinQuant.

C. Analysis of PE Unit

Table III summarizes the DSP-packing methods considering different performance metrics. In terms of the supported data width, the DSP-packing method of our PE supports both the INT4 and INT8 operands, also the design is capable of multiplying both signed operands, crucial for the application's correctness. For the resource utilization with the implementation, our method claims the maximum resource conservation.

D. Analysis of System Performance

In this work, the proposed hardware architecture design with the W4A4KV4 high processing capabilities is evaluated

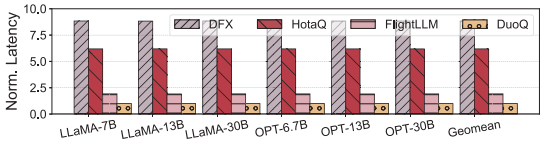
TABLE III
COMPARISON OF DIFFERENT DSP-PACKING METHODS

Method	Data Type	Signed Mult. Support	Utilization per Multiplication		
			DSP	LUT	FF
WP486 [34]	INT8	No	0.5	0	0
DoubleMAC [35]	INT8	Yes	0.5	11	12
ParallelDot [36]	INT8	Yes	0.125	24.63	25.13
WP521 [37]	INT4	Partial	0.25	0	0
DSP_Packing [24]	INT4	Partial	0.18	4.5	5.33
TransFRU [23]	INT8/INT4	Yes/Partial	0.5/0.125	7/6.25	8.5/5
Ours	**INT8/INT4/INT4**	**Yes/Yes/Partial**	**0.5/0.25/0.02**	**3.5/5.25/1.52**	**6/7.5/5**

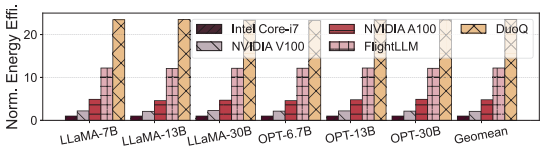
TABLE IV
FPGA IMPLEMENTATION RESULTS

Device	AMD Xilinx Alveo U280(16nm)
Frequency	150 MHz
Memory	8(DDR) & 32(HBM) GB

Resource	LUT	FF	DSP	BRAM
Total	484k (37.1%)	670k (18.5%)	8192 (90.7%)	668 (33.1%)



(a) Normalized latency



(b) Normalized energy efficiency

Fig. 8. Comparison of platforms built of different architectures.

with the available quantized INT4 LLM models evaluated in Section V-B, and the sequence length of 2048 is considered as the standard length of input and output tokens.

Table IV displays the implementation result of our design on Xilinx Alveo U280 operating at the frequency of 150 MHz. The runtime latency and the energy efficiency are reported in Fig. 8(a). Compared with other SOTA FPFA-based LLM accelerators [15], [16], [18], our work observes a minimum improvement of 1.9× and a maximum improvement of 8.8× in terms of the normalized runtime latency.

In consideration of our work's improvement of energy efficiency over other platforms. The power consumption processing of different LLM works under different architectures are recorded and analyzed in Fig. 8(b). A minimum of 1.9× and a maximum of 23.45× improvements are found compared to the FlightLLM FPGA and Intel Core-i7 CPU, respectively.

VI. CONCLUSION

In this work, we propose DuoQ, an FPGA-oriented algorithm-hardware co-design framework to enable high DSP utilization and the free outlier during LLM inference acceleration. In the algorithm optimization part, it introduces a hardware-friendly INT4 quantization to save the hardware cost while maintaining accuracy. In the hardware design part, an FPGA-based accelerator is built to enable the DSP's support for INT4 efficient multiplication. DuoQ achieves satisfactory accuracy among different datasets through the quantization scheme, averaging 66.54%, and the corresponding implemented accelerator improves the computation efficiency with a maximum of 8.8× and 23.45× for the energy efficiency.

REFERENCES

[1] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, and D.-K. Jeong, "Aˆ3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 328–341, 2020.

[4] M. Muller, A. Tyshka, M. Theisen, and D. Hanna, "Co-design of a tinyllm using programmable logic and software on an fpga," in *2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 253–257, 2024.

[5] P. Qi, Y. Song, H. Peng, S. Huang, Q. Zhuge, and E. H.-M. Sha, "Accommodating transformer onto fpga: Coupling the balanced model compression and fpga-implementation optimization," in *Proceedings of the 2021 Great Lakes Symposium on VLSI*, GLSVLSI '21, p. 163168, 2021.

[6] C. Xue, C. Zhang, X. Jiang, Z. Gao, Y. Lin, and G. Sun, "Oltron: Algorithm-hardware co-design for outlier-aware quantization of llms with inter-/intra-layer adaptation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, DAC '24, 2024.

[7] W. Shao, M. Chen, Z. Zhang, P. Xu, L. Zhao, Z. Li, K. Zhang, P. Gao, Y. Qiao, and P. Luo, "Omniquant: Omnidirectionally calibrated quantization for large language models," in *The Twelfth International Conference on Learning Representations*, 2024.

[8] Y. Zhao, C.-Y. Lin, K. Zhu, Z. Ye, L. Chen, S. Zheng, L. Ceze, A. Krishnamurthy, T. Chen, and B. Kasikci, "Atom: Low-bit quantization for efficient and accurate llm serving," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 196–209, 2024.

[9] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: accurate and efficient post-training quantization for large language models," in *Proceedings of the 40th International Conference on Machine Learning*, ICML'23, 2023.

[10] S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman, "Quarot: Outlier-free 4-bit inference in rotated llms," *arXiv preprint arXiv:2404.00456*, 2024.

[11] Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort, "Spinquant–llm quantization with learned rotations," *arXiv preprint arXiv:2405.16406*, 2024.

[12] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 811–824, 2020.

[13] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 692–705, 2021.

[14] W. Li, A. Hu, N. Xu, and G. He, "Quantization and hardware architecture co-design for matrix-vector multiplications of large language models," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 6, pp. 2858–2871, 2024.

[15] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "Dfx: A low-latency multi-fpga appliance for accelerating transformer-based text generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 616–630, 2022.

[16] S. Zeng, J. Liu, G. Dai, X. Yang, T. Fu, H. Wang, W. Ma, H. Sun, S. Li, Z. Huang, Y. Dai, J. Li, Z. Wang, R. Zhang, K. Wen, X. Ning, and Y. Wang, "Flightllm: Efficient large language model inference with a complete mapping flow on fpgas," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '24, pp. 223–234, 2024.

[17] H. Chen, J. Zhang, Y. Du, S. Xiang, Z. Yue, N. Zhang, Y. Cai, and Z. Zhang, "Understanding the potential of fpga-based spatial acceleration for large language model inference," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 18, no. 1, pp. 1–29, 2024.

[18] X. Shen, Z. Han, L. Lu, Z. Kong, P. Dong, Z. Li, Y. Xie, C. Wu, M. Leeser, P. Zhao, X. Lin, and Y. Wang, "Hotaq: Hardware oriented token adaptive quantization for large language models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[19] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, 2023.

[20] S. Ullah, S. Rehman, M. Shafique, and A. Kumar, "High-performance accurate and approximate multipliers for fpga-based hardware accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 2, pp. 211–224, 2022.

[21] A. C. Mert, H. Azgin, E. Kalali, and I. Hamzaoglu, "Efficient multiple constant multiplication using dsp blocks in fpga," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 331–3313, 2018.

[22] G. Yang, Y. Xie, Z. J. Xue, S.-E. Chang, Y. Li, P. Dong, J. Lei, W. Xie, Y. Wang, X. Lin, and Z. Fang, "Sda: Low-bit stable diffusion acceleration on edge fpgas," in *2024 34th International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 264–273, 2024.

[23] H. Wang, Y. Bai, J. Yu, and K. Wang, "Transfru: Efficient deployment of transformers on fpga with full resource utilization," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 521–526, 2024.

[24] J. Sommer, M. A. zkan, O. Keszocze, and J. Teich, "Dsp-packing: Squeezing low-precision arithmetic into fpga dsp blocks," in *2022 32nd International Conference on Field-Programmable Logic and Applications (FPL)*, pp. 160–166, 2022.

[25] M. Sun, X. Chen, J. Z. Kolter, and Z. Liu, "Massive activations in large language models," *arXiv preprint arXiv:2402.17762*, 2024.

[26] X. Chen and K. He, "Exploring simple siamese representation learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15750–15758, 2021.

[27] V. IsaacChassande, A. Evans, Y. Durand, and F. Rousseau, "Dedicated hardware accelerators for processing of sparse matrices and vectors: A survey," *ACM Trans. Archit. Code Optim.*, vol. 21, Feb. 2024.

[28] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: neural approximation of non-linear operations for efficient transformer inference," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC '22, p. 577582, 2022.

[29] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[30] Y. Bisk, R. Zellers, J. Gao, Y. Choi, *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 7432–7439, 2020.

[31] P. Clark, I. Cowhey, O. Etzioni, T. Khot, A. Sabharwal, C. Schoenick, and O. Tafjord, "Think you have solved question answering? try arc, the ai2 reasoning challenge," *arXiv preprint arXiv:1803.05457*, 2018.

[32] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi, "Hellaswag: Can a machine really finish your sentence?," *arXiv preprint arXiv:1905.07830*, 2019.

[33] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[34] Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep learning with int8 optimization on xilinx devices white paper (wp486)," 2024.

[35] D. Nguyen, D. Kim, and J. Lee, "Double mac: doubling the performance of convolutional neural networks on modern fpgas," in *Proceedings of the Conference on Design, Automation & Test in Europe*, DATE '17, p. 890893, 2017.

[36] M. Vstias, R. P. Duarte, J. T. de Sousa, and H. Neto, "Parallel dot-products for deep learning on fpga," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, 2017.

[37] T. Han, T. Zhang, D. Li, G. Liu, L. Tian, D. Xie, and Y. S. Shan, "Convolutional neural network with int4 optimization on xilinx devices (wp521)," 2024.

ADVISOR: Approximate Computing-friendly High-Level Synthesis Design Space Explorer

B. Parchamdar and B. Carrion Schaefer
The University of Texas at Dallas
Department of Electrical and Computer Engineering
{baharealsadat.parchamdar,schaferb}@utdallas.edu

***Abstract*—Approximate computing is a relatively new computing paradigm that allows to trade-off area/power with accuracy. Another relatively new VLSI design trend is to raise the level of design abstraction from the Register-Transfer Level (RTL) to the behavioral level and use High-Level Synthesis (HLS) to synthesize these behavioral descriptions. HLS has one unique advantage over RTL design. It completely decouples the functional description from the implementation details. This allows to design and verify the behavioral description once, but then generate a large number of functional equivalent hardware implementations of unique area vs. performance trade-offs. This is typically achieved through synthesis directives in the form of pragmas that the HLS user annotates at the source code to mainly control how to synthesize arrays (RAM, registers), loops (unroll, pipeline) and functions (inline or not).**

In this work we leverage this uniqueness and build an automated HLS design space explorer to find the hardware circuit most amenable to approximate computing, this is, has the highest potential for area/power savings. We have coined this explorer ADVISOR. The main problem with traditional exploration approaches is their long run time, which is worse in this case because every new implementation needs to be approximated to fully understand the trade-offs in terms of area/power vs. error. Thus, in order to accelerate this exploration process, we propose to evaluate each new designs based on an Approximation Friendliness Index (AFI) that can be computed statically, very fast, and only fully approximate the designs recommended by our flow that have high AFI values. Experimental results show that this approach leads to basically the same results as exhaustively approximating every new design, while being on average 68× faster.

***Index Terms*—High-Level Synthesis, Design Space Exploration, Approximate Computing.**

I. Introduction

Creating faster, smaller (cheaper) and lower-power integrated circuits (ICs) is paramount to continue supplying hardware devices that can accomplish more computationally intensive tasks while extending their battery life at smaller costs. This requires new VLSI design methodologies and computing paradigms that either complement or substitute traditional design and optimization techniques. Two of these methodologies that we address in this work are approximate computing and High-Level Synthesis (HLS).

Approximate computing leverages the fact that many applications tolerate errors at their output. These applications include Digital Signal Processing (DSP) and image processing applications. This can be leveraged to build smaller and/or lower power hardware implementations. This has become ever more relevant these days where modern ICs now include multiple hardware accelerators that specifically deal with these types of applications [1], [2].

At the same time HLS is often used to build these hardware accelerators which are typically first designed and verified using high-level languages like C, C++ or Matlab and then converted into efficient hardware implementations. HLS allows to directly synthesize these high-level descriptions into hardware (RTL). HLS has been shown to have many advantages over low-level Hardware Description Languages (HDLs) like reducing the turn-around time, allowing faster simulations and the ability to generate different types of hardware circuits from the same behavioral descriptions by simply setting different synthesis options [3].

This last advantage will be leveraged in this work to find an RTL implementation (micro-architecture) of a hardware accelerator that is amenable to approximate computing that has the highest potential for approximation. In particular we show that some implementations cannot be approximate at all (e.g., because of the high degree of resource sharing), while other implementations are very amenable to approximations. We call our proposed flow **ADVISOR**: **A**pproximate Computing-frien**D**ly High-Le**V**el Synthesis Des**I**gn **S**pace Expl**OR**er. In summary, the main contributions of this work are:

- Introduce an automated HLS design space explorer that has the goal of finding the most approximation friendly hardware implementations.
- Introduce an approximation friendliness index that can be quickly computed for RTL descriptions that significantly accelerates the design space exploration process.
- Present a comprehensive evaluation showing the effectiveness of our proposed flow.

II. Related Work

Many classes of applications exhibit significant tolerance to inaccuracies in their computations. Some examples include image processing, multimedia applications, and machine learning [1], [2], [4]. These inaccuracies can be exploited to build circuits with smaller areas, lower power, and higher performance. **Table I** highlights some of the most salient work, categorizing them based on: (1) The VLSI design stage where they are applied, e.g., HLS [5]–[10], logic synthesis [11]–[13] and physical design [14] or across all design stages [15]. (2) Specific approximate arithmetic unit (e.g., adders and multipliers), (3) approximate architectures like reconfigurable fabrics and approximate interconnects [16], [4], and (4) system-level optimizations [17], [18]. The main problem with these previously developed techniques is that they do not scale well and for larger circuits the runtime is extremely large without even the grantee that the circuit can be approximated.

TABLE I: Overview of the state of the art in approximate computing classified by VLSI design stage and component.

High-Level Synthesis: Joint precision [5], ABACUS [6], HLS schedule [7] AHLS [8], HLS real time [9], AxHLS [10] **Logic Synthesis:** SALSA [11], GALS [12] , Boolean network [13] **Physical Design:** FPGA wire removal [14]
Multi VLSI stage approximation: HLS to gate netlist [15]
Functional Units: Adders: [19], [20], Multipliers: [21]–[23]
Interconnect: AxBA [4], [16]
System-Level/Cross-Layer approximations: SoC: [17], [18]

To address this, the authors in [24] proposed a fast design space exploration for RTL descriptions to find quickly find Pareto-optimal configurations by intelligently assigning arithmetic functional to approximate versions. This previous work takes a given Verilog descriptions as input and hence the results are highly dependent on how amenable this architecture is to approximation.

Approximate computing can also be classified based on the type of approximation. Some approximation can be controlled dynamically at runtime like dynamic voltage overscaling where the supply voltage is reduced beyond V_{dd} [25], [26] and static approximations where the result of the approximation leads to a fixed hardware circuit (most common). Our work deals this latter type of approximations. To the best of our knowledge this is the first work that aims at leveraging the untimed behavioral description to quickly find the RTL descriptions that have the highest potential for approximation.

III. Background

This section covers important background information required to better understand the proposed contribution. In particular it will cover the basics of HLS and showcase some approximate computing primitives.

High-Level Synthesis: HLS converts untimed behavioral descriptions like C or C++ into RTL (Verilog or VHDL).

In order to allow users to generate a specific hardware architecture, HLS vendors make extensive use of synthesis directives in the form of pragmas to mainly control how to synthesize arrays, loops and functions. Arrays can be synthesized as registers or RAM, loops unrolled, partially unrolled, not unrolled or pipelined and functions inlined or not. Setting different combinations of these pragmas lead to hardware implementations of different area vs. performance trade-offs. In the code snippet in Fig. 1 if the array is synthesized as registers and the loop fully unrolled, then a hardware circuit of low latency but high area is obtained. In contrast, setting the pragma combinations to not unroll the loop and still synthesizing the array as registers leads to the smallest, but slowest implementation. The back end of the HLS process takes these results and outputs the synthesizable RTL.

As shown in Fig. 1, setting different pragma combinations leads to HW implementations of unique area vs. performance trade-offs. Because area and performance are conflicting design metrics there is no single HW implementation that dominates all of the others. Instead, out of all the pragma combinations there are some that lead to Pareto-optimal designs that form the Pareto-front ($\bar{P}$). The process of finding

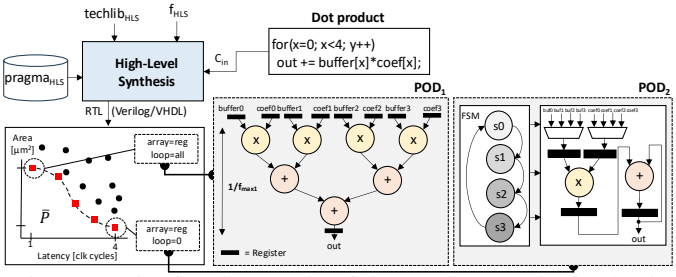


Fig. 1: Motivational example of dot product synthesized using different HLS pragma combinations. Pareto-optimal design (POD_1): Exposes 6 internal signals and 7 functional units (FUs). Pareto-optimal design (POD_2): Exposes 4 internal signal and 2 FUs.

these Pareto-optimal designs can be automated and is typically referred to as HLS design space exploration (DSE) [27]. Here, our goal is not to find the Pareto-optimal designs, but the design that is most amenable to approximate computing within a given area, performance and power constraints.

Approximate Computing: Approximate computing is based around a variety of different primitives that can be applied to different individual VLSI design stages like HLS [5], [6], [8], logic synthesis [11], [12] and physical design [14]. Some approximations are specific to a component in the SoC like on-chip interconnect [4] or arithmetic operations. These include approximated adders [19], [20], [24] and multipliers [21], [22] which have been widely studied due to their importance.

Fig. 2 continues the example shown in Fig. 1 where the design with highest performance (lowest latency) circuit implementation is further approximated. In this case we can see that the for-loop is fully unrolled in order to expose the four multiplications and three additions. Two different approximations are shown here. In POD_{apprx1}, the exact functional units (FUs) (adders and multipliers) are substituted by approximated versions. Based on the type of approximate FUs and the input data, different substitutions can lead to different area/power savings while leading to different output errors. In POD_{apprx2} we can observe other approximation primitives called Variable to Variable (V2V) or Variable to Constant (V2C) substitution. In V2C, If any signal in the circuit *hoovers* around a certain mean value with small standard deviation, then, this signal can be directly connected to a constant of that mean value. Additionally, if two signals are highly correlated one signal can be connected to another signal, directly or after re-sizing it. This is a V2V approximation.

As shown in Fig. 2 typically V2V and V2C approximations have a higher impact on the area/power reduction because they can eliminate entire portions of the circuit. As shown in the approximated circuits in Fig. 2, POD_{apprx2}, has only two multipliers and three adders and hence, is much smaller than POD_{apprx1}. This observation will be leveraged in this work to determine the order in which we approximate a design.

In the case of the smallest POD in Fig. 1 (POD_2), no approximation was possible (FU nor V2V or V2C) because the FSM cannot be approximated and the FUs execute different operations from the application (resource sharing). This

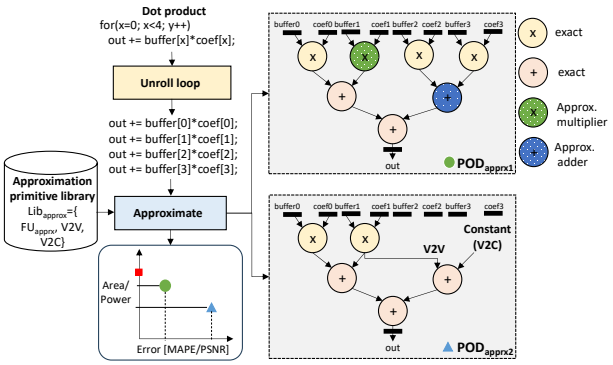


Fig. 2: Overview of approximate computing example including two different approximation primitives: Functional Unit approximation (FUs_{apprx}) and Variable to Variable and Variable (V2V) to Constant (V2C) approximations.

implies that they execute a larger variety of different data distributions, and hence, it is not possible to approximate these. Based on this, and without any loss of generality, we can make the following observations:

Observation 1: Designs with high amount of resource sharing are not good candidates for approximation. These designs include large FSMs which cannot be approximated and shared FUs which execute different operations with different data distributions on them.

Observation 2: Designs with larger amounts of parallelism contain a large number of FUs that only execute one operation and also have many internal signal have higher chances of being successfully approximated.

Observation 3: V2V and V2C approximations typically lead to larger area/power reductions as large portions of the circuits can be completely removed. Also, the closer the approximation is to the output the higher the savings are as larger parts of the circuit can be pruned.

IV. Proposed Flow

Fig. 3 shows an overview of the complete flow. The inputs to the proposed flow are the synthesizable behavioral description (C_{HLS}) for HLS, a library with synthesis directives for HLS to mainly control how to synthesize arrays, loops and functions ($pragma_{lib}$), the HLS technology library ($techlib_{HLS}$) and target synthesis frequency (f_{HLS}). In addition, a library of approximation primitives. In this work use V2V, V2C, and FU approximations, but other approximations like voltage over-scaling could also be included. Finally, the maximum allowable output error that this application tolerate (E_{max}) and the test vectors that represent the workload at which this design will be subject to (TV) also need to be given. Based on the type of application the E_{max} could be given as Mean Average Percentage Error (MAPE) or PSNR (for image processing applications).

The proposed flow is composed of two phases as shown in the figure. Phase I performs the approximation friendly HLS DSE and is further composed of three steps. Step 1 generates a new set of HLS synthesis directives, step 2 synthesizes the behavioral description with these new pragmas, and step 3 computes our newly proposed approximation friendliness index (AFI). These steps are repeated until a given exit condition is met (e.g., the AFI index is maximized). Phase II then approximates the design with highest AFI. The next subsections describe these phases in detail.

Phase I: Approximation friendly HLS Design Space Exploration

This first phase is the main contribution of this work. The idea is to quickly find an RTL implementation that has the maximum potential for approximation. As shown in the previous sections, some optimizations like resource sharing do not allow to approximate the circuit. Hence, this phase needs to filter out those design implementations and return other implementations with high potential for approximation. The alternative to this approach would be to generate a new RTL description for every new pragma combination, fully approximate it, and then evaluate its resultant area/power vs. output error. This is not only time consuming, but also not feasible for behavioral descriptions that have a larger number of unique pragma combinations. Hence, faster and more efficient approaches are needed.

Step 1: New Pragma Combination Generation: This first step starts by generating a random new pragma combination for the arrays, loops and functions in the given behavioral description (C_{HLS}). Although initially this pragma combination is random this phase updates a small portion of these pragmas through *fuzzing* in each iteration. Fuzzing has been widely used in verification and testing [28]–[30] where test vectors are modified (mutated) to detect hard to find bugs in hardware circuits. In the context of testing and verification the idea is to *fuzz* (update) the test vectors in order to find bugs and corner cases in the design. New test vectors are generated from one iteration to another by mutating (updating) a small number of test vectors between iterations. In the context of HLS DSE a pragma combination is equivalent to the test vectors and in every new iteration our flow mutates a small portion of the pragmas. Empirically we tried different mutation rates and found that updating 10% of the pragmas each iteration led to good results (quality of results and runtime). We used the following cost function to drive this process:

$$\begin{aligned} &maximize \quad Approximation\ Friendly\ Index(AFI) \\ &\text{subject to:} \quad A_{max}, L_{max} \end{aligned} \tag{1}$$

where the Approximation Friendly Index (AFI) estimates how amenable the circuit is to the approximations given in the approximation library (Lib_{approx}), and A_{max} and L_{max} are optional and represent the maximum area and latency that the final circuit should meet. The goal is to find the circuit that meets the given constraints with the highest possible AFI. The output of this step is the new behavioral description with the updated pragma combination (C_{pragma}).

Step 2: High-Level Synthesis: This next step takes as input the newly generated pragma combination (C_{pragma}) and synthesizes (HLS) if. The following inputs are needed by the HLS tool: C_{pragma}, $techlib_{HLS}$ and f_{HLS}. The results are the newly generated RTL code (RTL_{new}) and a report file

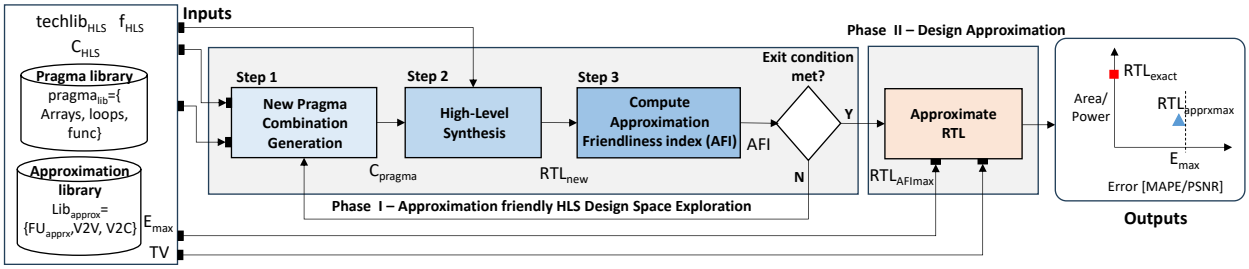


Fig. 3: Overview of proposed flow composed of 2 phases steps: **Phase I:** Approximation friendly HLS design space exploration composed of three step: **Step 1:** Generate synthesis directives for HLS. **Step 2:** Synthesize design (HLS). **Step 3:** Compute proposed approximation friendliness index (AFI). **Phase II:** Approximate RTL code.

that summarizes the area and performance of this hardware implementation.

Step 3: Compute Approximation Friendliness Index (AFI): This last step parses the newly generated RTL code (RLT_{new}) and computes the *Approximation Friendliness Index* (AFI). The AFI should have two main characteristics: (1) It should be easy and fast to compute and (2) it should correlate well with the potential of the RTL to be approximated.

Based on this The AFI is computed by statically parsing the RTL description and is highly dependent on the approximation primitives used (given as input to the flow). Table II highlights three cases based on three different approximation primitives. For the first case (V2V and V2C), we use internal signals. The intuition is that the more internal signals the circuit has, the higher the probability is that a V2V or V2C approximation is found. Larger number of internal signals also serves as a proxy for larger amount of parallelism and hence less resource sharing, which hampers approximations. Hence, for this first case our method parses the RTL code and counts the internal signals in the RTL as AFI. It should be noted that HLS generates designs that contains a FSM (controller) and a data path. Because the FSM cannot be approximated, we only consider the signals in the data path.

In the case of other approximation primitives like FUs approximations, the AFI simply counts the number of FUs in the newly generated RTL code. The more FUs, the higher the possibility of *finding* an approximation. Lastly when considering other approximations like voltage over-scaling, where the supply voltage (V_{dd}) is reduced beyond the nominal voltage, the AFI is proportional to the positive timing slack of the circuit, as voltage over-scaling reduces the switching speed of the transistors and hence, having a larger positive timing slack allows for higher voltage reductions. Table II shows an example of how to compute the AFI for the two implementations of the dot-product shown in Figure 1. POD_1 has a higher AFI index for the first two approximations, while smaller AFI in the timing slack case as compared to POD_2 as this last design does not chaining the FUs and hence, the register to register delay is smaller.

TABLE II: AFI predictor based on approximation primitive

AFI Predictor	Approximation	AFI POD_1/POD_2*
$\sum$ Signals in datapath	V2V, V2C	7/3
$\sum$ Functional units (FUs)	FUs	7/2
Positive timing slack	Voltage over-scaling	2.3/6.2†

*AFI computation example from Pareto-optimal examples from Figure 1
†Nangate 40nm technology and HLS frequency 100Mhz

In the case that of multiple approximation primitives , then the different predictive models are combined to form a single AFI predictor and weights on each predictor set to determine which approximation should have higher preference. This phase ends by sorting the explored designs based on AFI.

Phase II: Approximate RTL

This second phase takes as inputs the list of explored designs sorted by AFI (RTL_{AFImax}), the maximum allowable error threshold (E_{max}) and the test vectors (TV) that represent the typical workload of this circuit. It also requires the logic synthesis technology library ($techlib_{LS}$) in order to reveal the effect of the approximations on the final synthesized circuit. The output is the approximated RTL design that leads to the largest area savings ($RTL_{apprxmax}$) while leading to an error that is within the specified maximum error threshold (E_{max}). In this work we consider two approximation primitives: V2V/V2C and FUs approximations. Based on the discussion in section III and as shown in Fig. 2 some approximations typically lead to larger savings than others. E.g., a V2V approximation can remove entire parts of a circuit including FUs, while FUs approximations keep the original circuit, and only affect locally the approximated FU. Based on this our flow starts by executing the most aggressive approximations first (in this case V2V and V2C) and then *fine-tune* the process by approximating the remaining FUs.

Because in this work we only target two approximations, this phase is composed of 2 steps shown in Fig. 4. The next subsections describe these in detail:

Step 1: V2V and V2C Approximations: This phase takes as input the design found in phase 1 with the highest AFI (RTL_{AFImax}) and tries to find any V2V and V2C approximations such that the circuit can be simplified. For this, it starts by simulating it with the given TV. A VCD file dumping the simulation result including all of the internal signals is generated after the simulation. This generated VCD file is parsed and the mean (μ) and standard deviation (σ) computed for every internal signal in RTL_{AFImax}. We also compute the correlation between all the internal signals.

Next, our flow analysis the results and extract all possible approximation points. These are signals in RTL_{AFImax} that have the potential of being approximated through V2V or V2C

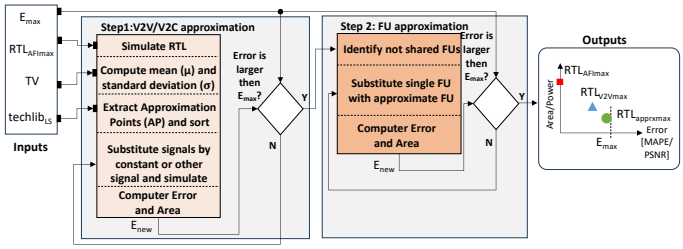


Fig. 4: Phase II Design Approximation overview of proposed flow composed of 2 phases.

approximations. For this we consider all of the signals with a standard deviation (σ) equal to E_{max} and correlation factor less than E_{max} %. At this stage we only select the signals that have the potential to approximate the circuit.

Our flow also traverses RTL_{AFImax} and sorts the approximations based on closeness to the primary outputs. Substituting a signal close to the outputs should intuitively lead to larger savings as a larger portion of the circuit can be pruned as opposed to approximating a signal closer to the primary inputs. Based on this, the flow proceeds to sort all of the approximation points (AP_{list}) based on their *closeness* the primary outputs. Our flow continues by setting the different approximations from AP_{list} following a greedy algorithm. Basically, the approximation closest to the primary outputs is enabled first (either V2V or V2C). A new RTL description is generated with this approximation (RTL_{apprx}). This RTL description is simulated, and the output error computed. This process is repeated by enabling the next closest approximation to the outputs in RTL_{apprx} while the measured error does not exceed E_{max}. If the measured error exceeds E_{max}, then the last approximation is removed and previous iteration RTL_{apprx} finally synthesized with the given logic synthesis technology library ($techlib_{LS}$). This will enable us to full characterize the area savings vs. output error. Note that our flow does not need to remove the unused logic as the logic synthesizer will automatically do this as it will eliminate any circuit part that does not affect the outputs. It should also be noted that if a V2V or V2C was successful then some of the approximation points extracted might not exist anymore as part of the circuit is removed during the approximation. These approximation points are therefore removed.

In order to compute the error, different error metrics have been proposed in the past [31], [32]. In this work we use the Mean Absolute Percentage Error ($MAPE$) for signal processing applications and Peak Signal to Noise Ratio ($PSNR$) for image processing applications, as PSNR is often used as a measure of quality in images [33]. $MAPE$ calculates the error between the output of the approximate circuit and the golden output from the exact solution as follows:$MAPE = \frac{1}{N}\sum_{i=1}^{N}|\frac{GO_i - APO_i}{GO_i}|$, where GO_i is the error-free output (golden output), and APO_i is the approximate output. PSNR on the other hand is computed here: $PSNR = 10\log_{10}\frac{MAX^2}{MSE}$, where MSE is the Mean Squared Error from the noise-free image and MAX the maximum pixel value of the original image.

The maximum allowable error is domain specific and hence, requires domain expertise. In our case we set the maximum error (E_{max}) to typical values reported in approximate computing literature [1], [2], [4] and study the robustness of our approach when the E_{max} changes. These include MAPE=10%, PSNR=20db and MAPE=20%, PSNR 10db.

The result of this step is the approximated RTL description RTL_{v2vmax}.

Step 2: FU Approximation: This next step takes as input the approximated RTL code and continues the approximation process by approximating only the FUs.

For this, this step starts by parsing RTL_{v2vmax} and extracting the FUs that are not shared. This is done by checking if the FUs' input is connected to muxes or not. As mentioned before, FU sharing does not allow FU approximation as multiple operations are mapped onto the same FU. Once these FUs are identified, our method starts by substituting iteratively following a greedy algorithm the largest FUs in the list. For this we use EvoApproxLib [34] an open-source library of approximated adders and multipliers.

These two step can be executed for only the RTL design with highest AFI or it can be repeated for the N designs with highest AFI. In the experimental results section we execute both scenarios where in the case of repeating this phase we do this until no smaller design withing E_{max} can be found.

The final output of the smallest design, $RTL_{apprxmax}$, with an error within E_{max}.

TABLE III: Benchmark characteristics

	Bench	Lines	Mul	Add/Sub	Input Stimuli	Error
DSP	fir	90	8	8	10,000 Random	MAPE
	fft	193	16	88	10,000 Random	MAPE
	interp	207	24	24	10,000 Random	MAPE
	decim	220	30	81	10,000 Random	MAPE
Image	sobel	70	0	45	512x512 BMP	PSNR
	idct	362	103	164	512x512 BMP	PSNR

V. Experimental Results

Six computationally intensive applications amenable to approximate computing were used to test our proposed method. These designs were taken from the open source Synthesizable SystemC Benchmark suite (S2CBench) [35]. Table III shows all the applications and highlights their main characteristics in terms of the number of lines of code, functional units required when the loops are fully unrolled (default HLS mode). The benchmarks can be coarsely classified into digital signal processing (fir, fft, decimation and interpolation filter) and image processing (sobel, idct). These families of designs have shown to be inherently error resilient.

Table IV summarizes the experimental setup in terms of the HLS and LS tool used and the server where we ran the experiments.

Three approaches were compared in this work. In the first one we use the same fuzzing-based design space exploration, but fully approximate each design generated instead of relying on the AFI index to determine which design to approximate. The cost function used in this case is not the

TABLE IV: Experimental Setup

HLS Tool	NEC CyberWorkBench 10.1 [36]
Logic Synthesis	Synopsys Design Compiler 2018.06-SP1
RTL Simulator	Synopsys VCS 2018.06
Synthesis Technology	Nangate 45nm Open Cell Library
Frequency (HLS and LS)	100 MHz
CPU	Intel Core i7-4790 CPU @3.6GHZ
Memory	16 GB DDR3
Operating System	CentOS v8.0

AFI index, but the actual area savings obtained after the approximation flow (phase 2) is fully executed for each design. This approach represents the naïve approach (**NAIVE**). In the second and third approach we use our proposed approach. In the first case we only approximate the design with highest AFI (**ADVISOR(AFI=1)**) , while in the second approach we approximate the N designs with highest AFI (**ADVISOR(AFI=N)**) until a design with smallest area cannot be found. In all cases, phase I exits if after 10 consecutively generated designs, the area savings or AFI cannot be improved for the **NAIVE** and **ADVISOR** approach respectively.

Fig. 5 and Table V summarize the area savings and speedup obtained by the different approaches for different maximum error thresholds (E_{max}). Figs. 5(a) show the results in the unconstraint case, when no area (A_{max}) and latency (L_{max}) constraint are set and Figs. 5(b) when a constraint is set to half of the area of the largest design and half the latency of the slowest design. This will help us to fully characterize our approach under different scenarios. From the results we can make the following observations:

Observation 1: All of the benchmarks could be successfully approximated. For the unconstraint case, on average the area reduction was 48% and 52% for the **ADVISOR(AFI=1)** and **ADVISOR(AFI=N)** and MAPE=10%, PSNR=20db case respectively. The area reduction was approximate 10% higher to 59% and 66% when E_{max} was further relaxed to MAPE=20%, PSNR=10db. Moreover, The results obtained by our approach are very similar to the **NAIVE** approach, in particular in the **ADVISOR(AFI=N)** case.

Observation 2: Approximating only the design with highest AFI index leads to very good results, but on average 2% and 3% worse then continuing to approximate the N designs with highest AFI. On average, the five designs with highest AFI were approximated (N=5).

Observation 3: The unconstraint exploration case leads to much better results then the constraint case. On average the area reduction worsen between 31 to 37% for the different E_{max} and ADVISOR approaches. This is mainly due to the fact that limiting the search space to generate a smaller circuits forces the explorer to discard circuits with small or no resource sharing We noticed that in for this case very few V2V and V2C approximations where found and that almost all of the area reduction is due to FU approximations.

Observation 4: From Table V we can observe that our flow is on average 66× and 25× faster than the **NAIVE** approach for **ADVISOR(AFI=1)** and **ADVISOR(AFI=N)** respectively,

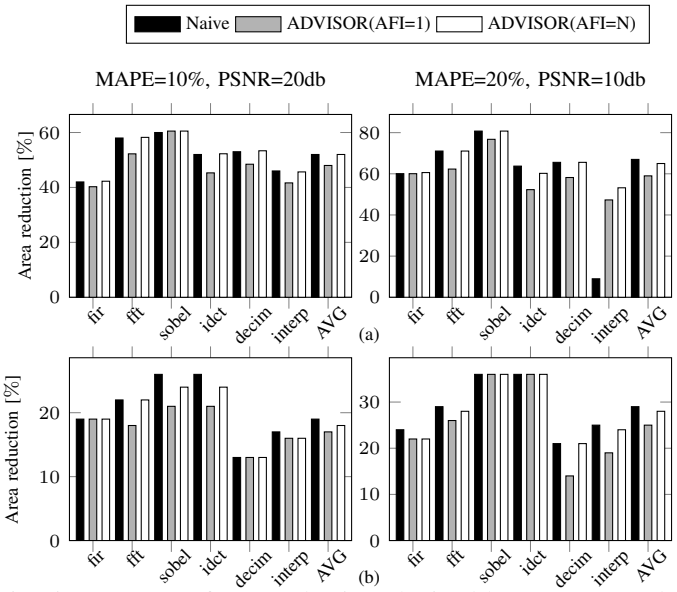


Fig. 5: Summary of area reduction obtained by our proposed flow (**ADVISOR**) vs. Naive search (**NAIVE**); (a) when no Area (A_{max}) and Latency (L_{max}) constraints are set; (b) when maximum area and latency constraint are set to half the max Area and Latency for different E_{max}.

while leading to similar results. This is one of the main contributions of this work, and we expect even higher speedups as more approximation primitives are added to the process.

TABLE V: Runtime comparison of NAIVE vs. proposed flow

Benchmark	NAIVE	ADVISOR (AFI=1)	ADVISOR (AFI=N)
	Runtime [h:mm]		
fir	7:45	0:06	0:14
fft	13:07	0:12	0:33
decimation	23:12	0:25	0:59
interpolation	26:56	0:37	1:33
sobel	19:23	0:13	0:43
idct	21:12	0:15	0:45
Geomean	**16:53**	**0:15**	**0:41**

Based on these results we believe that the proposed fully automated flow is very efficient finding the most amenable RTL design to approximate computing given an untimed behavioral description for HLS.

VI. Conclusion

In this work, we have introduced an automated High-Level Synthesis design space explorer that has the goal of finding the RTL implementation with the highest potential for approximation. Previous work on RTL approximation assumes that the RTL is given beforehand. Fortunately, HLS has the ability to generate a larger number of functional equivalent circuits with different hardware implementations (RTLs). We leverage this fact and propose a very simply approximation friendly index to find these designs. Future work includes adding more approximation primitives.

Acknowledgment

This work is partially supported by the NSF through award number #2323820.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[2] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[3] B. Carrion Schaefer, *High-Level Synthesis Made Easy*, 1st ed. highX Technlogies, 2023. [Online]. Available: https://www.hlsbook.com

[4] F. Betzel *et al.*, "Approximate communication: Techniques for reducing communication bottlenecks in large-scale parallel systems," *ACM Comput. Surv.*, vol. 51, no. 1, Jan. 2018.

[5] C. Li, W. Luo, S. S. Sapatnekar, and J. Hu, "Joint precision optimization and high level synthesis for approximate computing," in *DAC*, June 2015, pp. 1–6.

[6] K. Nepal, Y. Li, R. I. Bahar, and S. Reda, "ABACUS: A technique for automated behavioral synthesis of approximate computing circuits," in *DATE*, March 2014, pp. 1–6.

[7] S. Xu and B. Carrion Schafer, "On the design of high performance hw accelerator through high-level synthesis scheduling approximations," in *DATE*, 2020, pp. 1378–1383.

[8] S. Lee, L. K. John, and A. Gerstlauer, "High-level synthesis of approximate hardware under joint precision and voltage scaling," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 187–192.

[9] M. T. Leipnitz and G. L. Nazar, "High-level synthesis of approximate designs under real-time constraints," *ACM Trans. Embed. Comput. Syst.*, vol. 18, no. 5s, Oct. 2019. [Online]. Available: https://doi.org/10.1145/3358182

[10] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel, "Axhls: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

[11] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *DAC*, 2012, pp. 796–801.

[12] J. Miao, A. Gerstlauer, and M. Orshansky, "Approximate logic synthesis under general error magnitude and frequency constraints," in *ICCAD*, 2013, pp. 779–786.

[13] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *DAC*, 2016, pp. 1–6.

[14] Y. Wu, C. Shen, Y. Jia, and W. Qian, "Approximate logic synthesis for fpga by wire removal and local function change," in *ASP-DAC*, Jan 2017, pp. 163–169.

[15] S. Xu and B. Carrion Schafer, "Exposing approximate computing optimizations at different levels: From behavioral to gate-level," *IEEE TVLSI*, vol. 25, no. 11, pp. 3077–3088, 2017.

[16] J. R. Stevens, A. Ranjan, and A. Raghunathan, "Axba: An approximate bus architecture framework," in *ICCAD*, 2018, pp. 1–8.

[17] A. Agrawal, J. Choi, K. Gopalakrishnan, S. Gupta, R. Nair, J. Oh, D. A. Prener, S. Shukla, V. Srinivasan, and Z. Sura, "Approximate computing: Challenges and opportunities," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, 2016, pp. 1–8.

[18] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *DAC*, 2016, pp. 1–6.

[19] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," *IEEE TCAD*, vol. 32, no. 1, pp. 124–137, 2013.

[20] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, "A low latency generic accuracy configurable adder," in *DAC*, 2015, pp. 1–6.

[21] C. Liu, J. Han, and F. Lombardi, "A low-power, high-performance approximate multiplier with configurable partial error recovery," in *2014 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2014, pp. 1–4.

[22] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, "A comparative evaluation of approximate multipliers," in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, 2016, pp. 191–196.

[23] Y. Guo, Q. Zhou, and H. Sun, "High-efficiency fpga-based approximate multipliers with lut sharing and carry switching," in *2024 DATE*, 2024, pp. 1–6.

[24] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *DAC*, 2019, pp. 1–6.

[25] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: Enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, 2018, pp. 1–6.

[26] D. Mohapatra, "Approximate computing: Enabling voltage over-scaling in multimedia applications," Ph.D. dissertation, Purdue University, 2011.

[27] B. C. Schafer and Z. Wang, "High-level synthesis design space exploration: Past, present, and future," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2628–2639, 2020.

[28] R. Dutra, R. Gopinath, and A. Zeller, "Formatfuzzer: Effective fuzzing of binary file formats," vol. 33, no. 2, 2023.

[29] C. Zhang and J. Chen, "Fuzzing methods recommendation based on feature vectors," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1079–1081.

[30] R. Saravanan and S. M. Pudukotai Dinakarrao, "The fuzz odyssey: A survey on hardware fuzzing frameworks for hardware design verification," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, ser. GLSVLSI '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 192–197. [Online]. Available: https://doi.org/10.1145/3649476.3658697

[31] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Quality control for approximate accelerators by error prediction," *IEEE Design Test*, vol. 33, no. 1, pp. 43–50, Feb 2016.

[32] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, Sept 2013.

[33] R. Dosselmann and X. D. Yang, "Existing and emerging image quality metrics," in *Canadian Conference on Electrical and Computer Engineering*, May 2005, pp. 1906–1913.

[34] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017.

[35] B. Carrion Schafer and A. Mahapatra, "S2CBench:Synthesizable SystemC Benchmark Suite," *IEEE Embedded Systems Letters*, vol. 6, no. 3, pp. 53–56, 2014.

[36] NEC CyberWorkBench, 2019. [Online]. Available: www.cyberworkbench.com

GPart: A GNN-Enabled Multilevel Graph Partitioner

Magi Chen and Ting-Chi Wang
Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan

***Abstract*—This paper introduces GPart, a scalable multilevel framework for graph partitioning that integrates GNN embeddings with efficient coarsening and refinement techniques. On the Titan23 benchmarks, GPart achieves a cut size reduction of 34.13% to 42.92% over METIS and improves cut size by 9.30% on selected DIMACS benchmarks compared to G-kway. Furthermore, experiments on the Titan23 benchmarks show that GPart reduces normalized memory usage by 24.6x compared to GAP and 12.4x compared to GenPart. Unlike existing GNN-based methods, which require large hidden layers and substantial memory, GPart's multilevel architecture reduces hidden layer sizes, significantly optimizing memory efficiency.**

***Index Terms*—graph neural network, graph partitioning**

I. Introduction

Graph partitioning is a fundamental problem with extensive applications across various domains, including VLSI design, social network analysis, and scientific computing. Effective partitioning optimizes the allocation of graph components to different partitions, which reduces communication overhead and enhances computational efficiency. However, graph partitioning is an NP-hard problem, making it challenging to achieve both high-quality cuts and balanced partitions, especially as graph sizes continue to grow. In the VLSI domain, optimized graph partitioning plays a crucial role in dividing complex circuit designs into manageable subcomponents, facilitating efficient and parallel processing.

The multilevel approach has been successful in graph partitioning by decomposing the problem into coarsening, initial partitioning, and uncoarsening phases. METIS [2], a notable method, introduced the heavy edge matching (HEM) technique for efficient graph coarsening. To handle larger graphs, GPU acceleration has been leveraged to enhance scalability. GPU-based methods like G-kway [5] and [7] introduce parallelized coarsening mechanisms, demonstrating that GPUs can scale graph partitioning without compromising quality.

Spectral techniques have also been effective in capturing global structural properties for graph partitioning. Methods utilizing the Fiedler vector [16] and higher-order eigenvectors [15] have shown improved partitioning quality, including multi-way partitioning. For hypergraph partitioning, hypergraphs are converted into clique expansion graphs to enable spectral methods [14], [17], [18]. Recent works like SpecPart [17] and K-SpecPart [18] incorporate supervised hints to optimize vertex embeddings. However, spectral methods face scalability challenges due to increasing computational complexity with the number of eigenvectors required, especially for multi-way partitioning tasks.

To address these challenges, multilevel spectral coarsening strategies have been explored. GraphZoom [19] introduced approximate calculations to reduce computational costs while preserving key spectral properties. MedPart [20] adapted similar techniques for 2-way hypergraph partitioning. Nonetheless, these methods primarily focus on approximating the Fiedler vector and do not effectively extend to higher-order eigenvectors, limiting their applicability for k-way partitioning when $k \geq 3$.

Graph Neural Network (GNN)-based techniques have emerged as another approach. GAP [1], [6] uses an autoencoder-based architecture with GraphSAGE [8] layers and a normalized cut loss function for unsupervised learning, achieving competitive partitioning quality. GenPart [9] extends this framework to hypergraphs. Despite these advancements, GNN-based methods struggle with high computational complexity and memory usage, especially as the depth and width of hidden layers increase, posing difficulties in scaling to larger graphs.

To overcome these limitations, we propose **GPart**, a novel multilevel graph partitioner that integrates GNN embeddings with efficient coarsening and refinement techniques. GPart builds upon traditional methods and introduces innovative strategies to enhance performance while reducing computational and memory costs. Our contributions are summarized as follows:

1) We introduce GPart, the first multilevel graph partitioning framework that integrates learnable coarsening guided by GNN-generated node embeddings. Through *unsupervised* training with a carefully designed loss function, GPart captures essential structural properties by approximating both the Fiedler vector and higher-order eigenvectors. Its multilevel approach significantly reduces memory usage compared to existing GNN-based partitioners like GAP and GenPart, making it suitable for large-scale graphs.
2) GPart introduces a centroid-guided, gain-based refinement strategy during the uncoarsening phase. By leveraging learned node embeddings to compute weighted centroids for each partition, GPart identifies candidate nodes whose current assignments differ from their nearest centroids in the embedding space. It then selectively reassigns high-gain candidates—those offering the greatest improvement in cut size—while limiting the number of moves per level to ensure computational efficiency and maintain balance constraints.
3) Experimental evaluations on the Titan23 [13] and DIMACS [11] benchmarks demonstrate GPart's capacity to achieve notable improvements in cut size and competitive execution times over METIS. Additionally, experiments on the Titan23 benchmarks demonstrate that GPart reduces normalized memory usage by 24.6x compared to GAP and 12.4x compared to GenPart.

The remainder of this paper is organized as follows: Section II illustrates the problem formulation for graph partitioning. Sections III and IV describe the proposed approach and details. Section V presents our experimental results and analyzes the improvements achieved. Finally, Section VI concludes the paper.

II. Problem Formulation

A graph $G(V, E)$ consists of a vertex set V and an edge set E, where each edge $e \in E$ connects a pair of vertices in V. Each vertex v is assigned a weight w_v, and each edge e carries a weight w_e.

Given a graph $G(V, E)$, a positive integer $k \geq 2$, and a balance factor $0 < \varepsilon \ll 1$, the k-way graph partitioning problem is to partition the vertex set V into a collection of k disjoint subsets $P = \{P_0, P_1, \ldots, P_{k-1}\}$, where each P_i represents a partition. The total weight of the vertex set is denoted by $W = \sum_{v \in V} w_v$. The goal is to minimize the cut size, defined as:

$$\text{cutsize}(P) = \sum_{e \in \{(u,v) \in E \mid u \in P_i,\, v \in P_j, \text{ and } P_i \neq P_j\}} w_e,$$

subject to the following balance constraint:

$$\sum_{v \in P_i} w_v \leq (1+\varepsilon)\frac{W}{k}, \quad \text{for } 0 \leq i \leq k-1.$$

To simplify the presentation and analysis, this paper assumes uniform weights for vertices and edges, i.e., $w_v = 1$ for all $v \in V$ and $w_e = 1$ for all $e \in E$.

III. Overview of GPart

GPart is an efficient framework for graph partitioning, incorporating modules for embedding, coarsening, uncoarsening (projection), and refinement (See Figure 1). Unlike end-to-end GNN-based methods, GPart employs a modular approach with separate training and inference processes, enhancing both memory efficiency and partitioning quality.

In the **Training Phase**, GPart employs multilevel coarsening with GNN-based embeddings to guide the merging of nodes and minimize cut size. Afterward, embeddings are compressed and refined, preparing them for partitioning.

In the **Inference Phase**, the trained embedding and coarsening layers are reused, with an additional centroid-guided, gain-based refinement process to further improve cut quality while keeping the balance constraint. This approach identifies nodes with high potential for enhancement, enabling selective adjustments to the partitions based on centroid proximity. The inference process iteratively refines partitions until the original graph granularity is restored, yielding the final partitioning solution.

IV. Details of GPart

In this section, we describe the components of GPart's training and inference frameworks, providing a detailed breakdown of each module. Additionally, we explain the loss functions utilized during training to enhance model performance.

A. Details of Training Framework

1) Embedding Layers: The embedding layers in GPart utilize a three-layer GraphSAGE architecture [8] to generate high-dimensional node embeddings, which serve as the foundation for the subsequent stages in the partitioning process. GraphSAGE is specifically chosen for its inductive learning capabilities, enabling the model to generalize effectively across different graph structures by aggregating information from node neighborhoods.

The initial feature of each node in the input graph is derived from a unique sequence ID (0,1,2,...), assigned to each node, which is then standardized by subtracting the mean and dividing by the standard deviation. These standardized IDs serve as the input to the first layer of the embedding layers.

The embedding layers are structured as three consecutive GraphSAGE layers, with input and output dimensions of $1 \to 16$, $16 \to 16$, and $16 \to 16$, respectively. Each layer employs a mean aggregation function, allowing the model to capture increasingly higher-order neighborhood information with each layer. This progressively enriched feature representation enables the embeddings to reflect both local and broader structural contexts within the graph. The feature update equation for each layer t is as follows:

$$f_i^{(t)} = \sigma\left(W^{(t)} \cdot \text{AGGREGATE}\left(\{f_j^{(t-1)}, \forall j \in \mathcal{N}(i)\}\right) + W_{\text{self}}^{(t)} \cdot f_i^{(t-1)}\right)$$

Here, $f_i^{(t)}$ represents the embedding of node i at layer t, $\mathcal{N}(i)$ denotes the neighbors of node i, and σ is a nonlinear activation function (in this work, tanh). The parameters $W^{(t)}$ and $W_{\text{self}}^{(t)}$ are learned weight matrices for neighborhood and self-feature transformation, respectively. This update allows each node to incorporate information from its immediate neighbors, progressively capturing higher-order neighborhood structures as the layers deepen.

The three-layer GraphSAGE model progressively enriches each node's feature representation by capturing higher-order neighborhood information while maintaining a consistent 16-dimensional hidden layer size. The final layer normalizes embeddings to unit length, aligning them with the unit-norm property of eigenvectors. This normalization encourages the embeddings to capture structural information akin to Laplacian eigenvectors, enhancing compatibility with spectral-based clustering and coarsening techniques.

2) Coarsening Module: The Coarsening Module in GPart employs a multilevel coarsening approach, allowing a reduction in GNN layer sizes and thus significantly decreasing memory usage. By iteratively simplifying the graph, this approach enables efficient computation while preserving essential structural properties, making it well-suited for large-scale graph partitioning tasks.

GPart's coarsening process leverages Graclus [3], a GPU-accelerated clustering algorithm that performs greedy matching to group nodes based on edge weights. Graclus aggregates neighboring nodes with strong connections to form compact, representative clusters. Typically, each level of coarsening reduces the number of nodes by approximately half, enabling an exponential decrease in graph size across multiple levels.

In GPart, the coarsening process is further refined with a similarity-based weighting mechanism, where edge weights are adjusted according to node embedding similarity. Specifically, the weight w_{uv} of an edge between nodes u and v is modified as follows:

$$w_{uv} = \text{sigmoid}(-\|\mathbf{x}_u - \mathbf{x}_v\|_2) \times \text{original_weight}_{uv}$$

where $\mathbf{x}_u$ and $\mathbf{x}_v$ are the embeddings of nodes u and v, and $\|\mathbf{x}_u - \mathbf{x}_v\|_2$ represents their Euclidean distance. This distance-based similarity score, modified by a sigmoid function, enhances edge weights for closely connected nodes, guiding the matching process toward forming clusters that are both structurally cohesive and feature-similar.

As nodes are merged into clusters, their inter-cluster edge weights are aggregated to reflect the combined connectivity strength, while each cluster's size is tracked to support the initial partitioning phase. This aggregation results in a coarsened graph representation with weighted edges and node sizes, ensuring respect for cluster sizes during partitioning.

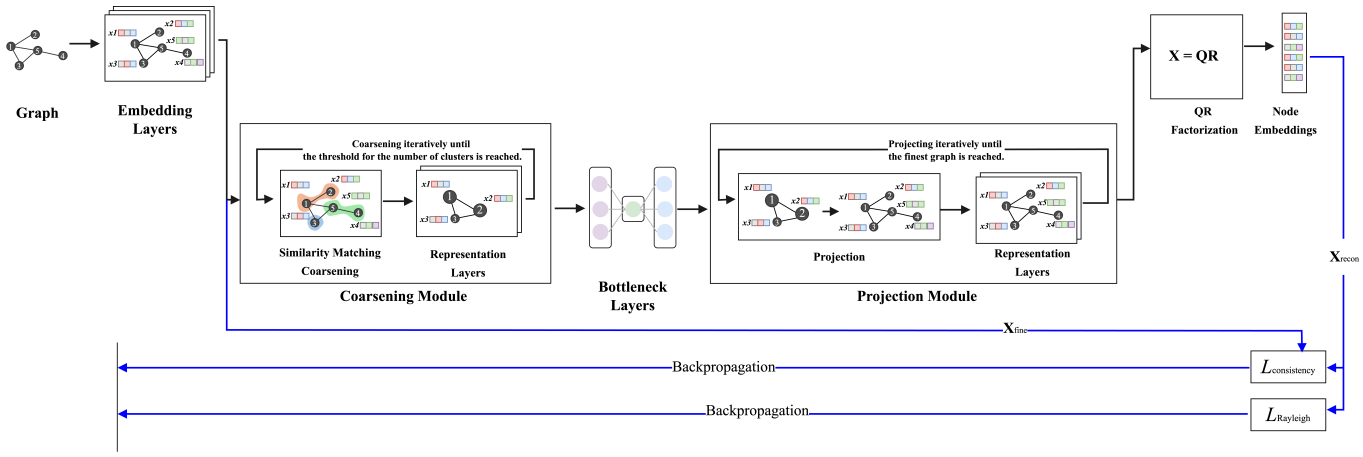
Within the Coarsening Module, **Representation Layers** are applied to refine embeddings at each coarsening level. These layers consist of two consecutive GraphSAGE layers with dimensions $16 \to 16$.

The coarsening process continues until the graph reaches a predefined size threshold, typically set to around $\frac{1}{20}$ of the original node count, following the G-kway strategy [5] and facilitating direct comparison with G-kway in later evaluations.

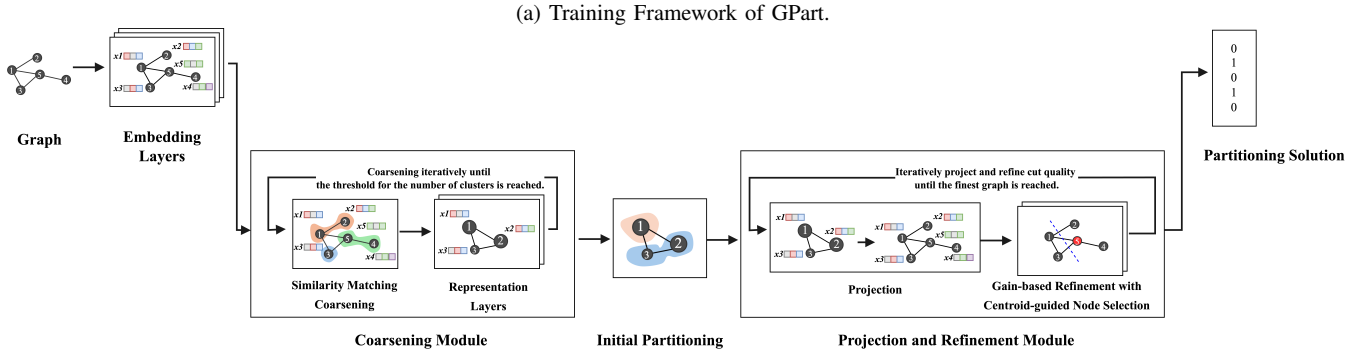
3) Bottleneck Layers: The Bottleneck Layers in GPart serve as a compression and refinement mechanism for node embeddings, applied exclusively during the training phase.

The Bottleneck Layers are structured with four linear layers, designed with the following dimensional configuration: $16 \to 8$, $8 \to 8$, $8 \to 8$, and $8 \to 16$. The bottleneck effect of these layers plays a crucial role in helping the coarsening module learn more focused and meaningful embeddings.

4) Projection Module: The Projection Module in GPart is responsible for reconstructing the original graph structure from the



(a) Training Framework of GPart.



(b) Inference Framework of GPart.

Fig. 1: The training and inference frameworks of GPart, illustrated with a 2-way partitioning example. The training framework includes embedding layers, coarsening module, bottleneck layers, projection module and QR factorization to generate node embeddings. The blue lines represent the backpropagation flow, applying consistency and Rayleigh losses to optimize embeddings through coarsening and projection modules. In the inference framework, the trained embedding layers and coarsening module are reused, followed by initial partitioning, iterative projection and refinement to produce the final partitioning solution.

coarsened representation, enabling refined and accurate final partitioning. This module utilizes a GPU-accelerated uncoarsening process to iteratively reverse the coarsening steps, projecting embeddings from the simplified graph back to finer levels while maintaining computational efficiency. During the training phase, the Projection Module focuses solely on restoring the graph's structure and refining embeddings, without performing any cut quality refinement. The refinement of cut quality is reserved exclusively for the inference phase.

The projection process begins with the embeddings from the deepest level of the coarsening hierarchy. Using the cluster mappings recorded during coarsening, the Projection Module "unpools" the embeddings back to finer graph levels. Mathematically, given the embeddings $\mathbf{X}^{(l)}$ at coarsening level l, the unpooling process maps these embeddings to the next finer level $l-1$ based on cluster information. Let $\mathbf{M}^{(l-1)}$ be the mapping matrix from clusters at level l to nodes at level $l-1$. The embeddings at level $l-1$ are obtained as:

$$\mathbf{X}^{(l-1)} = \mathbf{M}^{(l-1)}\mathbf{X}^{(l)}$$

where $\mathbf{M}^{(l-1)}$ is a mapping matrix that associates each node in the finer graph at level $l-1$ with the embedding of its corresponding cluster from the coarser graph at level l.

To refine the embeddings during this projection process when training, the Projection Module employs **Representation Layers**, consisting of two GraphSAGE layers with dimensions $16 \rightarrow 16$ each.

This iterative projection process continues until the finest granularity of the graph is reached, effectively restoring the input graph structure and embedding resolution. QR factorization is then applied to the embeddings to enforce orthogonality.

B. Loss Functions

After the QR factorization, GPart employs Rayleigh Quotient-based loss functions during training to ensure that the node embeddings capture essential spectral properties of the graph. These loss functions guide the learning process, promoting effective encoding of both local structural details and broader spectral characteristics crucial for graph partitioning.

1) Spectral Consistency Loss: The **Spectral Consistency Loss** is designed to preserve the spectral properties of the graph in the node embeddings produced by the embedding layers (referred to as the fine graph) and the reconstructed graph after coarsening and uncoarsening. Since the Laplacian matrix $\mathbf{L}$ of the fine graph is identical to that of the reconstructed graph, this loss compares the Rayleigh quotients of the node embeddings from both graphs using the same Laplacian matrix.

The Rayleigh quotient for a node embedding vector $\mathbf{x}$ with respect to a Laplacian matrix $\mathbf{L}$ is defined as:

$$R(\mathbf{x}, \mathbf{L}) = \frac{\mathbf{x}^\top \mathbf{L}\mathbf{x}}{\mathbf{x}^\top \mathbf{x}}. \tag{1}$$

Here, $\mathbf{x}^\top$ denotes the transpose of $\mathbf{x}$.

Let $\mathbf{x}_{\text{fine}}$ and $\mathbf{x}_{\text{recon}}$ represent the node embeddings of the fine and reconstructed graphs, respectively. The Spectral Consistency Loss is then defined as:

$$\mathcal{L}_{\text{consistency}} = \text{MSE}\left(R(\mathbf{x}_{\text{fine}}, \mathbf{L}),\ R(\mathbf{x}_{\text{recon}}, \mathbf{L})\right), \tag{2}$$

where MSE denotes the Mean Squared Error. This loss function minimizes the discrepancy between the spectral characteristics of the node embeddings from the fine and reconstructed graphs. By aligning their Rayleigh quotients, we ensure that the embeddings preserve important structural information during coarsening and uncoarsening.

2) Comprehensive Rayleigh Quotient Alignment Loss: The **Comprehensive Rayleigh Quotient Alignment Loss** is a composite loss comprising several components, each targeting a specific aspect of spectral alignment:

- **Spectral Residual Loss**: This component encourages the embeddings to align with the eigenvectors of the Laplacian matrix $\mathbf{L}$. It minimizes the residual between the Laplacian-transformed embedding $\mathbf{L}\mathbf{x}_{\text{recon}}$ and the embedding scaled by its Rayleigh quotient $R(\mathbf{x}_{\text{recon}}, \mathbf{L}) \cdot \mathbf{x}_{\text{recon}}$. This is based on the eigenvalue equation $\mathbf{L}\mathbf{x} = \lambda\mathbf{x}$, where λ is the eigenvalue.
  The Spectral Residual Loss is defined as:
  $$\mathcal{L}_{\text{residual}} = \text{MSE}\left(\mathbf{L}\mathbf{x}_{\text{recon}},\ R(\mathbf{x}_{\text{recon}}, \mathbf{L}) \cdot \mathbf{x}_{\text{recon}}\right). \tag{3}$$
  By minimizing this residual, we encourage the embeddings to approximate the eigenvectors of $\mathbf{L}$, aligning them with the graph's spectral properties.
- **Fiedler Vector Approximation Loss**: The Fiedler vector corresponds to the eigenvector associated with the second smallest eigenvalue of $\mathbf{L}$ and is crucial for capturing the graph's community structure. By minimizing the mean Rayleigh quotient across all embedding dimensions, we encourage the embeddings to align with the Fiedler vector.
  The Fiedler Vector Approximation Loss is defined as:
  $$\mathcal{L}_{\text{fiedler}} = \frac{1}{d}\sum_{i=1}^{d} R(\mathbf{x}_{\text{recon},i}, \mathbf{L}), \tag{4}$$
  where d is the embedding dimension, and $\mathbf{x}_{\text{recon},i}$ denotes the i-th column of the reconstructed embeddings.
- **Higher-Order Spectral Loss**: This component aims to capture eigenvectors associated with larger eigenvalues (higher-frequency components), enabling the embeddings to encode more complex structural features important for multi-way partitioning. We define a high-pass filtered version of the embeddings:
  $$\mathbf{h}_{\text{high}} = \mathbf{x}_{\text{recon}} - \alpha\mathbf{L}\mathbf{x}_{\text{recon}}, \tag{5}$$
  where α is a scaling factor that adjusts the influence of the Laplacian term, controlling the emphasis on higher-frequency components.
  The Higher-Order Spectral Loss is then defined as:
  $$\mathcal{L}_{\text{higher_order}} = \frac{1}{R(\mathbf{h}_{\text{high}}, \mathbf{L})}. \tag{6}$$
  By minimizing the inverse of the Rayleigh quotient of $\mathbf{h}_{\text{high}}$, we encourage these high-frequency components to align with higher-order eigenvectors of $\mathbf{L}$.

The combined Comprehensive Rayleigh Quotient Alignment Loss is expressed as:

$$\mathcal{L}_{\text{Rayleigh}} = \mathcal{L}_{\text{residual}} + \lambda_1\mathcal{L}_{\text{fiedler}} + \lambda_2\mathcal{L}_{\text{higher_order}}, \tag{7}$$

where λ_1 and λ_2 are weighting factors that balance the contributions of each component.

3) Total Loss Function: In summary, the total loss function used in GPart combines the Spectral Consistency Loss and the Comprehensive Rayleigh Quotient Alignment Loss:

$$\mathcal{L} = \mathcal{L}_{\text{Rayleigh}} + \mathcal{L}_{\text{consistency}}. \tag{8}$$

Since the Laplacian matrix $\mathbf{L}$ can be computed from the input graph, no labeled data is required, making this an unsupervised approach for graph partitioning tasks. By carefully designing these loss functions, we guide the embeddings to capture both global and local structural properties essential for effective graph partitioning.

C. Details of Inference Framework

The inference framework in GPart is designed to leverage the components trained during the learning phase, specifically reusing the embedding layers and coarsening module without requiring further training. The inference pipeline is composed of embedding layers, coarsening module, an initial partitioning step and a projection and refinement phase, with each stage meticulously structured to optimize partitioning quality and balance across partitions.

1) Initial Partitioning: The primary objective of the initial partitioning step is to establish a foundational partitioning scheme for the coarsened graph, which then serves as a basis for further refinement. Given the reduced size of the most coarsened graph, GPart utilizes the single-threaded METIS algorithm to perform this initial partitioning.

2) Projection and Refinement Module: In the projection and refinement phase, GPart reconstructs the input graph by progressively uncoarsening it from the coarsest level back to its original, finest granularity. At each uncoarsening level, GPart performs a refinement step to enhance partitioning quality by optimizing the cut size while ensuring balance across partitions.

Traditional refinement algorithms, such as the Fiduccia-Mattheyses (FM) algorithm [4], evaluate and potentially reassign every node based on computed gains, which is computationally intensive for large-scale graphs. GPart introduces an efficient refinement strategy that reduces computational overhead by leveraging the learned node embeddings and a centroid-based method.

Specifically, GPart calculates **weighted centroids** for each partition using the node embeddings generated by the coarsening module. These embeddings capture essential structural and spectral properties of the graph, positioning nodes in a high-dimensional space where proximity reflects structural similarity. This makes centroid calculations both meaningful and computationally feasible, especially on GPUs.

The refinement process focuses on nodes whose current partition assignments—determined by the solution from the previous coarser level—differ from the assignments suggested by their nearest centroids in the embedding space. Among these **candidate nodes**, GPart computes the **gain** for each node, representing the potential improvement in cut size if the node were moved to a different partition.

GPart prioritizes moving nodes with the highest gains. After each node movement, it recalculates the gains of affected neighboring nodes to reflect changes in the partitioning state. Throughout this process, GPart continuously tracks and records the partitioning configuration that achieves the best cumulative gain improvement. By the end of the refinement phase, even if subsequent moves do not yield better results, GPart selects the partitioning configuration with the highest recorded gain improvement among all moves performed.

To maintain computational efficiency, GPart restricts the refinement process to a maximum of **50** node moves per uncoarsening level, ensuring that only the most impactful node reassignments are performed while adhering to balance constraints.

This targeted gain-based refinement approach, combined with embedding-based centroid calculations, enhances both the computational efficiency and effectiveness of the partitioning process.

V. Experimental Validation

A. Experimental Setup

The experimental validation of GPart was conducted using a framework developed with Python, PyTorch, and the PyTorch Geometric library. The experiments were run on a machine equipped with an Intel Core i7-13700 CPU, 128GB of memory, and an Nvidia RTX 4090 GPU with 24GB of VRAM.

1) Dataset: For training, we generated a dataset of subgraphs derived from large graphs selected from the DIMACS Graph Partitioning Challenge [11][1]. From these base graphs, we randomly extracted 1600 subgraphs, each containing approximately 20,000 nodes.

2) Training Configuration: GPart was trained using the Adam optimizer for 1500 epochs. The initial learning rate was set to 0.00005 and was adjusted using a learning rate scheduler with a step size of 50 and a decay factor of 0.2.

3) Loss Function Parameters: The loss function in GPart incorporates weighted components to balance various optimization objectives. The weights for the loss function components were set to $\lambda_1 = 1$ and $\lambda_2 = 0.01$, ensuring that the $L_{fiedler}$ and L_{high_order} terms are appropriately balanced during training. Additionally, a scaling factor $\alpha = 1$ was used to regulate the high-frequency emphasis within the loss.

4) Experimental Design: The experimental design includes the following analyses:

1) **Cut Size and Runtime Comparison:** To assess partitioning quality and computational efficiency, we compare GPart with METIS [2] on the Titan23 circuit benchmark suite [13] and on graphs from the DIMACS Graph Partitioning Challenge [11]. Additionally, we include a comparison with G-kway [5] on the same DIMACS benchmarks used in its evaluation.
2) **Memory Usage Analysis:** We evaluate the memory efficiency of GPart on the Titan23 benchmarks by comparing its memory usage with that of GAP [1] and GenPart [9].

It is important to note that Titan23 is a circuit benchmark suite represented as hypergraphs. For compatibility with graph partitioning algorithms, we convert the Titan23 benchmarks into clique expansion graphs. In this conversion, hyperedges with degrees exceeding 1000 are removed to focus the analysis on more manageable hyperedge sizes. Table VI summarizes the statistics of the graphs converted from the Titan23 benchmarks.

B. Cut Size and Runtime Comparison

1) Results on Titan23 Benchmarks: In this section, we compare the performance of GPart with METIS in terms of cut size and runtime efficiency on the Titan23 circuit benchmark suite. The experiments were conducted with partition counts $k = 2$ and $k = 3$, and balance factors ε of 3% and 10%.

The results in Table I present the cut size comparison for METIS and GPart at $k = 2$. GPart consistently demonstrates significant improvements in cut size reduction across most benchmarks, with an average improvement of 34.13% at $\varepsilon = 3\%$ and 42.92% at $\varepsilon = 10\%$. These results indicate that GPart can achieve smaller cut sizes compared to METIS, particularly under more relaxed balance constraints (10% balance factor). This improvement is especially notable in benchmarks such as *cholesky_mc*, *directrf*, and *segmentation*, where GPart outperforms METIS by over 65%.

TABLE I: Cut size comparison for METIS and GPart on Titan23 benchmarks at $k = 2$ with balance factors ε of 3% and 10%.

	$k = 2,\ \varepsilon = 3\%$			$k = 2,\ \varepsilon = 10\%$		
Benchmark	*METIS*	*GPart*	**Improvement (%)**	*METIS*	*GPart*	**Improvement (%)**
bitcoin_miner	4,173	2,904	30.41%	4,169	2,628	36.96%
bitonic_mesh	142,095	150,766	-6.10%	142,427	140,738	1.19%
cholesky_bdti	37,439	19,066	49.07%	35,755	18,557	48.10%
cholesky_mc	14,882	4,435	70.20%	8,949	4,173	53.37%
dart	44,648	35,857	19.69%	47,938	36,453	23.96%
denoise	1,976	1,518	23.18%	1,785	1,151	35.52%
des90	142,863	77,042	46.07%	142,870	78,970	44.73%
directrf	30,901	8,042	73.97%	31,201	7,715	75.27%
gsm_switch	14,416	8,189	43.20%	20,062	6,370	68.25%
LU230	54,328	47,267	13.00%	54,047	42,828	20.76%
LU_Network	14,883	13,421	9.82%	12,495	7,911	36.69%
mes_noc	81,956	51,047	37.71%	81,966	36,143	55.90%
minres	10,915	7,951	27.16%	11,162	5,426	51.39%
neuron	6,349	2,926	53.91%	6,322	3,084	51.22%
openCV	28,222	20,190	28.46%	29,169	15,139	48.10%
segmentation	1,661	573	65.50%	7,799	501	93.58%
SLAM_spheric	5,291	5,164	2.40%	5,291	5,041	4.73%
sparcT1_core	19,039	15,509	18.54%	20,953	18,968	9.47%
sparcT2_core	27,217	26,976	0.89%	24,869	18,712	24.76%
stap_qrd	31,383	20,882	33.46%	26,679	17,831	33.16%
stereo_vision	1,212	288	76.24%	1,821	289	84.13%
Average Cut Size Improvement			**34.13%**			**42.92%**

Similarly, Table II provides the cut size comparison for $k = 3$. GPart again demonstrates substantial improvements in cut size, with an average improvement of 31.47% at $\varepsilon = 3\%$ and 36.38% at $\varepsilon = 10\%$. GPart consistently outperforms METIS, particularly in graphs such as *openCV* and *segmentation*, where improvements exceed 56%. However, there are a few cases, such as *bitonic_mesh* at $\varepsilon = 10\%$ and *SLAM_spheric* at $\varepsilon = 3\%$, where METIS slightly outperforms GPart. These exceptions highlight that while GPart generally excels in reducing cut size, specific instances exist where traditional algorithms like METIS remain competitive.

In addition to cut size, we evaluated the execution times of GPart and METIS on the Titan23 benchmarks. The results, shown in Table III, demonstrate that GPart achieves superior time efficiency compared to the Python-wrapped METIS implementation [10]. On average, the initial partitioning step in GPart accounts for approximately 43.16% of the total execution time, highlighting its significance in the overall process. This proportion suggests that while GPart manages the coarsening and refinement steps, the initial partitioning phase remains a substantial component of the total execution time.

TABLE II: Cut size comparison for METIS and GPart on Titan23 benchmarks at $k = 3$ with balance factors ε of 3% and 10%.

	$k = 3,\ \varepsilon = 3\%$			$k = 3,\ \varepsilon = 10\%$		
Benchmark	*METIS*	*GPart*	**Improvement (%)**	*METIS*	*GPart*	**Improvement (%)**
bitcoin_miner	9,650	8,227	14.75%	10,350	8,250	20.29%
bitonic_mesh	230,579	212,809	7.71%	194,302	200,764	-3.33%
cholesky_bdti	53,503	31,405	41.30%	53,959	26,452	50.98%
cholesky_mc	21,754	7,835	63.98%	20,617	7,351	64.34%
dart	66,685	61,514	7.75%	70,156	50,369	28.20%
denoise	12,742	4,359	65.79%	9,878	4,253	56.94%
des90	204,411	106,937	47.69%	201,682	101,921	49.46%
directrf	38,632	23,860	38.24%	39,245	20,173	48.60%
gsm_switch	26,532	11,829	55.42%	28,218	12,118	57.06%
LU230	102,335	88,965	13.06%	95,167	83,638	12.11%
LU_Network	31,830	26,736	16.00%	29,255	19,858	32.12%
mes_noc	103,146	84,880	17.71%	111,964	83,234	25.66%
minres	18,736	12,392	33.86%	17,931	11,426	36.28%
neuron	8,597	5,352	37.75%	9,331	4,887	47.63%
openCV	133,203	58,412	56.15%	96,606	50,337	47.89%
segmentation	8,031	1,880	76.59%	3,798	1,858	51.08%
SLAM_spheric	31,396	32,577	-3.76%	34,734	30,610	11.87%
sparcT1_core	37,576	35,529	5.45%	35,797	26,465	26.07%
sparcT2_core	46,023	39,660	13.83%	45,776	40,966	10.51%
stap_qrd	42,891	32,511	24.20%	41,638	30,808	26.01%
stereo_vision	2,140	1,554	27.38%	2,499	894	64.23%
Average Cut Size Improvement			**31.47%**			**36.38%**

[1]The graphs used include *G3_circuit*, *asia.osm*, *delaunay_n15*, *delaunay_n16*, *delaunay_n17*, *delaunay_n18*, *delaunay_19*, *delaunay_20*, *ldoor*, and *NLR*.

TABLE III: Comparison of execution times for METIS and GPart.

	Time (s)	
Benchmark	*METIS*	*GPart*
bitcoin_miner	7.27	5.72
bitonic_mesh	5.90	2.50
cholesky_bdti	3.47	2.45
cholesky_mc	1.41	1.04
dart	11.32	3.34
denoise	10.35	3.81
des90	2.95	1.35
directrf	12.81	5.91
gsm_switch	25.96	10.96
LU230	7.57	6.86
LU_Network	50.95	15.50
mes_noc	33.09	9.58
minres	3.64	1.83
neuron	1.10	1.25
openCV	10.83	6.22
segmentation	5.68	1.79
SLAM_spheric	4.02	1.22
sparcT1_core	7.55	2.44
sparcT2_core	27.30	8.77
stap_qrd	2.86	2.47
stereo_vision	2.04	1.12
Average Normalized Time	**1.00**	**0.53**

TABLE IV: Cut size comparison for METIS and GPart on graphs from DIMACS Graph Partitioning Challenge with varied partition counts k.

	Statistics		$k=2,\ \varepsilon=10\%$			$k=3,\ \varepsilon=10\%$		
Benchmark	$\|V\|$	$\|E\|$	*METIS*	*GPart*	**Improvement (%)**	*METIS*	*GPart*	**Improvement (%)**
delaunay_n19	524,288	1,572,823	1,492	1,361	8.78%	2,326	2,284	1.81%
delaunay_n20	1,048,576	3,145,686	2,035	1,849	9.14%	3,342	2,978	10.89%
delaunay_n21	2,097,152	6,291,408	2,807	2,593	7.62%	4,809	4,235	11.94%
delaunay_n22	4,194,304	12,582,869	4,038	3,694	8.52%	6,646	6,013	9.52%
delaunay_n23	8,388,608	25,165,784	5,888	5,234	11.11%	9,544	8,577	10.13%
delaunay_n24	16,777,216	50,331,601	8,568	7,453	13.01%	13,675	12,069	11.74%
ldoor	952,203	22,785,136	24,696	22,265	9.84%	37,583	33,732	10.25%
NLR	4,163,763	12,487,976	4,091	3,805	6.99%	7,072	6,377	9.83%
asia.osm	11,950,757	12,711,603	7	7	0.00%	19	14	26.32%
Average Cut Size Improvement			-	-	**8.33%**	-	-	**11.38%**

2) Results on DIMACS Benchmarks: We further assessed GPart's performance on graphs from the DIMACS Graph Partitioning Challenge, focusing on cut size improvements over METIS across various graph sizes and partition counts ($k = 2$ and $k = 3$) with a balance factor of $\varepsilon = 10\%$.

As shown in Table IV, GPart consistently achieves lower cut sizes than METIS across all tested graphs, with average improvements of 8.33% for $k = 2$ and 11.38% for $k = 3$.

While the industrial benchmark instances used in G-kway [5] are unavailable, we compared GPart's cut performance for $k = 2$ and $\varepsilon = 3\%$ with G-kway on four DIMACS graphs reported in G-kway's publication, as shown in Table V. The results for G-kway in the table are from [5]. On average, GPart achieved a 9.30% improvement in cut size over G-kway.

C. Memory Usage Analysis

In this section, we present a comparative analysis of memory usage among GAP [1], GenPart [9], and GPart within the context of graph partitioning. GAP and GenPart each employ an end-to-end GNN-based framework that requires both an embedding encoder and a partitioning decoder with large hidden layers to maintain high partitioning quality. This end-to-end design inherently demands greater memory during inference due to the increased number of parameters and activations in the larger hidden layers. In contrast, GPart utilizes a multilevel framework with smaller GNN layers, enabling a significant reduction in hidden layer size.

As stated in [9], memory constraints limit the applicability of GenPart to most Titan23 benchmarks. Due to frequent out-of-memory (OOM) issues during execution, GenPart could not be evaluated on large-scale designs and is therefore excluded from direct comparison. In contrast, GAP's source code and test cases are not publicly available, preventing us from reproducing its model performance. As a result, our comparison is limited to reported memory usage. Since directly measuring the memory consumption of both GAP and GenPart on large-scale benchmarks was impractical due to their high resource demands, we estimated their memory usage based on each model's architecture and parameters.[2] To validate our estimation method, we compared GPart's estimated memory usage against actual measurements on the Titan23 benchmarks, observing a deviation of less than 5%.

TABLE V: Cut size comparison between G-kway and GPart on graphs from the DIMACS Graph Partitioning Challenge, with a partition count of $k = 2$ and a balance factor of $\varepsilon = 3\%$.

	Statistics		$k=2,\ \varepsilon=3\%$		
Benchmark	$\|V\|$	$\|E\|$	*G-kway* [5]	*GPart*	**Improvement (%)**
delaunay_n24	16,777,216	50,331,601	8,463	7,465	11.79%
ldoor	952,203	22,785,136	25,578	23,932	6.44%
NLR	4,163,763	12,487,976	4,705	3,812	18.98%
asia.osm	11,950,757	12,711,603	7	7	0.00%
Average Cut Size Improvement			-	-	**9.30%**

Our analysis compares GPart's embedding and representation layers with the encoders used in GAP and GenPart. GPart's memory usage, however, was measured directly.

Table VI presents the memory usage comparison for GAP, GenPart, and GPart across benchmarks, with GAP and GenPart's values being estimated, while GPart's are measured directly. As shown, GPart achieves significant memory savings, requiring on average 24.6x less memory than GAP and 12.4x less than GenPart, highlighting its memory efficiency.

TABLE VI: Memory usage comparison for GAP, GenPart, and GPart.

	Statistics		Memory Usage (GB)		
Benchmark	$\|V\|$	$\|E\|$	*GAP*	*GenPart*	*GPart*
bitcoin_miner	1,089,284	3,662,788	20.61	10.51	0.81
bitonic_mesh	192,064	12,508,583	49.33	24.92	2.07
cholesky_bdti	266,422	4,660,080	19.54	9.89	0.79
cholesky_mc	113,250	2,592,938	10.67	5.39	0.44
dart	202,354	16,465,022	64.63	32.65	2.72
denoise	275,638	22,737,143	89.23	45.07	3.76
des90	111,221	6,752,436	26.68	13.48	1.12
directrf	931,275	19,432,048	80.41	40.68	3.26
gsm_switch	493,260	60,337,200	235.36	118.87	9.93
LU230	574,372	14,618,649	59.74	30.21	2.45
LU_Network	635,456	82,063,167	319.90	161.56	13.50
mes_noc	547,544	60,370,265	235.81	119.10	9.94
minres	261,359	6,495,763	26.58	13.44	1.09
neuron	92,290	2,770,749	11.23	5.68	0.47
openCV	217,453	28,852,387	112.44	56.78	4.75
segmentation	138,295	10,471,126	41.16	20.79	1.73
SLAM_spheric	113,115	5,636,907	22.39	11.31	0.94
sparcT1_core	91,976	16,697,490	64.87	32.76	2.75
sparcT2_core	300,109	46,519,712	180.99	91.40	7.65
stap_qrd	240,240	4,071,793	17.12	8.67	0.69
stereo_vision	94,050	3,785,776	15.15	7.65	0.64
Average Normalized Memory Usage			24.6x	12.4x	**1x**

VI. Conclusion

We introduced **GPart**, a scalable multilevel graph partitioning framework that integrates GNN-generated embeddings with efficient coarsening and refinement techniques. By capturing essential structural properties through unsupervised training, GPart achieves high-quality partitioning results. Additionally, its multilevel approach significantly reduces memory usage compared to existing GNN-based partitioners, making it suitable for large-scale graphs.

Experimental results on the Titan23 and DIMACS benchmarks demonstrate that GPart achieves substantial improvements in cut size over leading methods like METIS and G-kway. These advancements highlight GPart's effectiveness and scalability in practical graph partitioning applications.

[2]GAP uses five GraphSAGE layers with a 512-dimensional hidden size and a 64-dimensional output, while GenPart employs a five-layer GraphConv encoder [12] with a 256-dimensional hidden size and a variational layer as the final layer, also producing a 64-dimensional output.

REFERENCES

[1] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini, "A Deep Learning Framework for Graph Partitioning," in *Proceedings of International Conference on Learning Representations*, 2019.

[2] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[3] B. Fagginger Auer and R. Bisseling, "A GPU Algorithm for Greedy Graph Matching," *Facing the Multicore-Challenge*, vol. 2, pp. 108-119, 2012.

[4] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proceedings of Design Automation Conference*, pp. 175-181, 1982.

[5] W. L. Lee, D. L. Lin, T. W. Huang, S. Jiang, T. Y. Ho, Y. Lin, and B. Yu, "G-kway: Multilevel GPU-Accelerated k-way Graph Partitioner," in *Proceedings of Design Automation Conference*, 2024.

[6] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini, "GAP: Generalizable Approximate Graph Partitioning Framework," 2019, arXiv:1903.00614.

[7] B. Goodarzi, F. Khorasani, V. Sarkar, and D. Goswami, "High Performance Multilevel Graph Partitioning on GPU," in *Proceedings of International Conference on High Performance Computing & Simulation*, pp. 769–778, 2019.

[8] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," in *Proceedings of Neural Information Processing Systems*, 2017.

[9] M. Chen and T. C. Wang, "A Hypergraph Partitioner Utilizing a Novel Graph Generative Model," in *Proceedings of International Conference on Computer-Aided Design*, 2024.

[10] A. L. Bruggeman and I. van Gent, "metis-python," [Online]. Available: https://pypi.org/project/metis/.

[11] D. A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, "Graph Partitioning and Graph Clustering," in *Proceedings of DIMACS Implementation Challenge*, 2012.

[12] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and Leman Go Neural: Higher-order Graph Neural Networks," in *Proceedings of AAAI Conference on Artificial Intelligence*, 2018.

[13] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling Large and Complex Benchmarks in Academic CAD," in *Proceedings of International Conference on Field Programmable Logic and Applications*, 2013, pp. 1–8.

[14] L. W. Hagen and A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, 1992, pp. 1074–1085.

[15] C. J. Alpert and S.-Z. Yao, "Spectral Partitioning: The More Eigenvectors, The Better," in *Proceedings of Design Automation Conference*, 1995, pp. 195–200.

[16] M. Fiedler, "A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Application to Graph Theory," *Czechoslovak Mathematical Journal*, vol. 25, no. 4, pp. 619–633, 1975.

[17] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "SpecPart: A Supervised Spectral Framework for Hypergraph Partitioning Solution Improvement," in *Proceedings of International Conference on Computer-Aided Design*, 2022.

[18] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "K-SpecPart: Supervised Embedding Algorithms and Cut Overlay for Improved Hypergraph Partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 4, pp. 1232–1245, 2024.

[19] C. Deng, Z. Zhao, Y. Wang, Z. Zhang, and Z. Feng, "GraphZoom: A Multi-level Spectral Approach for Accurate and Scalable Graph Embedding," in *Proceedings of International Conference on Learning Representations*, 2020.

[20] R. Liang, A. Agnesina, and H. Ren, "MedPart: A Multi-Level Evolutionary Differentiable Hypergraph Partitioner," in *Proceedings of International Symposium on Physical Design*, 2024, pp. 3–11.

[21] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application In VLSI Domain," in *Proceedings of Design Automation Conference*, 1997, pp. 526–529.

PyraNet: A Multi-Layered Hierarchical Dataset for Verilog

1st Bardia Nadimi
Dept. of Computer Science & Eng.
University of South Florida
Tampa, Florida, United States
bnadimi@usf.edu

2nd Ghali Omar Boutaib
Dept. of Computer Science & Eng.
University of South Florida
Tampa, Florida, United States
ghaliomar@usf.edu

3rd Hao Zheng
Dept. of Computer Science & Eng.
University of South Florida
Tampa, Florida, United States
haozheng@usf.edu

***Abstract*—Recently, there has been a growing interest in leveraging Large Language Models for Verilog code generation. However, the current quality of the generated Verilog code remains suboptimal. This is largely due to the absence of well-defined, well-organized datasets with high-quality samples, as well as a lack of innovative fine-tuning methods and models specifically trained on Verilog. In this paper, we introduce a novel open-source dataset and a corresponding fine-tuning technique, which utilizes a multi-layered structure that we refer to as PyraNet. Our experiments demonstrate that employing the proposed dataset and fine-tuning approach leads to a more accurate fine-tuned model, producing syntactically and functionally correct Verilog code. The evaluation results show improvements by up-to 32.6% in comparison to the CodeLlama-7B baseline model and up-to 16.7% in comparison to the state-of-the-art models using VerilogEval evaluation platform.**

***Index Terms*—Large Language Models, fine-tuning, Verilog, dataset, transformers**

I. Introduction and Motivation

The introduction of attention-based models [1] represents a landmark development in the field of language processing. This breakthrough spurred the widespread adoption of transformer architectures, driving significant advancements in the area. Many models, such as Generative Pre-trained Transformers (GPT) [2], Bidirectional Encoder Representations from Transformers (BERT) [3], and Language Model for Dialogue Applications (LaMDA) [4], have leveraged transformer technology to achieve notable success.

The motivation behind utilizing large language models (LLMs) for hardware code generation is centered on developing a tool that streamlines the hardware modeling process for designers [5], [6]. It also enhances the security aspects of hardware design [7]. Integrating LLMs into this field aims to reduce the complexities traditionally involved in hardware design. By harnessing the advanced features of these models, hardware model development is expected to become more intuitive and efficient.

Additionally, the automation of hardware code generation through LLMs is vital for reducing human error. Given the intricate and technical nature of hardware design, manual coding is often prone to mistakes [8]. By automating the process, LLMs not only speed up development but also significantly lower the chances of introducing errors that could arise from human oversight. This results in more reliable and robust hardware designs, as the automated system ensures consistent, high-quality code generation with reduced risk of faults. Ultimately, employing LLMs in this domain marks a move toward more efficient, error-resilient, and user-friendly approaches to hardware design.

While there has been extensive research focused on software program synthesis, hardware code synthesis remains relatively underexplored. Recently, researchers have begun to investigate the use of LLM architectures for generating Hardware Description Language (HDL) code [9]–[19].

Despite early successes in using LLMs for HDL code generation in hardware design, several challenges remain that must be addressed to push the field forward. One key issue is the limited availability of labeled data necessary for effective fine-tuning of LLMs, as seen in the more mature field of software code generation. Existing approaches to Verilog code generation often suffer from syntax and functionality errors [9], [10], highlighting the need for further improvements.

In this paper, we introduce PyraNet, a multi-layered open-source dataset structure [1] that utilizes a wide range of open-source Verilog samples, irrespective of their code quality. By organizing the dataset into different quality tiers, we are able to maximize the value from both the high-quality samples and those of lower quality. The overall structure of the dataset is depicted in Fig.1-a.

The proposed fine-tuning technique is designed to fully leverage all data entries across the different tiers of the dataset. This is done by assigning higher loss weights to the high-quality data, and progressively reducing these weights for lower-quality data as we move down the pyramid [20]. Additionally, we implement a curriculum learning approach [21], starting the fine-tuning process with simpler data and gradually introducing more complex data as training progresses. By combining both the loss weighting and curriculum learning strategies, we can achieve a more effective fine-tuned model. The overall proposed fine-tuning architecture is depicted in Fig. 1. The **key contributions** of this paper is summerized as follows:

- The introduction of the PyraNet open-source dataset that includes various quality tiers, categorized based on the quality of the Verilog code, as well as the available labels for each entry. The labels include information such as the complexity level of the code, code rankings, design descriptions, and compile details.

[1] https://huggingface.co/datasets/bnadimi/PyraNet-Verilog

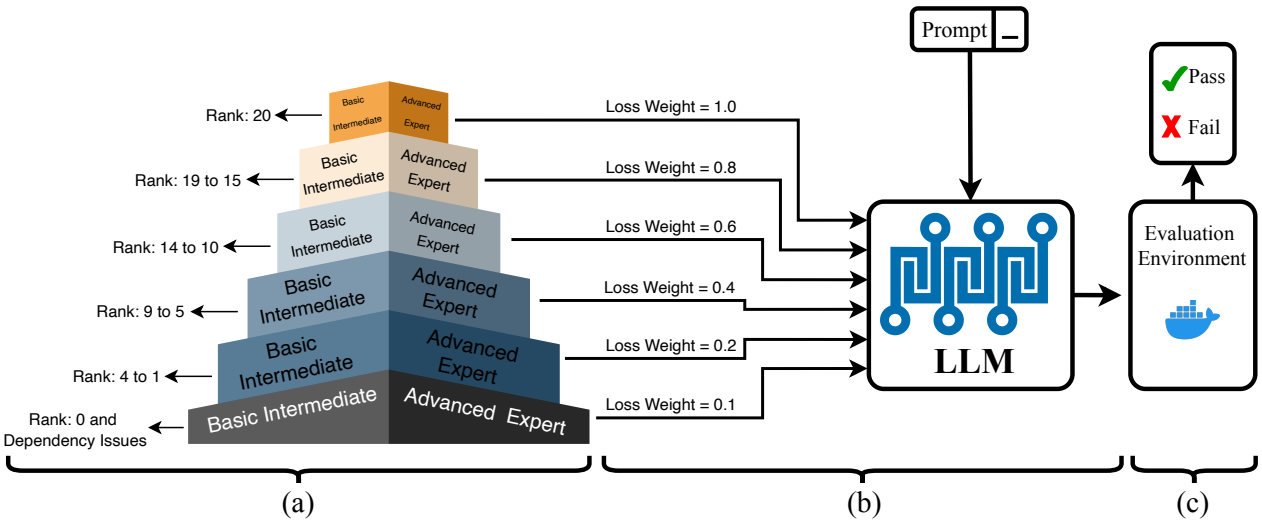


Fig. 1. Proposed Overall Architecture. (a) PyraNet dataset architecture. (b) PyraNet fine-tuning architecture. (b) Evaluation process.

- The introduction of the corresponding fine-tuning approach which is designed to maximize the benefits of the proposed PyraNet dataset by incorporating both the loss weighting technique and the curriculum learning strategy.
- Experiments show that the new PyraNet approach improves the Verilog code generation by up-to 32.6% in comparison to the CodeLlama-7B baseline model and up-to 16.7% in comparison to the state-of-the-art approaches including RTLCoder [18] and OriGen [22].

The remainder of this paper is organized as follows: Section II reviews related research in the field. Section III introduces the methodology, along with the proposed dataset and fine-tuning approach. Section IV focuses on comparisons and evaluations. Lastly, conclusions and future directions are discussed in Section V.

II. Related Works

The study in [9] explores the capability of LLMs to generate Verilog code for hardware design, demonstrating significant success in achieving syntactically accurate outputs through fine-tuning on Verilog-specific datasets. By emphasizing the role of LLMs in reducing human errors and enhancing automation in hardware design, the research underscores the need for further advancements in functional correctness and sets the stage for broader AI integration in hardware development.

The study in [10] presents VerilogEval, a benchmarking framework for evaluating the performance of LLMs in generating Verilog code for hardware design and verification. Utilizing a dataset of 156 problems from HDLBits, it investigates the automation of Verilog code generation across varying complexities, from simple circuits to advanced finite-state machines. The findings demonstrate that fine-tuning LLMs improves the quality of generated code, highlighting the transformative role of AI in optimizing hardware design workflows. Additionally, the research emphasizes the potential of supervised fine-tuning to enhance LLM capabilities, representing a significant step forward for both academic research and practical applications in the domain.

The study in [12] addresses the absence of standardized benchmarks for evaluating LLMs, such as ChatGPT, in hardware design, with a focus on Register Transfer Level (RTL) generation from natural language. It introduces RTLLM, an open-source benchmark designed to evaluate the effectiveness of LLMs in RTL design, establishing new standards for syntax, functionality, and design quality. The study also proposes a novel prompt engineering technique, "self-planning," which enhances GPT-3.5's performance, representing a major advancement in leveraging LLMs for complex and scalable hardware design tasks.

MG-Verilog [23] introduces a multi-grained dataset tailored for LLM-assisted Verilog code generation, addressing limitations in size, complexity, and description granularity found in existing hardware datasets. MG-Verilog includes over 11,000 Verilog code samples with corresponding descriptions at varying levels of detail, such as high-level summaries, block summaries, and line-by-line comments. The dataset facilitates balanced fine-tuning, enabling LLMs to improve both code implementation accuracy and generalization across diverse hardware design tasks. Extensive experiments demonstrate the dataset's effectiveness, showcasing superior performance compared to models trained on other datasets, with enhanced metrics in code generation accuracy and functionality.

RTLCoder [18] introduces a fully open-source framework for RTL code generation, addressing the challenges of privacy concerns and the lack of high-performance open-source LLMs for Verilog generation. The framework includes a dataset of over 27,000 instruction-code pairs, generated through an automated pipeline that ensures diversity and complexity across a wide range of hardware design tasks. RTLCoder employs a lightweight LLM architecture with only 7 billion parameters, fine-tuned using a novel training scheme that incorporates code quality feedback to enhance generation accuracy.

MEV-LLM [24] presents a multi-expert large language model architecture tailored for Verilog code generation, addressing challenges of syntax and functionality errors in exist-

ing approaches. MEV-LLM employs multiple expert models, each fine-tuned to handle specific design complexity levels, ranging from basic to expert-level Verilog designs. The study develops a categorized dataset with fine-grained and coarse-grained labels to enhance the fine-tuning process, enabling better alignment between input descriptions and generated code. Evaluation against state-of-the-art models demonstrates significant improvements, with MEV-LLM achieving up to a 23.9% increase in the pass@k metric.

The paper [22] introduces OriGen, an open-source framework for Register Transfer Level (RTL) code generation, designed to address the limitations of existing approaches in data quality and self-reflection. OriGen employs a novel code-to-code augmentation methodology to generate high-quality RTL datasets and incorporates a self-reflection mechanism for error correction using compiler feedback. The framework trains two specialized LoRA (Low-Rank Adaptation) models, one for generating initial RTL code and another for rectifying syntactic errors, enhancing its ability to handle complex Verilog tasks. Experimental results show that OriGen outperforms all existing open-source LLMs, achieving a 12.8% improvement over the best-performing model on the VerilogEval-Human benchmark and demonstrating superior self-reflection capabilities, surpassing GPT-4 Turbo by 19.9% in syntactic correctness.

CodeV [25] presents a robust framework for Verilog code generation by leveraging multi-level summarization and fine-tuning large language models (LLMs). The approach constructs a high-quality dataset of 165,000 Verilog modules collected from real-world sources, enriched by GPT-3.5-generated functional and high-level summaries. The experimental results demonstrate state-of-the-art performance on benchmarks like VerilogEval and RTLLM, with CodeV surpassing GPT-4 by 22.1% on VerilogEval and achieving notable gains in syntax and functional correctness. This work highlights the potential of combining domain-specific datasets and advanced fine-tuning techniques to push the boundaries of hardware description language generation.

BetterV [26] introduces a framework for controlled Verilog generation that enhances Verilog implementation through a novel combination of instruct-tuning and generative discriminators. BetterV utilizes a domain-specific dataset, created through rigorous filtering and augmentation techniques, and aligns Verilog with C programs to enhance large language models' (LLMs) understanding of hardware description languages. By employing task-specific generative discriminators, BetterV optimizes downstream Electronic Design Automation (EDA) tasks such as synthesis node reduction and verification runtime. The framework achieves state-of-the-art performance on the VerilogEval benchmark, outperforming GPT-4 in functional correctness.

AutoVCoder [27] addresses challenges in register-transfer level (RTL) code generation through three key innovations: a high-quality hardware dataset generation strategy, a two-round fine-tuning process, and a domain-specific retrieval-augmented generation (RAG) mechanism. Experimental evaluations demonstrate AutoVCoder's superiority over existing state-of-the-art models, achieving notable improvements across multiple benchmarks, including up to 3.4% gains in functional correctness on the RTLLM benchmark. By integrating domain-specific retrieval and fine-tuning techniques, AutoVCoder bridges the gap between general-purpose LLM capabilities and the specialized requirements of RTL code generation, paving the way for efficient and high-quality hardware design automation.

Each of the previously discussed studies faces challenges in producing Verilog codes that are both syntactically and functionally accurate.

III. Methodology and Dataset

This section starts by explaining the structure and organization of the PyraNet dataset, followed by a detailed description of the proposed fine-tuning process.

A. PyraNet Dataset

1) Dataset gathering: The majority of our dataset was gathered from publicly accessible GitHub repositories. Additionally, some Verilog code examples were generated using commercial large language models like GPT-4o-mini.

2) Post-download dataset filters: Following the collection of all Verilog code samples, we implemented multiple filtering procedures to guarantee the dataset's quality and relevance. The following are the filtration steps:

- `Empty/Broken files`: The initial step involved the removal of empty and corrupted or broken files. A Python script was employed to detect files with encoding issues by attempting to read each file. Any files that could not be processed due to encoding errors were discarded. Likewise, files that were successfully read but were empty were also excluded from the dataset.
- `Module declaration`: Files without a valid module declaration were similarly filtered out. Using a Python script, we identified and excluded any files that lacked module declarations, following the same procedure as the check for empty or corrupted files.
- `Code Deduplication`: We employed the Jaccard similarity algorithm to perform deduplication. This method computes the similarity between sets of tokens derived from the code samples by measuring the intersection over the union of the sets. Code pairs with a Jaccard similarity score above a predefined threshold were identified as duplicates and subsequently removed, ensuring the dataset's integrity and reducing redundancy.
- `Syntax Check`: The most crucial filtering step involved checking for syntax errors, as a dataset containing such errors would cause significant problems when training a model. To mitigate this, each file was processed using Icarus Verilog [28], and files that failed due to syntax errors were discarded. Files containing other types of errors, such as missing imports or undefined references, were also labeled under "dependency issues" since these do not compromise the syntactic correctness of the code. Given that running Icarus Verilog is more computationally demanding than the earlier filtering steps, this step

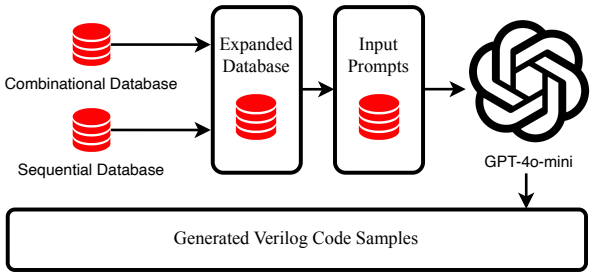


Fig. 2. Process for generating Verilog code samples using commercial LLMs.

was performed last, ensuring it was applied to a reduced subset of files.

3) Verilog code generation using commercial LLMs: In addition to the collected code samples, we also generated a set of code samples using commercial LLMs, such as GPT-4o-mini. To guide this process, we first compiled a database of keywords and categorized them into combinational and sequential circuits, focusing on general hardware and Verilog design terms such as adders, multipliers, counters, FSMs, etc. For each keyword, we expanded the database by detailing specific variations, for instance, ripple carry adders or carry-save adders—this step was referred to as "expanded-keywords." Finally, for each expanded keyword, we crafted input prompts with detailed design descriptions and queried GPT-4o-mini 10 times for each prompt using different temperature values. Fig. 2 depicts the overall process for the generating the Verilog code samples.

4) Ranking and Complexity labels: Each entry in the PyraNet dataset was evaluated and ranked using the GPT-4o-mini model. Rankings were assigned on a scale from 0 to 20, where 0 denotes the lowest score and 20 the highest. To facilitate the ranking process, we generated specific input prompts for each Verilog code sample in the dataset. The GPT-4o-mini model was instructed to score each code based on guidelines provided in a pre-prompt text, which detailed the scoring methodology. The scoring criteria focused on the overall Verilog coding style and the efficiency of the code. Fig.3 presents an example of a prompt along with its corresponding response.

To assign a complexity level to each data sample in the PyraNet dataset, we closely followed the methodology presented in the MEV-LLM work [24]. This approach allowed us to generate and categorize the dataset into four distinct tiers: `Basic`, `Intermediate`, `Advanced`, and `Expert`.

5) Dataset Organization: We organized the collected and generated Verilog code samples into six distinct layers:

`Layer 1` comprises a total of 235 data samples ranked 20 out of 20, representing the highest tier of Verilog codes. These samples are distinguished by their efficiency, coding style, and syntactic correctness.

`Layer 2` includes $150,279$ data samples with rankings ranging from 19 to 15.

`Layer 3` contains a total of $105,973$ data samples with rankings between 14 and 10.

Prompt:
Act as a teacher and rank the quality of this Verilog code in scale of 0 to 20, with 0 being syntactically incorrect and 20 being a good Verilog code in terms of efficiency and coding style:

```
module halfAdder(
    input A,
    input B,
    output Sum,
    output Cout
    );

    assign Sum = A ^ B;
    assign Cout = A & B;
endmodule
```

Just give me the score only.

Response:

```
Score: 20 out of 20.
```

Fig. 3. Example of a prompt and the response used for ranking.

`Layer 4` includes $5,015$ data samples with rankings ranging from 9 to 5.

`Layer 5` contains 275 data samples with rankings between 4 and 1.

It is important to note that for Layers 1 through 5, we ensured representation across all complexity levels. Additionally, all samples in these layers successfully compiled without errors.

`Layer 6` contains a total of $430,461$ data samples with dependency issues that could not be compiled without errors or ranking of 0.

Initially, we collected approximately 2.4 million code samples from open-source GitHub repositories and generated an additional 150,000 samples using GPT-4o-mini. Following the pre-processing phase, we obtained 692,238 code samples that were deduplicated, ranked, described, and compiled without syntax errors or having only dependency issues.

B. PyraNet fine-tuning

To effectively leverage the proposed PyraNet dataset, we have introduced a dedicated fine-tuning architecture. This architecture incorporates two key techniques: loss weighting and curriculum learning. In this section, we will first elaborate on the loss weighting approach, followed by a detailed explanation of the curriculum learning strategy.

1) Loss Weighting: As previously described, the PyraNet dataset is structured into six layers of data samples, each corresponding to different tiers. In our loss weighting technique, we fine-tune the model by assigning distinct loss weights to each tier within the dataset. Starting with the top layer—which contains highly ranked and high quality code samples—we set the loss weight to 1.0. This maximal weight ensures

that entries in this layer have the most significant impact on the fine-tuning process. As we descend through the layers of the PyraNet dataset, the loss weights are progressively decreased. This reduction lessens the influence of data entries from lower tiers on the final fine-tuned model. Consequently, the model prioritizes learning from high-quality data samples while minimizing the effects of those with lower quality. For the layer of 2 to 6 the loss weights are 0.8, 0.6, 0.4, 0.2, and 0.1 respectively, as depicted in Fig.1-b.

2) Curriculum Learning: As demonstrated in [21], employing a meaningful training sequence—beginning with simpler samples and progressing to more complex ones—can enhance model performance compared to standard training methods that use random sampling. In our proposed PyraNet dataset, we have assigned complexity levels to all code samples within each tier. Considering the benefits of curriculum learning, we propose initiating the fine-tuning process with the `basic` complexity level for each tier, followed by `intermediate`, `advanced`, and finally `expert` levels.

The fine-tuning process generally commences with the highest tier of the PyraNet dataset. Each tier within the PyraNet dataset is further categorized according to code complexity. Specifically, the process begins with data entries of `basic` complexity within the top tier, followed by `intermediate`, `advanced`, and `expert` complexity levels in sequence. This hierarchical structure is maintained across all tiers as the fine-tuning progresses downward through the dataset. By progressing through complexity levels in each tier, the model effectively leverages structured data to enhance its performance.

IV. Evaluation and Discussion

A. Baseline LLMs

In this study, we aim to demonstrate the effectiveness of our proposed PyraNet dataset and the accompanying fine-tuning technique by conducting three distinct experiments and comparing the results with state-of-the-art models. We selected the CodeLlama-7b, CodeLlama-13b, and DeepSeek-Coder-7B models as our base models and fine-tuned them using the PyraNet dataset alongside our proposed fine-tuning methodology. For evaluation, we employed the VerilogEval platform to assess the performance of the models across all experiments.

B. Experiments

To thoroughly evaluate the effectiveness of our proposed PyraNet dataset and the accompanying fine-tuning technique, we designed a series of three experiments. These experiments aim to systematically assess the individual and combined impacts of the dataset and the fine-tuning approach on model performance. The experimental setup is as follows:

- Baseline Experiment: In the first experiment, we evaluated the CodeLlama and DeepSeek-Coder models by providing prompts without any fine-tuning. This established a baseline performance metric for comparison with subsequent experiments.

TABLE I
PyraNet vs SOTA models on VerilogEval [10]

Model	Verilog-Machine			Verilog-Human		
	$pass@1$	$pass@5$	$pass@10$	$pass@1$	$pass@5$	$pass@10$
MG-Verilog-CodeLlama-7B [23]	54.5	60	63	-	-	-
RTLCoder-DeepSeek [18]	61.2	76.5	81.8	41.6	50.1	53.4
OriGen-DeepSeek [22]	74.1	82.4	85.7	54.4	60.1	64.2
codeLlama-7B-Instruct	41.9	46.1	46.8	19.2	23.0	25.0
codeLlama-7B-Instruct PyraNet-Dataset	58.0	62.9	67.8	44.2	50.0	55.7
codeLlama-7B-Instruct PyraNet-Achitecture	62.9	69.2	74.1	48.0	53.2	57.6
codeLlama-13B-Instruct	48.6	54.5	57.3	32.6	35.8	39.1
codeLlama-13B-Instruct PyraNet-Dataset	58.7	63.6	67.8	45.5	51.2	57.0
codeLlama-13B-Instruct PyraNet-Achitecture	69.2	73.4	78.3	50.6	55.1	58.3
DeepSeek-Coder-7B-Instruct-v1.5	55.2	72.7	76.9	32.6	43.5	49.3
DeepSeek-Coder-7B-Instruct-v1.5 PyraNet-Dataset	73.4	81.8	86.7	53.8	60.8	63.4
DeepSeek-Coder-7B-Instruct-v1.5 PyraNet-Achitecture	**77.6**	**84.6**	**89.5**	**58.3**	**62.8**	**67.9**

TABLE II
Pre-trained LLM architectures and fine-tuning information

Model	Layers	# of Heads	Head Size	Context Size	learning rate	# of epochs
CodeLlama-7b-Instruct	32	32	128	100,000	$2e^{-4}$	$1, 2, 3$
CodeLlama-13b-Instruct	40	40				
DeepSeek-Coder-7B-Instruct-v1.5	30	30		4000		

- PyraNet-Only Fine-Tuning (PyraNet-Dataset): In the second experiment, the CodeLlama and DeepSeek-Coder models were fine-tuned using the PyraNet dataset without applying the specialized fine-tuning approach. This allowed us to isolate and assess the improvements attributable solely to the PyraNet dataset.
- Combined PyraNet dataset and PtraNet Fine-Tuning Approach (PyraNet-Architecture): In the final experiment, we fine-tuned the CodeLlama and DeepSeek-Coder models using both the PyraNet dataset and our proposed fine-tuning technique. This combined approach was designed to evaluate the synergistic effects of the dataset and the fine-tuning methodology on model performance.

Through these experiments, we systematically evaluate the contributions of the PyraNet dataset and the fine-tuning technique, providing a comprehensive analysis of their impact compared to existing state-of-the-art models. The results are organized in Table I.

C. Fine-tuning Explained

As outlined in the preceding sections, we fine-tuned two models: one using the PyraNet dataset without applying the proposed fine-tuning technique, and the other utilizing both the PyraNet dataset and our proposed fine-tuning method.

In the first experiment, we fine-tuned the CodeLlama-7B, CodeLlama-13B, and DeepSeek-Coder-7B models using each available (data, description) pair from the dataset. Specifically, the descriptions were used as inputs to the models, and the corresponding Verilog code served as the output. Throughout this experiment, the learning rate was maintained at $2e^{-4}$, and the loss weights were set to 1.0. The fine-tuning method utilizes the LoRa technique [29], adhering to its standard training configurations. The baseline LLM architectures chosen for this study, along with the fine-tuning parameters utilized, are succinctly summarized in Table II.

For the second experiment, we employed a hierarchical fine-tuning approach. We began by fine-tuning the aforementioned models with data entries from the top layer of the PyraNet dataset. This phase consisted of four sequential fine-tuning steps, starting with data entries labeled `Basic`, followed by those labeled `Intermediate`, `Advanced`, and `Expert`.

In each step, descriptions were used as inputs and the corresponding Verilog codes as outputs, effectively implementing a curriculum learning strategy. During this initial fine-tuning phase, the loss weight was set to 1.0, reflecting the use of the highest quality data available.

In subsequent rounds of fine-tuning, we progressed to lower layers of the PyraNet dataset, continuing the curriculum learning approach for each layer. The primary modification in these rounds was the adjustment of the loss weights, as illustrated in Fig.1-b. By conducting multiple fine-tuning iterations with varying loss weights, we implemented the loss weighting approach. The fine-tuning process concluded upon completing the lowest layer of the PyraNet dataset. It is important to note that the learning rate was maintained at a constant value of $2e^{-4}$ for each round of fine-tuning throughout this experiment as well. This structured methodology allowed us to systematically assess the impact of both the hierarchical fine-tuning and loss weighting techniques on model performance, ensuring that the models effectively leveraged high-quality data while appropriately incorporating varying levels of complexity from the dataset.

D. Comparison and Explanation of the Results

To ensure a fair evaluation, the proposed PyraNet fine-tuning approach was applied to both the CodeLlama model, for comparison with MG-Verilog [23], and the DeepSeek-Coder model, for comparisons with RTL-Coder [18] and OriGen [22]. The results in Table I indicate that the proposed model achieved up to a 32.6% improvement in the $pass@k$ metric over the CodeLlama baseline models and up to 25.7% over the DeepSeek-Coder baseline model. When compared to state-of-the-art models, the proposed approach demonstrated an 11.1% improvement in the $pass@k$ metric on the Verilog-Machine dataset compared to MG-Verilog. For the Verilog-Human, the proposed model achieved improvements of up to 16.7% against RTL-Coder and 3.9% against OriGen, both of which are based on fine-tuning with DeepSeek-Coder.

It is worth noting that OriGen incorporates a Self-Reflection loop, which includes an additional round of error correction—a feature not considered in this work due to time constraints. This accounts for the relatively smaller improvements over OriGen. Nevertheless, integrating OriGen's Self-Reflection loop with inference using the models fine-tuned with the proposed PyraNet architecture is expected to yield even greater performance gains, which will be performed in the future. All the gains achieved by the proposed PyraNet dataset and architecture are illustrated in Table III.

E. Dataset Quality Verification

As emphasized in [10], the integrity of the compiled dataset is crucial for the effectiveness of the fine-tuning process. Due to challenges in verifying each description and additional information incorporated into the PyraNet dataset via the GPT-4o-mini model, we adopted the verification methodology employed by the researchers in [10] to validate the labels generated by the GPT-4o-mini model. To assess the quality of the dataset, we randomly shuffled the codes, descriptions, and ranking information among the data entries, thereby creating mismatched sets of codes, descriptions, and rankings within each row of the PyraNet dataset. We then proceeded to fine-tune the model using this intentionally distorted dataset. Table IV presents the results of fine-tuning the CodeLlama-7B model with both the flawed and accurate datasets. The findings clearly demonstrate that the model's code generation capabilities are significantly compromised when trained on the erroneous dataset. This outcome verifies the accuracy of the labels and underscores the critical importance of dataset quality in the fine-tuning process. Furthermore, the results demonstrate that the accuracy of the fine-tuned model decreases when trained on the erroneous dataset compared to training solely on the PyraNet dataset without the proposed fine-tuning technique. Consequently, we decided to forgo the fine-tuning process using the proposed method, as it is evident that an erroneous dataset would similarly impair the effectiveness of the fine-tuning technique and result in reduced model performance.

TABLE III
PYRANET GAINS VS BASELINE MODEL AND SOTA

Model	VS	Verilog-Machine			Verilog-Human		
		$pass@1$	$pass@5$	$pass@10$	$pass@1$	$pass@5$	$pass@10$
codeLlama-7B-Instruct	vs Baseline	16.1	16.8	21	25	27	30.7
PyraNet-Dataset	vs MG-Verilog	3.5	2.9	4.8	-	-	-
codeLlama-7B-Instruct	vs Baseline	21	23.1	27.3	28.8	30.2	32.6
PyraNet-Achitecture	vs MG-Verilog	8.4	9.2	11.1	-	-	-
codeLlama-13B-Instruct	vs Baseline	10.1	9.1	10.5	12.9	15.4	17.9
PyraNet-Dataset	vs MG-Verilog	4.2	3.6	4.8	-	-	-
codeLlama-13B-Instruct	vs Baseline	20.6	18.9	21	18	19.3	19.2
PyraNet-Achitecture	vs MG-Verilog	14.7	13.4	15.3	-	-	-
DeepSeek-Coder-7B-Instruct-v1.5	vs Baseline	18.2	9.1	9.8	21.2	17.3	14.1
PyraNet-Dataset	vs RTL-Coder	12.2	5.3	4.9	12.2	10.7	10
	vs OriGen	-0.7	-0.6	1	-0.6	0.7	-0.8
DeepSeek-Coder-7B-Instruct-v1.5	vs Baseline	22.4	11.9	12.6	25.7	19.3	18.6
PyraNet-Achitecture	vs RTL-Coder	16.4	8.1	7.7	16.7	12.7	14.5
	vs OriGen	3.5	2.2	3.8	3.9	2.7	3.7

TABLE IV
RESULTS FOR ERRONEOUS DATASET

Model	Verilog-Machine			Verilog-Human		
	$pass@1$	$pass@5$	$pass@10$	$pass@1$	$pass@5$	$pass@10$
CodeLlama-7B with erroneous dataset	43.3	48.2	50.3	21.1	25.6	28.8
CodeLlama-7B with correct dataset	58.0	62.9	67.8	44.2	50.0	55.7

V. CONCLUSION AND FUTURE WORKS

In this study, we present a novel structured dataset, PyraNet, alongside its associated fine-tuning architecture. This architecture addresses the limitations of previous studies by achieving improved performance across diverse Verilog code types. The PyraNet fine-tuning framework employs advanced techniques such as loss weighting and curriculum learning to mitigate the challenges posed by the available datasets. Evaluation results reveal that the PyraNet dataset and its proposed architecture achieve the anticipated benefits, demonstrating up to a 32.6% improvement in the $pass@k$ metric. We suggest that there is substantial scope for further advancements, both through the development of more comprehensive and detailed datasets and by exploring alternative fine-tuning architectures.

ACKNOWLEDGMENT

This work is supported by the National Science Foundation under Grant No. 2434247. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Vaswani et al. Attention is all you need. 2017.

[2] Radford et al. Improving language understanding by generative pre-training. 2018.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[4] Cohen et al. Lamda: Language models for dialog applications. In *arXiv*. 2022.

[5] Md Rubel Ahmed, Bardia Nadimi, and Hao Zheng. System-on-chip message flow mining with masked-language models. In *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 496–500, 2023.

[6] Deepak Vungarala, Sakila Alam, Arnob Ghosh, and Shaahin Angizi. Spicepilot: Navigating spice code generation and simulation with ai guidance, 2024.

[7] Seyedsina Nabavirazavi, Rahim Taheri, Mohammad Shojafar, and Sundararaja Sitharama Iyengar. Impact of aggregation function randomization against model poisoning in federated learning. TrustCom Proceedings Series, pages 165–172, United States, May 2024. Institute of Electrical and Electronics Engineers Inc.

[8] Dessouky et al. HardFails: Insights into Software-Exploitable hardware bugs. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 213–230, Santa Clara, CA, August 2019. USENIX Association.

[9] Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Benchmarking large language models for automated verilog rtl code generation. pages 1–6. EDAA.

[10] M. J. Liu, N. Pinckney, B. Khailany, and H. X. Ren. Invited paper: Verilogeval: Evaluating large language models for verilog code generation. *2023 Ieee/Acm International Conference on Computer Aided Design, Iccad*, 2023.

[11] Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li. Chipgpt: How far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019*, 2023.

[12] Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 722–727. IEEE.

[13] Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. Chip-chat: Challenges and opportunities in conversational hardware design. In Andrew B. Kahng and Siddharth Garg, editors, *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, [Piscataway, NJ], 2023. IEEE. Literaturangaben.

[14] Fu et al. Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models, 2023.

[15] Liu et al. Chipnemo: Domain-adapted llms for chip design, 2023.

[16] Yun-Da Tsai, Mingjie Liu, and Haoxing Ren. Rtlfixer: Automatically fixing rtl syntax errors with large language models, 2024.

[17] Thakur et al. Autochip: Automating hdl generation using llm feedback, 2023.

[18] Liu et al. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution, 2024.

[19] Chang et al. Improving large language model hardware generating quality through post-LLM search. In *Machine Learning for Systems 2023*, 2023.

[20] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

[21] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[22] Fan Cui et al. Origen:enhancing rtl code generation with code-to-code augmentation and self-reflection, 2024.

[23] Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan Celine Lin. Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation, 2024.

[24] Bardia Nadimi and Hao Zheng. A multi-expert large language model architecture for verilog code generation. pages 1–5, June 2024.

[25] Yang Zhao et al. Codev: Empowering llms for verilog generation through multi-level summarization. *ArXiv*, abs/2407.10424, 2024.

[26] Zehua Pei, Huiling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. BetterV: Controlled verilog generation with discriminative guidance. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 40145–40153. PMLR, 21–27 Jul 2024.

[27] Mingzhe Gao et al. Autovcoder: A systematic framework for automated verilog code generation using llms, 2024.

[28] Stephen Williams. Icarus verilog.

[29] Edward J. Hu et al. Lora: Low-rank adaptation of large language models, 2021.

DARIS: An Oversubscribed Spatio-Temporal Scheduler for Real-Time DNN Inference on GPUs

Amir Fakhim Babaei
Department of Electrical & Computer Engineering,
Virginia Tech
Arlington, Virginia, USA
babaei@vt.edu

Thidapat Chantem
Department of Electrical & Computer Engineering,
Virginia Tech
Arlington, Virginia, USA
tchantem@vt.edu

***Abstract*—The widespread use of Deep Neural Networks (DNNs) is limited by high computational demands, especially in constrained environments. GPUs, though effective accelerators, often face underutilization and rely on coarse-grained scheduling. This paper introduces DARIS, a priority-based real-time DNN scheduler for GPUs, utilizing NVIDIA's MPS and CUDA streaming for spatial sharing, and a synchronization-based staging method for temporal partitioning. In particular, DARIS improves GPU utilization and uniquely analyzes GPU concurrency by oversubscribing computing resources. It also supports zero-delay DNN migration between GPU partitions. Experiments show DARIS improves throughput by 15% and 11.5% over batching and state-of-the-art schedulers, respectively, even without batching. All high-priority tasks meet deadlines, with low-priority tasks having under 2% deadline miss rate. High-priority response times are 33% better than those of low-priority tasks.**

I. Introduction

Deep Neural Networks (DNNs) demand high computational power and challenge resource-constrained systems. GPUs improve DNN training and inference [1], but sequential processing leads to under-utilization, requiring multiple parallel (multi-tenant) DNN scheduling [2]. Inference tasks in constrained settings often need real-time performance [3], common in fields like autonomous driving [4], healthcare [5], [6], AI at the edge [7], and NLP [8], [9]. GPU inference servers batch inputs for utilization [10], [11], but real-time schedulers cannot typically use input batching, as they require immediate task handling [12].

NVIDIA provides various concurrency mechanisms for designers, but no general guideline exists on the efficient GPU configurations and resource partitioning strategies in the literature. Most works isolate resources, with some exploring oversubscription [13]. Also, most works focus on maximizing throughput rather than achieving predictable real-time performance. GPUs have a gray-box architecture that makes timing predictions difficult. Approaches that consider *Worst-Case Execution Time* (WCET) prediction often either sacrifice throughput [14] or result in overly pessimistic estimates [15].

Each GPU has $N_{SM,max}$ *Streaming Multiprocessors* (SMs), its smallest independent units. GPU processes, or *kernels*, may number in the hundreds per DNN. Kernels run sequentially in *CUDA Streams* (referred to as streams after this), and running multiple streams in parallel improves GPU utilization by reducing SMs' idle time. With Multi-Process Service (MPS), we can create multiple *CUDA Contexts* (referred to as contexts after this), each assigned a portion of SMs. When total SMs allocated to contexts surpass $N_{SM,max}$, it is termed *Oversubscription*.

In this paper, we propose *DARIS*, a Deadline-Aware Real-Time DNN Inference Scheduler to address efficient GPU concurrency configuration challenges. DARIS can provide better predictability for periodic soft real-time tasks with two priority levels. Our primary goals are to minimize the deadline miss rate and maximize overall throughput. While we cannot guarantee that all high-priority tasks will meet their deadlines, DARIS demonstrates significantly fewer deadline misses compared to state-of-the-art methods. We also introduce *staging* as a coarse-grained preemption mechanism to achieve better priority-based scheduling. Our main contributions are as follows:

- To the best of our knowledge, this is the first work to conduct an in-depth analysis of GPU concurrency mechanisms, focusing on resource oversubscription and its benefits on real platforms.
- We propose a deadline-aware real-time spatio-temporal GPU scheduler that surpasses the throughput of single-tenant batching, without relying on batched inputs.

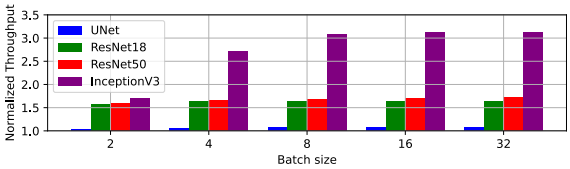


Fig. 1: Normalized throughput using batching

II. Design Choices

A. What Concurrency Mechanism to Use?

SGPRS [16] defines "knee point" as the maximum number of DNNs a scheduler can handle without missing deadlines. They found that without temporal partitioning, throughput drops significantly beyond the knee point due to interference. Higher oversubscription values generally improve throughput, though not consistently. These results suggest that a spatio-temporal scheduler is more effective than a purely spatial one, but further analysis is needed to comprehensively examine contexts, streams, and oversubscription levels.

B. Is Oversubscription Good?

Systematic studies on SM oversubscription are limited. NVIDIA suggests 200% SM oversubscription as optimal [17], though some research indicates it may cause interference [18]–[20], with limited experimental data. Laius [13] uses structured oversubscription, dedicating 100% of SMs to user-facing and another 100% to batched services. SGPRS [16] highlighted the benefits of oversubscription, with the remaining challenge of finding the best trade-off. In Section VI-E, we compare oversubscription levels, denoted $OS \geq 1$, to find this balance.

C. Is Batching Enough or Necessary?

Batching boosts throughput in GPUs but is often impractical for real-time inference with non-identical DNNs since waiting for jobs can cause missed deadlines. This study examines whether batching

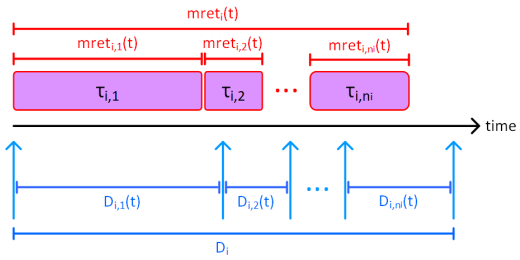


Fig. 2: Task staging and virtual deadline assignment

is sufficient and if its throughput can be matched without it. Experiments (Figure 1) show limited benefits for some DNNs (e.g., UNet) and significant gains for others (e.g., InceptionV3) with batching. DARIS achieves higher throughput without batching (Section VI).

D. How to Predict the Execution Time of DNNs?

Timing prediction is essential for real-time schedulers, but GPU parallelism makes execution times unpredictable. Clockwork [14] ensures predictable WCET by scheduling one DNN at a time, sacrificing throughput for predictability. Multi-tenant systems face resource contention, leading to variability and overly conservative WCET estimates. Wang *et al.* use history-based WCET from 10-minute simulations [15], also yielding conservative results. We propose a dynamic history-based prediction (Figure 9) for more optimistic estimates for soft real-time systems.

III. System Model

A. Task Model

We consider a set of N_c contexts, each with N_{SM} SMs. Periodic tasks correspond to individual DNNs, divided into sub-tasks or stages. We define a task set $TS = \{\tau_1, ..., \tau_{N_{ts}}\}$ with N_{ts} tasks, where each task τ_i has n_i sequential sub-tasks ($\tau_i = \{\tau_{i,1}, ..., \tau_{i,n_i}\}$). Tasks have two priority levels: high-priority (HP) and low-priority (LP), with counts N_h and N_l, respectively so that $N_{ts} = N_h + N_l$.

Each task is represented as $\tau_i(T_i, D_i, mret_i(t), p_i, ctx_i(t))$, where T_i is the period, D_i the relative deadline, $mret_i(t)$ the Maximum Recent Execution Time (MRET) at time t, p_i the priority level, and $1 \leq ctx_i(t) \leq N_c$ the current context. Deadlines are set equal to periods ($D_i = T_i$). MRET is used instead of WCET to avoid pessimistic estimates common with colocated DNNs (see Section III-B2). A stage is defined as $\tau_{i,j}(D_{i,j}(t), mret_{i,j}(t))$, where $D_{i,j}$ is the virtual deadline (determined using Equation (8)). Figure 2 shows task stages and their virtual relative deadlines based on MRET.

B. Definitions

1) Staging: To be able to enforce priorities more efficiently on a smaller scale, we introduce a synchronization-based, coarse-grained preemption mechanism called *staging*. Synchronization points can be placed after a few kernels [21], after each layer [22], after several layers [23], or even dynamically [24]. Excessive synchronization, however, can reduce GPU utilization, as deep learning frameworks release kernels asynchronously to optimize throughput. We segment DNNs into sequential stages/sub-tasks, allowing preemption only at these boundaries. Stages are selected based on the DNN's logical structure; for instance, ResNet [25] is divided into four stages.

2) Maximum Recent Execution Time (MRET): We propose a dynamic WCET estimation, *Maximum Recent Execution Time* (MRET), capturing the maximum execution time within a recent window, adapting to workload changes. MRET is computed per stage rather than for the entire task (Figure 2).

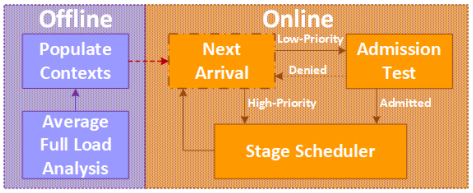


Fig. 3: Proposed Scheduler

$$mret_{i,j}(t) = \max_{p \in \{t-ws,...,t-1\}} et_{i,j}(p) \quad (1)$$

$$mret_i(t) = \sum_{1 \leq j \leq n_i} mret_{i,j}(t), \quad (2)$$

Here, ws is the window size; $mret_{i,j}$ and $et_{i,j}(t)$ are the MRET and actual execution time of $\tau_{i,j}$ at time t.

3) Utilization: In DARIS, we define the utilization of a task as:

$$u_i(t) = \frac{mret_i(t)}{T_i} \quad (3)$$

We use Equations (4) to (6) to measure the utilization of a context:

$$U_k^{h,t}(t) = \sum_{\substack{1 \leq i \leq N_h \\ ctx_i = k}} u_i^h(t) \quad (4)$$

$$U_k^{l,t}(t) = \sum_{\substack{1 \leq i \leq N_l \\ ctx_i = k}} u_i^l(t) \quad (5)$$

$$U_k^t(t) = U_k^{h,t}(t) + U_k^{l,t}(t) \quad (6)$$

where $U_k^t(t)$ is the total utilization of context k, and $U_k^{h,t}(t)$ and $U_k^{l,t}(t)$ are the total utilization of HP and LP tasks in context k, respectively. We will use Equation (6) for load balancing between contexts in the offline phase. For the admission test during the online phase, we will rely on the active utilization:

$$U_k^a(t) = U_k^{h,t}(t) + U_k^{l,a}(t) \quad (7)$$

The active utilization is defined as the sum of the total utilization of the HP tasks ($U_k^{h,t}(t)$) and the total utilization of *currently active* LP tasks ($U_k^{l,a}(t)$), i.e., LP tasks that have an active job released but not yet finished.

4) Virtual Deadline: The task deadlines are usually an application requirement set by the designer. We introduce the *virtual deadline*, an intermediary parameter assigned to each stage. The virtual deadline for $\tau_{i,j}$ is calculated as:

$$D_{i,j}(t) = \frac{mret_{i,j}(t)}{mret_i(t)} D_i \quad (8)$$

Figure 2 illustrates the relationship between MRET and virtual deadlines, with longer stages receiving a larger share of the task's total relative deadline (D_i).

C. Hyperparameters

1) Number of Parallel Tasks: Each stream can execute one task at a time, with the total number of streams determining the maximum number of jobs. We use N_c contexts, each containing N_s streams, allowing a maximum of $N_p = N_c \times N_s$ concurrent DNNs. The hyperparameters N_c and N_s define the desired parallel task count and need to be determined. Three policies, based on the context and stream numbers, are discussed in Section V.

TABLE I: Batching performance of different DNNs

DNN	min (JPS)	max (JPS)	batching gain
ResNet18	627	1025	1.63x
ResNet50	250	433	1.73x
UNet	241	260	1.08x
InceptionV3	142	446	3.13x

TABLE II: Task sets

Name	#High	#Low	Task JPS
ResNet18	17	34	30
UNet	5	10	24
InceptionV3	9	18	24

2) Oversubscription Level and Number of SMs: All contexts are allocated an equal number of SMs as:

$$N_{SM} = \text{ceil}_{\text{even}}\left(\frac{OS \times N_{SM,max}}{N_c}\right) \tag{9}$$

where $\text{ceil}_{\text{even}}$ rounds up to the nearest even number, $N_{SM,max}$ is the total GPU SMs, and OS is the oversubscription value, constrained to $1 \leq OS \leq N_c$. $OS = 1$ assigns each context its own SMs, with $OS = N_c$ context will share all SMs, anything in between has a variable level of SM sharing (i.e. lower values reduces interference while higher values enhance utilization through increased sharing).

IV. Real-Time Scheduler

We propose a multi-tenant inference scheduler for real-time DNNs, where each task corresponds to a distinct DNN and is classified as either HP or LP. Our scheduler (Figure 3) has two phases: offline and online. The offline phase sets the initial state, while the online phase is the main body of the scheduler.

A. Offline Phase

In the offline phase, we allocate N_c contexts to tasks. HP tasks are given fixed contexts, while LP tasks can migrate between contexts as needed. This initial assignment prioritizes load balancing across contexts, establishing an efficient starting point for the online phase.

1) Average Full Load Analysis: In the offline phase, with no measurement history, MRET cannot be used. Instead, we compute the *Average Full-Load Execution Time* (AFET) by executing the target task in one stream while randomly executing others in the remaining streams, providing a pessimistic initialization metric. AFET is replaced in later iterations as it does not adapt to recent load trends (existing WCET estimates can be used if desired). Substituting $afet_i$ for $mret_i(t)$ in Equation (3) when $t = 0$, task utilization is computed as:

$$u_i(t) = \begin{cases} \frac{afet_i}{T_i} & \text{if } t = 0 \\ \frac{mret_i(t)}{T_i} & \text{if } t \geq 1 \end{cases} \tag{10}$$

Algorithm 1 Initial Context Assignment

```
1: procedure PopulateContexts( )
2:     // pool : context pool
3:     for all task in highTasks do
4:         ctx ← minUtil(pool)
5:         task.context ← ctx
6:         ctx.totalUtil ← ctx.totalUtil + task.util
7:     end for
8:     for all task in lowTasks do
9:         ctx ← minUtil(pool)
10:        task.context ← ctx
11:        ctx.totalUtil ← ctx.totalUtil + task.util
12:    end for
13: end procedure
```

2) Populating Contexts: The objective is to assign tasks to contexts such that total context utilization ($U_k^t(0)$), HP ($U_k^{h,t}(0)$), and LP ($U_k^{l,t}(0)$) task utilization are balanced across contexts. Algorithm 1 presents the pseudo-code for the initial context assignment. Lines 3 to 7 assigns HP tasks to contexts, while Lines 8 to 12 distributes LP tasks to balance utilization.

B. Online Phase

The goal of the online phase is to promptly schedule HP tasks, minimize deadline misses, and maximize throughput. This phase consists of two main components, discussed in the following sections.

1) Admission Test: A utilization-based admission mechanism for LP tasks is preferable when demand exceeds capacity. LP tasks undergo a utilization-based test within each context. HP tasks reserve a portion of context utilization, leaving the remaining capacity for LP tasks, which are subject to this utilization test. Given multiple streams (N_s) within each context, the utilization test for ctx_k at time t is defined as:

$$U_k^r(t) = N_s - U_k^{h,t}(t) \tag{11}$$

$$U_k^{l,a}(t) + u_j(t) < U_k^r(t) \tag{12}$$

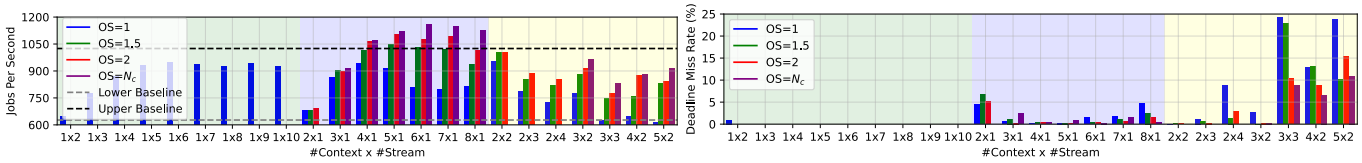
Here, $U_k^r(t)$ is the remaining utilization of context k. If a job satisfies Equation (12) for $k = ctx_i(t)$, it will be scheduled in context k. Otherwise, we will test other contexts ($k \in \{1, \ldots, N_c\} \setminus \{ctx_i(t)\}$) as migrations candidates. If any of them satisfy Equation (12), τ_i will be migrated to the context with earliest predicted finish time. If no context is found that passes the admission test, the task is rejected.

2) Stage Scheduler: We extend task priorities from two to eight fixed levels for stages. Stages from HP tasks always take precedence over LP tasks, with the last stage of each task (τ_{i,n_i}) prioritized higher. Any stage with an immediate preceding missed virtual deadline has the next priority level (e.g., if $\tau_{i,j}$ misses its deadline, $\tau_{i,j+1}$ receives higher priority). This hierarchy ensures HP tasks are serviced first, emphasizes the final stage of each task to prevent overall deadline misses, and prevents cascading misses by prioritizing stages with preceding deadline misses. Finally, EDF within fixed priority levels ensures tasks with approaching deadlines are prioritized.

V. Implementation and Setup

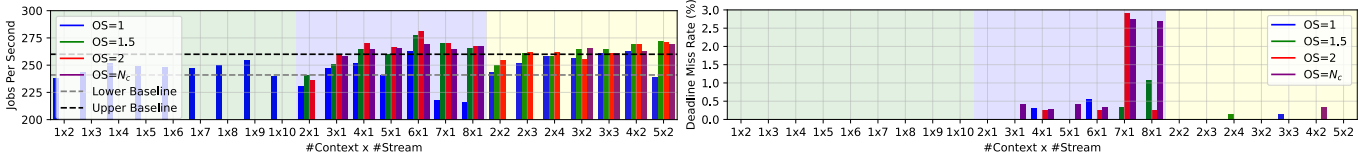
We evaluated DARIS on an RTX 2080 Ti GPU implemented using LibTorch (PyTorch's C++ API). Backend modifications enabled multi-context support via thread-local variables. PyTorch's multi-device API was customized to treat each device as a context while replacing device-level with context-level synchronizations to avoid inter-context locks. Key components for streams, memory, cache, and cuBLAS were updated to support our algorithm.

Our benchmarks include three DNNs: *ResNet* [25], *UNet* [26], and *InceptionV3* [27], each with 224x224x3 input. Throughput gains from batching, evaluated as an upper limit for throughput without colocation, are reported in Figure 1 and Table I (throughput in *Jobs Per Second* (JPS)). ResNet, with a linear design, is widely used, while UNet's wide architecture and skip connections make it memory-heavy, achieving only a 1.08x from batching. InceptionV3, with multiple parallel paths, exhibits the highest batching gain of 3.13x, highlighting its reliance on batching.



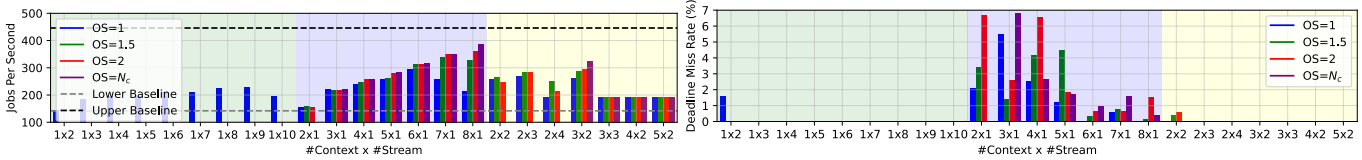
(a) Throughput (b) LP Deadline Misses

Fig. 4: Scheduling results for ResNet18 task set (STR , MPS , MPS+STR)



(a) Throughput (b) LP Deadline Misses

Fig. 5: Scheduling results for UNet task set (STR , MPS , MPS+STR)



(a) Throughput (b) LP Deadline Misses

Fig. 6: Scheduling results for InceptionV3 (STR , MPS , MPS+STR)

We use MPS and CUDA Streams with oversubscription to colocate DNNs, aiming to minimize deadline misses and maximize throughput. DARIS variations are assessed using three partitioning policies:

1) ***STR***: Exclusively uses streams for scheduling DNNs.
2) ***MPS***: Relies solely on MPS for scheduling DNNs.
3) ***MPS+STR***: Combines MPS and streams to assess their joint effect on timeliness and throughput.

We schedule 2–10 parallel DNNs ($2 \leq N_p \leq 10$). Four oversubscription options are explored: $OS \in \{1, 1.5, 2, N_c\}$, where $OS = 1$ indicates no SM sharing and $OS = N_c$ represents full SM sharing.

Performance is assessed using total JPS and Deadline Miss Rate (DMR) for throughput and timeliness, with configurations denoted as $N_c \times N_s$ or $N_c \times N_s_OS$. Three main task sets (Table II), each tied to a DNN type, are tested with 30 jobs/sec for ResNet18 and 24 jobs/sec for others. Experiments are run under 150% overload, using the upper baseline as full load due to varying maximum loads across different configurations. A 2:1 LP-to-HP task ratio is maintained, with alternative ratios explored in Section VI-I.

VI. Experimental Results

Main scenario results are presented in Figures 4 to 6. Throughput figures (Figures 4a to 6a) compare with lower (single DNN) and upper (pure batching) baseline throughput from Table I. Although meeting every deadline is not guaranteed, we did not observe any HP deadline misses, and only LP DMRs are shown in Figures 4b to 6b. DMR is the ratio of missed deadlines to accepted jobs.

1) Throughput: As shown in Figures 4a to 6a, *MPS* delivers the highest throughput across all DNNs. ResNet18 and UNet achieve peak throughput at $N_c = 6$, while InceptionV3 benefits from increased concurrency up to $N_c = 8$. The *MPS+STR* policy outperforms *STR*, showing that multiple contexts enhance throughput on MPS-enabled GPUs. For ResNet18 and UNet, we exceeded the pure batching baseline (Table I) by 13% (1158 JPS vs. 1025 JPS) and 8% (281 JPS vs. 260 JPS) respectively, without batching. However, for InceptionV3, we only achieved 87% of its upper baseline. Its complex, narrow architecture limits throughput. We conducted an experience to release parallel paths of InceptionV3 using distinct streams to improve throughput but gained only 9% in throughput still below the upper baseline. By using batching instead, we were able to surpass upper baseline for InceptionV3 which will be discussed in Section VI-H.

A. Deadline Misses

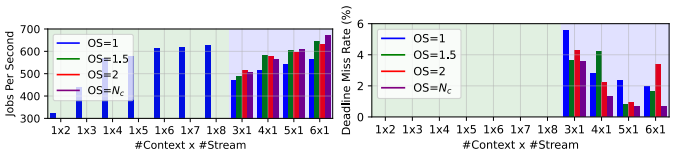
DARIS prioritizes minimizing HP task response time to reduce the chance of deadline misses, though it does not guarantee meeting every deadline. In our main experiments, no HP deadline misses were observed. From here on, DMR refers to LP DMR unless stated otherwise. Except for less than 2% DMR in the 1×2 configuration for InceptionV3, no deadline misses occurred with the *STR* policy. Each context uses a dedicated job queue, leaving *STR* with a single global queue. While this may reduce throughput, it ensures optimal timeliness. UNet showed the lowest DMR, peaking at less than 3%, with only 0.25% at its best throughput configuration ($6 \times 1_2$). The *MPS+STR* policy produced the worst DMR, reaching 25% for ResNet18. Meanwhile, with the *MPS* policy, both ResNet18 and InceptionV3 maintained DMRs below 7%, with around 2% at their peak throughput configurations ($6 \times 1_6$ and $8 \times 1_8$, respectively).

B. Comparison with State-of-the-Art

GSlice [10] is a state-of-the-art inference server, achieving 1152 JPS for ResNet50 with batching and 1193 JPS with GSlice—a 3.5% gain. In comparison, we achieved 433 JPS with batching and 498 JPS using DARIS on our hardware, improving throughput by 15% over batching and 11.5% over GSlice. Without oversubscription, DARIS throughput drops to 374 JPS, 8% below batching. Inference servers like [10], [14] generally allow some deadline misses without detailed rates. The closest comparison, [15], reports no HP deadline misses and up to 12% for LP tasks. In our case, LP deadline misses are below 2% in the best and under 7% in the worst scenario with the *MPS* policy, and zero with the *STR* policy. RTGPU [28] lacks task prioritization, reporting up to 11% overall deadline misses.

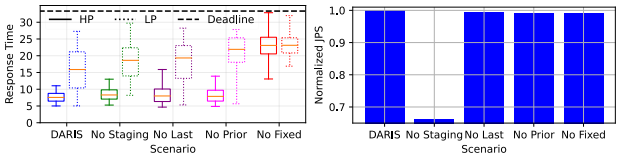
C. DARIS Policy Comparison

Overall observations indicate that the *MPS* policy offers the best throughput, while *STR* results in ideal DMR. Although *MPS* may not



(a) Throughput (b) LP Deadline Misses

Fig. 7: Scheduling results for mixed task set (STR , MPS)



(a) Response time (b) Normalized Throughput

Fig. 8: Response time and throughput for different scenarios

achieve the lowest DMR, it remains below 7% in all configurations. For many applications, this DMR is acceptable, but for those with stringent constraints, *STR* is the safest policy. In scenarios with embedded GPUs lacking MPS support, *STR* is the sole feasible option. Conversely, the *MPS+STR* policy yields the least favorable outcomes, with subpar throughput and DMR across all configurations. Also, UNet has the most consistent performance in all scenarios, making it the least sensitive DNN to concurrency mechanisms and configurations.

D. Mixed Task Set

It is essential to analyze a mixed task set with all DNN types, reflecting real-world scenarios. The results for the mixed task set are shown in Figure 7. As with individual task sets, the *MPS* policy achieves the highest throughput, while the *STR* policy offers the most reliable deadline performance.

E. Oversubscribing SMs

Contrary to some claims in the literature [18]–[20], our study shows that, in real-time scenarios without batching, oversubscription significantly boosts throughput. While its impact on timeliness is not strictly monotonic, higher OS values generally result in fewer deadline misses. Isolating SMs ($OS = 1$) leads to a sharp drop in throughput. Our findings confirm that oversubscription consistently benefits both throughput and timeliness. For wide DNNs like UNet, which gain little from batching, 200% SM oversubscription is sufficient, while narrower DNNs require even higher oversubscription. Although more concurrent DNNs combined with oversubscription can increase resource contention, it also enhances throughput. The goal is to identify a trade-off point to maximize benefit.

F. DARIS Modules Contribution

Works addressing real-time constraints often treat meeting deadlines or minimizing latency as binary outcomes [10], [14], [15], [28]. However, enhancing the quality of service by minimizing response time is crucial for better responsiveness. We conducted experiments to evaluate how different DARIS modules impact overall performance. Figure 8a shows the response times, and Figure 8b presents normalized throughput for DARIS and four alternative scenarios using ResNet18:

- *No Staging*: tasks treated as whole units without staging
- *No Last*: last stages of tasks are not prioritized
- *No Prior*: no high priority when preceding stage misses deadline
- *No Fixed*: no differentiation in task priority among stages

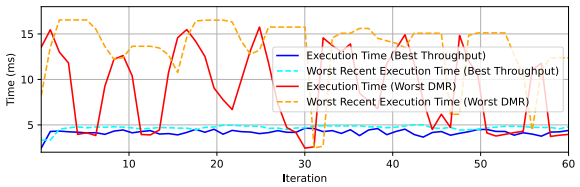


Fig. 9: Execution time and MRET of ResNet18

The original DARIS achieves response times of 5–12 ms for HP tasks and 5–27.5 ms for LP tasks, making HP tasks finish roughly 2.5 times faster. In *No Staging* scenario, throughput drops by 33%, and response times increase due to the lack of preemption, with 5.5% and 22.5% deadline misses for HP and LP tasks, respectively. *No Last* scenario increases worst-case response times for HP tasks by 38% without significantly affecting throughput. In *No Prior* scenario, average response times rise for all tasks. *No Fixed* scenario, which removes inter-task priority differentiation, results in a 2.5% deadline miss rate for both task priorities. These findings highlight the effectiveness of DARIS modules, emphasizing the importance of *staging* and *task priority* in improving throughput and meeting deadlines.

G. Maximum Recent Execution Time

We introduced MRET to address the unpredictability of GPU execution time and avoid the pessimism of history-based WCET approaches. Figure 9 shows the actual execution time and MRET for ResNet18 under the best throughput configuration ($6 \times 1_6$) and the worst deadline miss rate configuration ($3 \times 3_1$). We selected a window size (ws) of 5, as smaller values increase DMR, while larger values reduce throughput. With the $6 \times 1_6$ configuration, MRET accurately predicts execution time in most cases, whereas in the $3 \times 3_1$ configuration, execution time often exceeds MRET predictions.

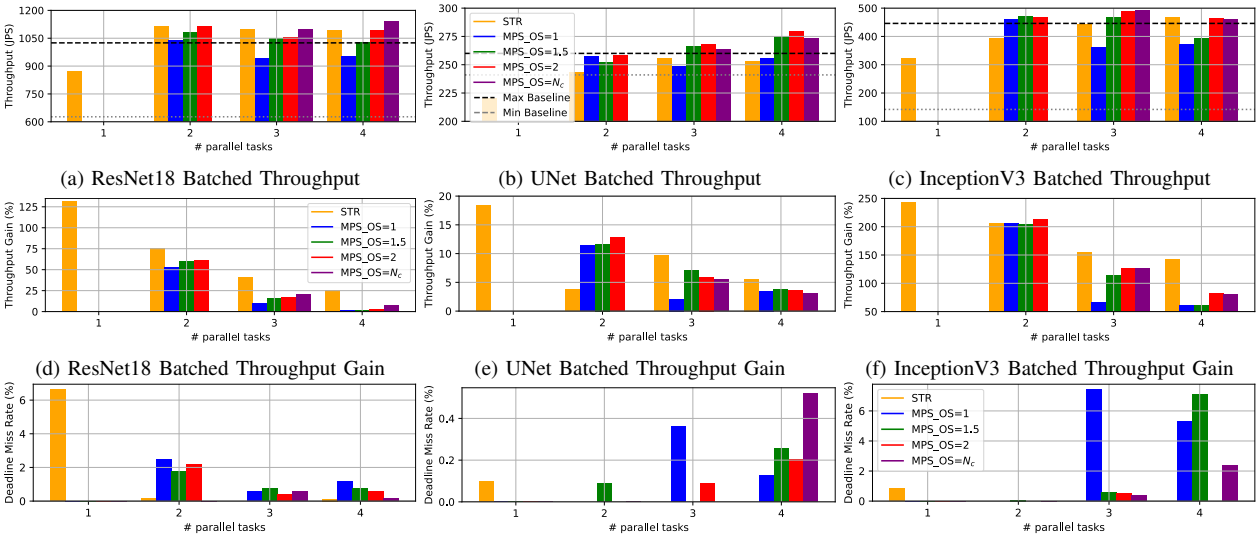
H. Batching

We conducted an additional experiment to demonstrate the impact of batching combined with DARIS, using batch sizes of 4, 2, and 8 for ResNet18, UNet, and InceptionV3, respectively. Figures 10a to 10c present the throughput results. Our key observation is that fewer parallel tasks are needed to exceed the upper baseline, with decent throughput achieved even with 1 or 2 parallel tasks. We also observed similar benefits from SM oversubscription compared to the main experiment (Figures 4 to 6) without batching, though the difference is less pronounced.

Figures 10d to 10f show the throughput improvements compared to similar configurations from the main experiment (Figures 4a to 6a). UNet exhibits the smallest gain, with improvements up to 18%, reaffirming its efficient architecture in utilizing GPU resources even without batching. In contrast, InceptionV3 achieves the highest improvement, with at least a 55% increase over the main experiment without batching. For DMR (Figures 10g to 10i), ResNet18 and InceptionV3 show slightly better DMRs compared to the main experiment (Figure 4band Figure 6b). UNet demonstrates the most significant improvement, with its DMR reduced to under 0.5%.

I. Overloading And Task Ratio

We conducted a final experiment to examine the scheduler's behavior under different HP-to-LP task ratios. Figure 11 presents the results for throughput and deadline miss rates for both priorities, using ResNet18 and UNet in full load and overloaded scenarios. We also evaluate an overloaded scenario where HP tasks must pass the admission test (*Overload+HPA*).

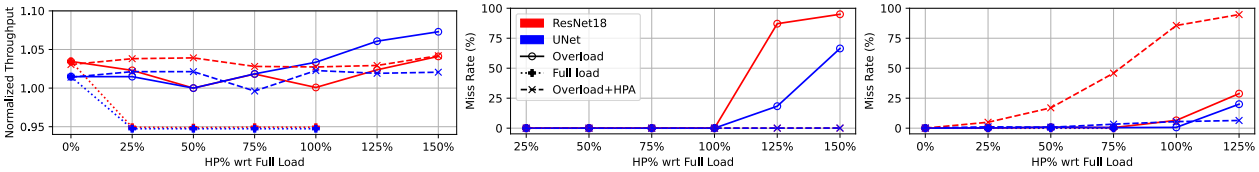


(a) ResNet18 Batched Throughput (b) UNet Batched Throughput (c) InceptionV3 Batched Throughput

(d) ResNet18 Batched Throughput Gain (e) UNet Batched Throughput Gain (f) InceptionV3 Batched Throughput Gain

(g) ResNet18 LP DMR (h) UNet LP DMR (i) InceptionV3 LP DMR

Fig. 10: Absolute Throughput When Batching



(a) Normalized Throughput (b) High-Priority DMR (c) Low-Priority DMR

Fig. 11: Overloading with different HP to LP ratios

As shown in Figure 11a, throughput remains stable across different task ratios and overload scenarios. In the full load scenario, throughput drops consistently by 5% with the presence of LP tasks, but no HP or LP deadline misses occur. However, in the overload scenario, when HP load exceeds 100% of full capacity, DMR for HP tasks rises sharply since HP tasks are scheduled without an admission test. Although this ensures all HP tasks are scheduled, it results in exponentially increasing deadline misses when the system is overwhelmed by HP tasks.

To address this, we introduce *Overload+HPA*, ensuring zero deadline misses for HP tasks by applying the admission test, even when the system is overloaded with HP tasks. The trade-off is that some HP tasks may be dropped, and DMR for LP tasks increases. However, UNet avoids this downside due to its wide efficient architecture. We recommend limiting HP tasks to 50% of the full load to minimize deadline misses, as shown in the main experiments. In cases of overloaded HP tasks, using the *Overload+HPA* approach ensures safer scheduling with minimal HP deadline misses.

VII. Conclusions

In this work, we presented DARIS, a novel real-time GPU scheduler designed for periodic tasks in a multi-tenant DNN setup with task priorities. DARIS applies oversubscribed spatio-temporal scheduling to minimize deadline misses for high-priority tasks while maximizing throughput by scheduling as many low-priority tasks as possible. For spatial partitioning, we used both MPS and CUDA streams to colocate multiple DNNs. Through extensive experiments, we compared the throughput and timeliness achieved by using MPS and streaming separately and in combination. We found that MPS provides the highest throughput, while streaming yields the lowest deadline miss rate. Additionally, we evaluated DARIS under batched inputs, overloaded scenarios, and various task ratios. Our experiments showed that oversubscribing SMs can boost throughput significantly, even in the presence of batched inputs and DNNs with wide architecture.

For temporal scheduling, we used staging as synchronization points to enable coarse-grained preemption, improving throughput and reducing deadline misses. By prioritizing stages based on task priority, stage order, and past missed deadlines, we further reduced deadline misses and enhanced response times to improve the quality of service. While we explored leveraging the parallel paths of non-linear DNNs to boost throughput, we found that batched inputs provided superior results in managing parallel paths.

Acknowledgment

This work was supported in part by NSF under grant number CPS-2038726 and by the Commonwealth Cyber Initiative, an investment in the advancement of cyber R&D, innovation, and workforce development. For more information about CCI, visit www.cyberinitiative.org.

References

[1] X. He, J. Liu, Z. Xie, *et al.*, "Enabling energy-efficient dnn training on hybrid gpu-fpga accelerators," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, pp. 227–241.

[2] W. Xu, Y. Zhang, and X. Tang, "Parallelizing dnn training on gpus: Challenges and opportunities," in *Companion Proceedings of the Web Conference 2021*, 2021, pp. 174–178.

[3] Z. Ye, W. Gao, Q. Hu, *et al.*, "Deep learning workload scheduling in gpu datacenters: A survey," *ACM Computing Surveys*, vol. 56, no. 6, pp. 1–38, 2024.

[4] L. Liu, Z. Dong, Y. Wang, and W. Shi, "Prophet: Realizing a predictable real-time perception pipeline for autonomous vehicles," in *2022 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2022, pp. 305–317.

[5] S. Roshan, J. Tanha, M. Zarrin, A. F. Babaei, H. Nikkhah, and Z. Jafari, "A deep ensemble medical image segmentation with novel sampling method and loss function," *Computers in Biology and Medicine*, p. 108 305, 2024.

[6] S. H. Mostafaei, J. Tanha, A. Sharafkhaneh, Z. H. Mostafaei, M. H. A. Al-Jaf, and A. F. Babaei, "An ensemble model for sleep stages classification," in *2023 31st International Conference on Electrical Engineering (ICEE)*, IEEE, 2023, pp. 327–332.

[7] Z. Aghapour, S. Sharifian, and H. Taheri, "Task offloading and resource allocation algorithm based on deep reinforcement learning for distributed ai execution tasks in iot edge computing environments," *Computer Networks*, vol. 223, p. 109 577, 2023.

[8] Y. Li, Z. Li, W. Yang, and C. Liu, "Rt-lm: Uncertainty-aware resource management for real-time inference of language models," *arXiv preprint arXiv:2309.06619*, 2023.

[9] A. F. Babaei, J. Tanha, A. F. Babaei, and M. G. Matin, "A hybrid deep learning model for 5-digit handwritten recognition," in *2024 10th International Conference on Artificial Intelligence and Robotics (QICAR)*, IEEE, 2024, pp. 329–333.

[10] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan, "Gslice: Controlled spatial sharing of gpus for a scalable inference platform," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020, pp. 492–506.

[11] K. Guo, Y. Xu, Z. Qi, and H. Guan, "Optimum: Runtime optimization for multiple mixed model deployment deep learning inference," *Journal of Systems Architecture*, vol. 141, p. 102 901, 2023.

[12] F. Yu, D. Wang, L. Shangguan, M. Zhang, C. Liu, and X. Chen, "A survey of multi-tenant deep learning inference on gpu," *arXiv preprint arXiv:2203.09040*, 2022.

[13] W. Zhang, W. Cui, K. Fu, *et al.*, "Laius: Towards latency awareness and improved utilization of spatial multitasking accelerators in datacenters," in *Proceedings of the ACM international conference on supercomputing*, 2019, pp. 58–68.

[14] A. Gujarati, R. Karimi, S. Alzayat, *et al.*, "Serving {dnns} like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 443–462.

[15] Y. Wang, M. Karimi, Y. Xiang, and H. Kim, "Balancing energy efficiency and real-time performance in gpu scheduling," in *2021 IEEE Real-Time Systems Symposium (RTSS)*, IEEE, 2021, pp. 110–122.

[16] A. F. Babaei and T. Chantem, "Sgprs: Seamless gpu partitioning real-time scheduler for periodic deep learning workloads," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2024, pp. 1–2.

[17] NVIDIA Corporation, *NVIDIA Multi-Process Service Documentation*, https://docs.nvidia.com/deploy/mps/index.html.

[18] A. Dhakal, K. Ramakrishnan, S. G. Kulkarni, P. Sharma, and J. Cho, "Slice-tune: A system for high performance dnn autotuning," in *Proceedings of the 23rd ACM/IFIP International Middleware Conference*, 2022, pp. 228–240.

[19] A. Dhakal, S. G. Kulkarni, and K. Ramakrishnan, "D-stack: High throughput dnn inference by effective multiplexing and spatio-temporal scheduling of gpus," *arXiv preprint arXiv:2304.13541*, 2023.

[20] G. Rattihalli, N. Hogade, A. Dhakal, *et al.*, "Fine-grained heterogeneous execution framework with energy aware scheduling," in *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, IEEE, 2023, pp. 35–44.

[21] M. Han, H. Zhang, R. Chen, and H. Chen, "Microsecond-scale preemption for concurrent {gpu-accelerated}{dnn} inferences," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 539–558.

[22] Z. Bai, Z. Zhang, Y. Zhu, and X. Jin, "{Pipeswitch}: Fast pipelined context switching for deep learning applications," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020, pp. 499–514.

[23] H. Zhang, Y. Tang, A. Khandelwal, and I. Stoica, "{Shepherd}: Serving {dnns} in the wild," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 787–808.

[24] Z. Xia, Y. Hao, J. Duan, C. Wang, and J. Jiang, "Towards optimal preemptive gpu time-sharing for edge model serving," in *Proceedings of the 9th International Workshop on Container Technologies and Container Clouds*, 2023, pp. 13–18.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[26] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, Springer, 2015, pp. 234–241.

[27] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[28] A. Zou, J. Li, C. D. Gill, and X. Zhang, "Rtgpu: Real-time gpu scheduling of hard deadline parallel tasks with fine-grain utilization," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 5, pp. 1450–1465, 2023.

Multicore Environment State Representation for Agent-Directed Test Generation

Bruno D. Miranda, Luiz M. V. Pereira, Márcio Castro, Luiz C. V. dos Santos
Federal University of Santa Catarina
Florianópolis, SC, Brazil
b.miranda@posgrad.ufsc.br; luiz.m.v.p@grad.ufsc.br; {marcio.castro, luiz.santos}@ufsc.br

***Abstract*—A crucial step in the design of multicore systems is to validate the interaction between cores. This involves test program generation and runtime analysis. We propose a novel reinforcement learning approach to directed test generation, where an agent induces a suite of programs, which are executed in a simulation environment for a multicore. It focuses on how to recover state information from raw observations of the environment such that the agent can learn from interaction how to improve coverage for any verification task. We evaluated our state representation for different verification tasks involving 16 and 32-core ARMv8 2-level MOESI designs.**

***Index Terms*—Reinforcement learning, test generation.**

I. Introduction

The verification of the interactions between cores in multicore systems requires the execution of concurrent test programs until coverage goals are met, and no errors are left in the design. Such interactions through shared memory must obey a *Memory Consistency Model* (MCM) [1], [2], which allows many valid behaviors for the same concurrent program, leading to multiple correct outcomes for the same input stimuli. The lack of a one-to-one golden model for the expected behavior makes the generation of self-checking tests very complex [3], [4].

A pragmatic alternative is to split verification into complementary tasks: a generator produces a suite of test programs, while an external checker analyzes whether or not the observed behavior during the execution of each program complies with the MCM [5]–[9]. This allows the generator to focus on coverage improvement, whereas the checker focuses on error diagnosis. This paper follows such strategy for test generation.

Various approaches to *Directed Test Generation* (DTG) have been reported for the target verification domain, such as Bayesian networks [10], Euler tour [11], [12], genetic programming [13], local search [14], and reinforcement learning [15].

Reinforcement learning (RL) involves the interaction between an *agent* and an *environment*. The agent receives a *reward* signal, and takes *actions* depending on the environment *state*. In many practical cases, the agent can only observe partial information about the state. The choice of proper *observations* depends on the target environment, and it impacts performance.

This paper proposes a novel reinforcement learning approach to DTG which focuses on proper state representation for environments containing a *design under verification* (DUV) that is a multicore system. We show (1) what should be observed, (2) how observations should be represented, and (3) how they should be used to recover state information.

Firstly, we propose the notion of *witness* to execution instances of a concurrent program as a proper key to observing the response of a multicore environment to an action taken by an agent. To define a witness, we repurposed the well-known *reads-from* relation [16] for recovering state information. Despite its standard use for the synthesis of litmus tests [16], [17], that relation has never been cast before into a state representation for enabling RL agents to *direct* test generation.

Secondly, we show that test generation can be conveniently constrained to enable a provenly *unique representation* for every observable execution witness induced by a test suite.

Finally, we show how environment state information can be recovered from the bounded *history of action-observation pairs*.

This paper is organized as follows. Section II discusses our motivation in face of related work. Section III describes the proposed approach to DTG. Section IV formulates our proposal for suitable observation, proper representation, and state recovery. Section V reports our experimental evaluation, and Section VI draws overall conclusions.

II. Motivation in face of related work

Since model-based DTG [11], [12] is usually tied to protocol or architecture, data-driven approaches are preferred [10], [13]–[15]. One of them [10] dynamically redefines directives for a random test generation engine by means of statistical inference. That approach is based on modeling the relationship between coverage information and test directives as Bayesian networks. MCjammer [18] employs multiple agents in distinct core domains that generate concurrent memory access patterns. MCjammer is not a test program generator, but a data-driven verification environment that directly generates stimuli based on the coverage goals of multiple agents. The McVersi Test Generator (MTG) [13] relies on Genetic Programming, and it led to superior coverage and error discovery as compared to random test generation and litmus tests. MTG represents the state of the art for evolutionary DTG. The Reinforcement Learning Generator (RLG) [15] is the only RL approach to DTG reported so far for the target domain, and it obtained results competitive with MTG. Thus, MTG and RLG are qualified baselines for the evaluation of novel DTG approaches.

This work was financed in part by the Conselho Nacional de Desenvolvimento Científico e Tecnológico - Brasil (CNPq) - grants 307093/2021-2 (PQ), 308933/2020-6 (PQ) - and by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Most DTG techniques are *coverage-directed* test generators [10], [13]–[15], where coverage is the only observation that is tracked from the DUV to direct the generation of new tests. However, coverage is not an injection, since the same fraction of the coverage goals may correspond to different subsets with same cardinality. Although the cumulative coverage captures the evolution of the verification process, it does not accurately track the actual trajectory of responses to the induced stimuli.

Thus, to be improved, DTG needs extra observations from the DUV, but such information is usually restricted to external checkers, because most generators rely on them for being able to focus on coverage improvement. Such extra observations should not be tied to MCM, coherence protocol or microarchitecture to avoid dependence on a specific verification task.

Such DUV-agnostic observations are naturally exploited by litmus tests [2]. Since litmus test generation [16], [17] cannot provide coverage guarantees [13], and coverage-directed generation is restricted due to limited observation, proper observations should be extracted from the DUV to guide novel approaches to DTG. This motivates the use of a sequence of domain-specific observations as a trajectory that can be used to recover the RL notion of environment state [19].

Successful applications of RL used to require handcrafted tuning [20] to each specific problem until the Deep Q-Network (DQN) approach [21] has shown that the combination of Q-learning [22], deep neural network, and experience replay [23] could handle a collection of different problems (video games) of the same application domain (Atari suite) without having to resort to problem-specific tuning. High performance was obtained with the same hyper-parameters, network architecture, and observations (raw video frames). Since then, several extensions to DQN have been reported [24].

This feature of recent agents is attractive to the verification of multicore systems, where different tasks are induced by a variety of coherence protocols, core counts, microarchitectures, and coverage metrics. To benefit from RL, the raw input for the agent should capture primal shared-memory behaviors, which are specified as *order relations* in formal frameworks for litmus test generation [16], [17] and MCM checking [6], [9]. Among them, the *reads-from* relation is particularly convenient, because it allows order to be inferred from observed values. The use of values for representing order was proposed as part of a profiling technique [25], where a signature encoded the order of basic blocks in an execution path of a sequential program. That idea inspired a technique for trace compaction [26], where signatures were used to capture the order of memory operations induced by concurrent programs. Such compaction technique inspired us to rely on signatures for environment *state recovery*, enabling the use of RL agents for the target verification domain.

III. The proposed approach

A. Background in brief

The interaction between agent and environment occurs at discrete time steps $0, 1, 2, \cdots, t$. At a given time step t, the agent receives some representation of the environment state, S_t, and it responds with some action A_t. In the next time step, the environment sends to the agent a scalar reward signal R_{t+1}, and evolves to a new state S_{t+1}. Both R_{t+1} and S_{t+1} are determined as a consequence of action A_t. The agent learns from interaction how to take actions on the environment in order to maximize expected rewards. Thus, the interaction between agent and environment can be expressed as $<S_0, A_0, S_1, R_1, A_1, S_2, R_2, A_2, S_3, R_3, \cdots, A_{t-1}, S_t, R_t>$.

The interaction is usually formalized as a Markov decision process, where R_{t+1} and S_{t+1} can be defined only in terms of S_t and A_t. This means that a state must be able to completely capture all the interaction history that could be relevant for future decision making [19]. Such ideal property often does not hold in practice, because the agent observes only partial information about the environment state. Thus, some technique is required to approximate a Markov state from observations.

To handle partial observability, we do not assume that the environment emits the actual state, but it exposes some observation of signals that depends on that state. We also assume that rewards can be obtained from observations. Under these assumptions, the interaction can be expressed as an alternating sequence of actions A_{t-1} and observations O_t, as follows: $<A_0, O_1, A_1, O_2, A_2, O_3, \cdots, A_{t-1}, O_t>$.

B. Agent-based approach to DTG

Let P_t denote a concurrent *test* program, and let $P_1, P_2, \cdots, P_t, \cdots, P_T$ denote a *test suite*.

Typically, each concurrent program is executed a certain number of times to expose multiple *execution instances* that are likely to exhibit distinct shared-memory behavior. Let I denote a set of execution instances induced by a given program, which are obtained by running the program $|I|$ times, in such a way that each execution is differently perturbed due to the non-determinism of thread concurrency.

Coverage is quantified as a fraction of the coverage goals reached during the execution of the programs of a test suite. Let C_{t+1} be the cumulative coverage obtained by running I execution instances of every program of the sequence $P_1, P_2, \cdots, P_t$.

DTG aims to synthesize a test suite $P_1, P_2, \cdots, P_t, \cdots, P_T$ that maximizes C_{T+1}. To cast DTG as an RL problem, we introduce the notion of *verification episode*, which comprises successive epochs, each defined by discrete time steps. Each epoch between steps t and $t+1$ consists of three phases: test generation, multiple execution, and coverage evaluation, as follows. An action A_t induces the generation of a program P_t. Then P_t is executed $|I|$ times in the DUV. Finally, C_{t+1} is updated with the *individual* contribution of P_t to the cumulative coverage (i.e. the joint contribution of all its $|I|$ execution instances). This is used to define the reward obtained with P_t.

Definition 1: A *reward* is a function $\rho : \mathcal{R} \times \mathcal{R} \mapsto \mathcal{R}$ such that $R_{t+1} = \rho(C_{t+1}, C_t)$.

Fig. 1 illustrates our RL approach to DTG. The environment includes the DUV, a *test generation engine* (TGE), and a *coverage engine* (CVE), which computes the fraction of the coverage goals achieved by all instances of every program executed so far. Between environment and agent, three modules compute functions that convert actual environment information into the conventional interface expected by an RL agent.

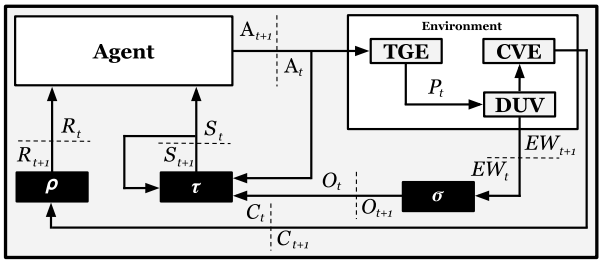


Fig. 1: A reinforcement learning approach to DTG

Dotted lines represent temporal barriers to indicate the evolution between successive time steps. At a given time step t, when the environment is in state S_t, the action A_t induces the generation of a test program P_t. Then multiple instances of that program are executed in the DUV. At time step $t+1$, monitors in the DUV provide a *witness* EW_{t+1} to those executions, and the coverage engine provides the cumulative coverage C_{t+1}.

The function ρ converts coverage information into a reward R_{t+1}, and the function σ converts witnessed information into an observation O_{t+1}. Both R_{t+1} and O_{t+1} are jointly obtained from the multiple executions of the test launched at step t.

The function τ computes a transition from the state S_t to the following state S_{t+1} for a given action-observation pair A_t, O_t.

The notion of execution witness and the functions σ and τ are the keys to our state representation, as follows.

IV. The proposed state representation

Let us define *what* should be observed (to properly capture the environment's activity) and *how* it should be encoded (to ensure a non-ambiguous, compact representation).

A. Choice of suitable observation and proper representation

Let $\mathcal{OP}$ be the set of memory operations of a program. Let S^i_j denote that store operation $S_j \in \mathcal{OP}$ executes in core i. Let L^k_m denote that load operation $L_m \in \mathcal{OP}$ executes in core k.

Each test may induce different execution instances of a concurrent program, which can be seen as distinct orderings of the memory operations of a concurrent program [27].

Definition 2: An *execution instance*, written $<$, is a total order on the set $\mathcal{OP}$.

An execution instance $<$ determines which value is observed by each load. Let Val^0 denote the initial value assigned to some location. Let $Val[S]$ denote the value written by some store S, and let $Val_<[L]$ be the value read by some load L.

An execution instance $<$ induces the following relation.

Definition 3: L and S are in the *reads-from* relation, written L rf S, iff $Val_<[L^k] = Val[S^i] \neq Val^0$.

The outcome of a non-synchronized concurrent program may serve as witness to a given execution instance. First, let us define a witness from the perspective of some core i. Such local witness relies on the outcome induced by the set $\mathcal{L}^i$ of load operations belonging to the thread executed by that core.

Definition 4: A core *witness* to execution instance $<$ is $EW^i_< = \{v : v = Val_<[L] \wedge L \in \mathcal{L}^i\}$.

Note that a witness is determined by which store (if any) is observed by each load: if there exists some S such that L rf S, then $Val_<[L] = Val[S]$, otherwise $Val_<[L] = Val^0$.

Now let us define a core witness induced by a set I of execution instances of the same test program.

Definition 5: A core *witness* to execution instances I is:

$$EW^i_I = \bigcup_{< \in I} EW^i_< \ .$$

Let us define an aggregation of the witnesses at each core.

Definition 6: A multicore *witness* to execution instances I is $EW_I = (EW^1_I, EW^2_I, \cdots, EW^i_I, \cdots, EW^p_I)$.

To properly encode execution witnesses, let us constrain store value assignment: different natural numbers must be assigned to distinct stores, and 0 is reserved for initial values, as follows.

Constraint 1: $Val[S_j] \in \mathcal{N}$ for all $S_j \in \mathcal{OP}$.

Constraint 2: $Val^0 = 0$ for all locations.

Constraint 3: $Val[S_j] \neq Val[S_{m \neq j}] > 0$.

Such constraints are commonly applied to test generation, because unique values serve as non-ambiguous identifiers for stores, thereby reducing the complexity of MCM checkers [6].

A straightforward way to satisfy Constraint 3 *would* be to assign the next natural number to the next synthesized store, during the generation of an *individual test* program. However, to rule out ambiguity when observing execution witnesses induced by *different* programs, let us define a unified assignment policy for all programs of a *test suite*. The key is to assign unique values to stores, independently of the number of stores chosen for each thread. Let θ denote an upper bound for the number of stores in a thread, and let S^i_j be the j-th store of a given thread i. The following implies Constraint 3.

Constraint 4: $Val[S^i_j] = \theta(i-1)+j$ and $1 \leq j \leq \theta$ for all tests.

To encode witnesses under those constraints, let $\mathcal{EW}^i = \{EW^i_<\}$ denote the witnesses induced by every possible execution instance $<$, and let $\mathcal{P}(\mathcal{EW}^i)$ be its power set.

Definition 7: The signature of a witness to execution instances I at core i is a function $sig : \mathcal{P}(\mathcal{EW}^i) \mapsto \mathcal{N}$ such that:

$$sig(EW^i_I) = \sum_{v \in EW^i_I} v$$

Let us prove that a signature encodes a core witness properly.

Lemma 1: Under Constraints 1, 2, and 4, sig is an injection.

Proof: Let us consider the witnesses $EW^i_I, EW^i_{I'} \in \mathcal{P}(\mathcal{EW}^i)$, and let us assume that they are different. $EW^i_I \neq EW^i_{I'}$ implies $I \neq I'$. Thus, I and I' must have each at least one execution instance, say $<$ and $\prec$, which is not a member of the other, i.e.: $\{<\} \subseteq I \setminus I'$ and $\{\prec\} \subseteq I' \setminus I$. Therefore, $EW^i_I \neq EW^i_{I'}$ implies $EW^i_< \neq EW^i_\prec$ for some $<$ and $\prec$. If $EW^i_< \neq EW^i_\prec$ holds, there must exist at least one load $L \in \mathcal{L}^i$ such that $Val_<[L] = Val[S]$ and $Val_\prec[L] = Val[S']$ or $Val_\prec[L] = Val^o$. Since Constraint 4 implies Constraint 3, under Constraints 1, 2, and 3, we have $Val_<[L] \neq Val_\prec[L]$. For convenience, let $U = \{u : u \in EW^i_{I \cap I'}\}$, $V = \{v : v \in EW^i_{I \setminus I'}\}$, and $V' = \{v' : v' \in EW_{I' \setminus I}\}$. From Definition 7, we have: $sig(EW^i_I) = \sum_{u \in U} u + \sum_{v \in V} v$ and $sig(EW^i_{I'}) = \sum_{u \in U} u + \sum_{v' \in V'} v'$. Since $Val_<[L] \neq Val_\prec[L]$ holds for some L, it implies $\sum_{v \in V} v \neq \sum_{v' \in V'} v'$, which

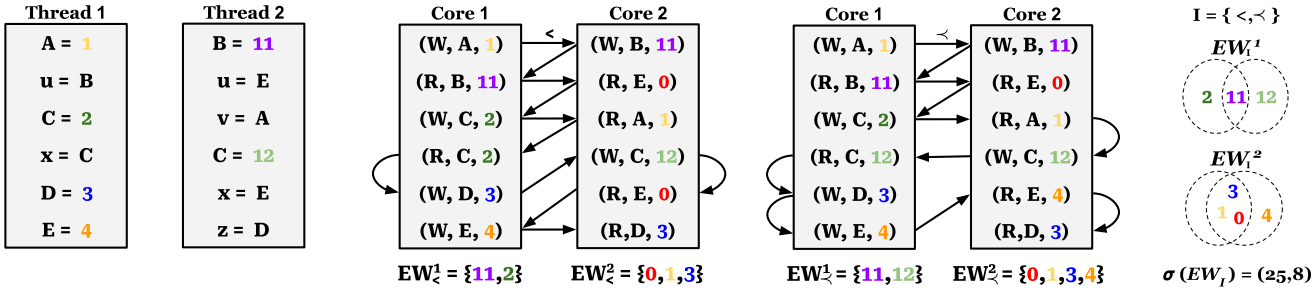


Fig. 2: An observation captured by a joint witness to execution instances $I = \{<, \prec\}$, and its representation as a signature.

leads to distinct signatures. Since $EW_I^i \neq EW_{I'}^i$ implies $sig(EW_I^i) \neq sig(EW_{I'}^i)$, sig is an injection. ■

Let $\mathcal{EW} = \{EW_<\}$ be the set of *multicore* witnesses induced by every possible execution instance $<$.

Definition 8: The signature of a multicore witness to execution instances I is a function $\sigma : \mathcal{P}(\mathcal{EW}) \mapsto \mathcal{N}^p$ such that: $\sigma(EW_I) = (sig(EW_I^1), sig(EW_I^2), \cdots, sig(EW_I^p))$.

Theorem 1: Under Constraints 1, 2, and 4, σ is an injection.

Proof: Let us consider the witnesses $EW_I, EW_{I'} \in \mathcal{P}(\mathcal{EW})$, and assume that they are different. If $EW_I \neq EW_{I'}$ holds, there must exist at least one core i such $EW_I^i \neq EW_{I'}^i$, which from Lemma 1 implies $sig(EW_I^i) \neq sig(EW_{I'}^i)$. From Definition 8, this is sufficient to conclude that $\sigma(EW_I) \neq \sigma(EW_{I'})$. Since $EW_I \neq EW_{I'}$ implies $\sigma(EW_I) \neq \sigma(EW_{I'})$, σ is an injection. ■

Theorem 1 guarantees that signatures are *unique* representations for observations of a multicore DUV.

B. Illustrative example

Fig. 2 illustrates the proposed representation for two execution witnesses of the same test program. Fig. 2(a) shows a test program with two threads, where capital letters denote shared memory locations, and small letters represent variables allocated in registers. Note that all values are natural numbers (Constraint 1), the locations are all initialized with zero (Constraint 2), and values written by stores are unique, greater than zero (Constraint 3), and obtained with $\theta = 10$ (Constraint 4).

Fig. 2(b) and Fig. 2(c) show distinct execution instances of that test program induced by different orders ($<$ and $\prec$). In each core, a trace captures the sequence of memory accesses resulting from the execution of load and store operations. Each memory access is represented as a triple specifying the access type as write (W) or read (R), the location, and the value. Fig. 2(d) shows the joint witness and its signature.

From now on, when we need to timestamp a witness, instead of $EW_{I,t}$, we simply write EW_t (assuming an implicit set I).

C. Recovering of state information from observations

Given the sequence $<A_0, O_1, A_1, O_2, A_2, O_3, \cdots, A_{t-1}, O_t>$, the recurrent pattern A_{t-1}, O_t is the basis for the approximation for the notion of state, where the interaction history is bounded.

Definition 9: For a bound $h \geq 1$, a state at time step $t \geq h$ is $S_t = <A_{t-h}, O_{t-h+1}, \cdots, A_{t-2}, O_{t-1}, A_{t-1}, O_t>$.

A proper value for the bound h can be obtained empirically.

Let $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the sets of states, actions, and observations. A transition from state S_t to S_{t+1} can be computed by shifting the representation of S_t one pair to the left, and appending A_t, O_{t+1}, so as to keep history bounded [19].

Definition 10: A *transition* is a function $\tau : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \mapsto \mathcal{S}$ such that $S_{t+1} = <A_{t-h+1}, O_{t-h+2}, \cdots, A_{t-1}, O_t, A_t, O_{t+1}>$.

We propose a state representation where $O_t = \sigma(EW_t)$.

V. Experimental evaluation

We built agent-based generators with the same set of actions. The *Random Agent Generator* (RAG) selects every action with the same probability. The *Deep-Q Generator* (DQG) relies on the DQN agent [24], and it has variants with distinct state representations, as shown at the top of Table I. RAG rules out learning effects (because its agent's policy is independent of state), serving as a baseline for evaluating the impact of state representation on learning, whether using general-purpose observations (coverage values in DQG$^+$) or the proposed domain-specific observations (execution witnesses in DQG*).

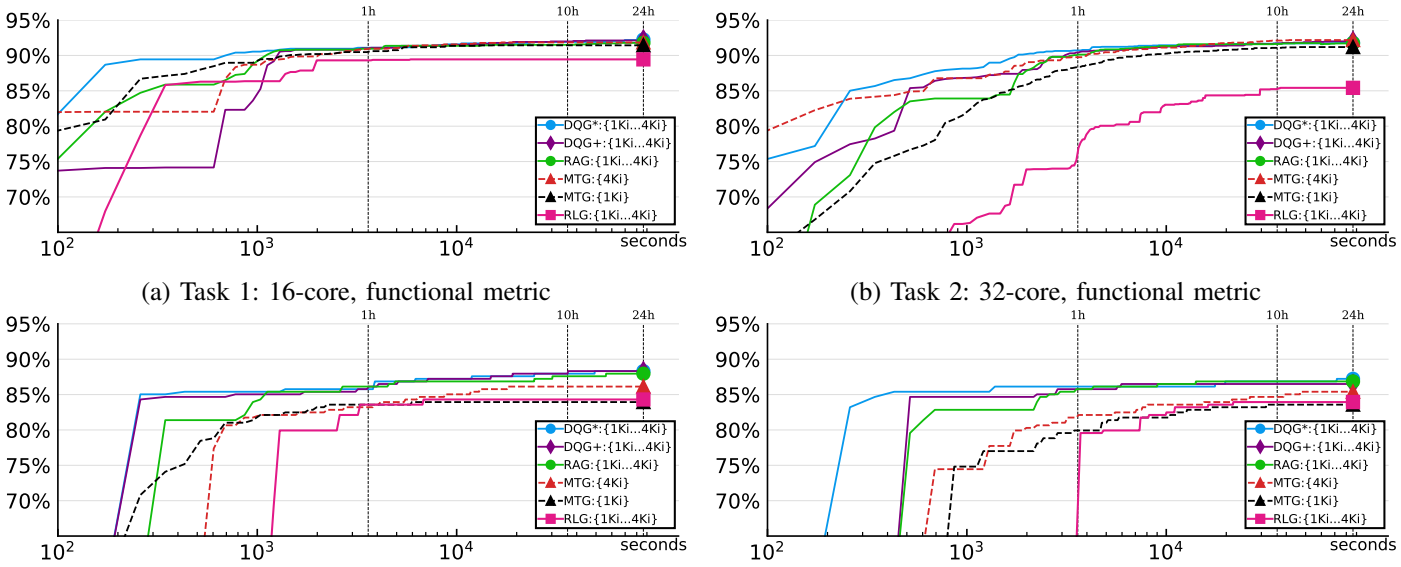
TABLE I: Generators used to evaluate our state representation

Generator	Agent	State Representation	
		Observation	Trajectory record
RAG	random	n.a.	n.a.
DQG$^+$	DQN [21]	C_t	action-observation history
DQG*		$\sigma(EW_t)$	action-observation history
RLG	[15]	C_t	t
MTG	none	n.a.	n.a.

We compared RAG and DQG variants with directed test generators described in related works: the Reinforcement Learning Generator (RLG) [15], whose state representation is not based on action-observation history, and the McVersi Test Generator (MTG) [13], which does not rely on an agent, but on evolutionary directed test generation, as shown at the bottom of Table I.

We evaluated each generator when performing different *verification tasks* targeting ARMv8 DUVs. Each task is obtained as a combination of distinct core counts (16 and 32) with different coverage metrics (structural transition coverage, as defined in [13], and functional transition coverage, as defined in [14]).

We relied on gem5's infrastructure [28] for simulation and design representation (we chose *O3*, *Ruby*, and *simple* for CPU, memory, and network models, respectively). For the DUVs, we



(a) Task 1: 16-core, functional metric

(b) Task 2: 32-core, functional metric

(c) Task 3: 16-core, structural metric

(d) Task 4: 32-core, structural metric

Fig. 3: Median (cumulative) coverage evolution for 24-hour verification episodes

adopted 64KiB (4-way) private L1 caches and a 4MiB (16-way) shared L2 cache, all with same block size (64 bytes), under a 2-level MOESI directory-based coherence protocol.

We ran experiments under the same range constraints on the number of operations ($n_{min} = 1Ki$, $n_{max} = 4Ki$) and shared locations ($s_{min} = 4$, $s_{max} = 128$). The allowed values within those ranges had to comply with generator's differences[1].

RAG, DQG, and RLG can generate programs with *variable* numbers of operations in the same test suite, but MTG does not: all programs must have a *fixed* number of operations. That is why we report individual MTG results for range extremes.

DQG employs a DQN agent, which uses a *Q-network* for non-linear approximation of the state-action value function. We adopted a feed-forward Q-network containing two hidden layers with 256 neurons each (ReLU and tanh were used for activation in successive layers). At each discrete time step t, the agent maximizes the expected discounted return $G_t = \sum_{t'=t+1}^{T} \gamma^{t'-(t+1)} R_{t'}$. The values chosen for the hyper-parameters were: discount rate $\gamma = 0.9$, learning rate $\eta = 0.025$, batch size $\beta = 32$, and the probability of taking a random action was linearly scaled down from $\epsilon = 0.9$ (when the episode starts) down to $\epsilon = 0.05$ (when it reaches one hour). We adopted $h = 4$ as history bound and the reward function defined by Equation 1.

$$R_{t+1} = \rho(C_{t+1}, C_t) = \begin{cases} +1 & \text{if } C_{t+1} > C_t \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

The number of execution instances of each test was set to five, and was induced as suggested in [13]. In each verification episode, every test was run five times under different simulation states (not related to the test itself) in such a way that the distinct executions of the same test were all perturbed differently.

[1]MTG requires the static definition of a *single* address-space constraint, which is called *test memory size* (TM). To accommodate such difference, we report MTG results under the (best) static constraint defined in [13], i.e. TM=8KiB. We preserved all genetic parameters as they were originally set.

Runtimes were measured on a Linux Server (Intel Xeon E5-2640 v4, 2.4 GHz, with 128GB of main memory).

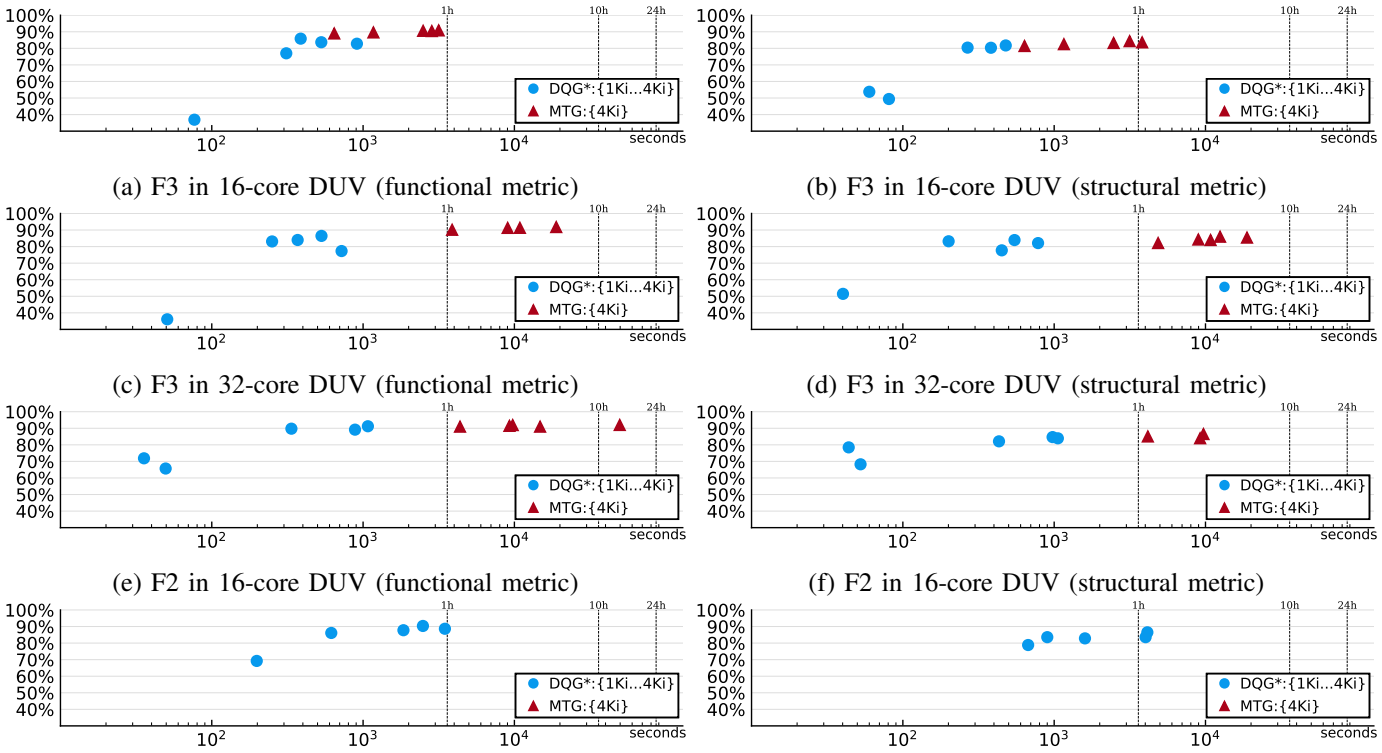
A. Coverage evolution with time

We performed independent 24-hour verification episodes for each task. We launched each generator five times with distinct random seeds, and reported median coverage values. For DQG and RLG, a new set of random weights was used to initialize their neural networks at each new launching.

Fig. 3 shows the coverage evolution for different tasks. RLG led to the slowest coverage evolution. The state representations used by RLG and DQG$^+$ both rely on coverage values as generic observations. However, RLG does not take the history of action-observation pairs into account to approximate the environment state as DQG$^+$ does. This explains why RLG exhibits limited coverage evolution as compared to DQG$^+$. Note that DQG* was superior to RAG for all tasks, while DQG+ was mostly superior to RAG, except for Task 1. This indicates that the history of action-observation pairs *did* enable learning, which is more prominent when our state representation is used.

DQG* was superior to DQG$^+$, despite environment changes from one task to another. Thus, DQG* observations capture relevant information about the environment *independently of task*, i.e. they are properly tailored to the target *domain*.

As compared to related works, DQG* had the fastest coverage evolution for all tasks with exactly the same agent. Under the proposed state representation, the agent was able to learn on-line how to handle different tasks from "raw" observations of the environment, i.e. observations that were *not* represented in terms of specialized features handcrafted for any given task. Thus, a learning agent can make a directed test generator *effective* and *reusable* for distinct tasks of the same domain.



(a) F3 in 16-core DUV (functional metric)

(b) F3 in 16-core DUV (structural metric)

(c) F3 in 32-core DUV (functional metric)

(d) F3 in 32-core DUV (structural metric)

(e) F2 in 16-core DUV (functional metric)

(f) F2 in 16-core DUV (structural metric)

(g) F2 in 32-core DUV (functional metric)

(h) F2 in 32-core DUV (structural metric)

Fig. 4: Coverage-time error discovery constellations for faulty designs

B. Impact on error discovery

We evaluated the impact of our state representation on the runtime required to expose errors in faulty designs. For the experiments, we selected challenging errors described in the literature, namely F2 and F3 [9]. We modified our DUVs to obtain faulty designs, each containing a single error. Since coverage results have shown that DQG* dominates DQG$^+$ and MTG dominates RLG, we focused on comparing the best DQG version (DQG*) with the best-performing related work (MTG).

TABLE II: Median error discovery times (in seconds)

Error	Cores	Functional metric		Structural metric	
		DQG*	MTG	**DQG***	MTG
F3	16	**386**	2493	**267**	2470
	32	**369**	9948	**450**	10802
F2	16	**335**	9744	**431**	9216
	32	**1848**	-	**1597**	-

We launched five verification episodes with distinct random seeds. We measured the runtime until the error was discovered and the coverage at the time of discovery. Then we plotted a constellation of coverage-time points for each generator. Fig. 4 shows constellation plots for faulty designs with a single error under a given coverage metric. DQG* exposed every error in all 24-hour episodes, but MTG not always did. For a 32-core DUV, MTG did *not* expose F2 in *any* episode. When MTG exposed an error, it took much longer, as shown in Table II.

DQG* led to the fastest error diagnosis in all scenarios, it exposed errors with lower coverage (on average), and its discovery times were less sensitive to the increase in core count. This indicates that the proposed state representation leads to high-quality tests, quite independently of coverage metric.

Higher-quality tests can be obtained when an agent directs generation instead of an evolutionary algorithm, mainly because the former is farsighted, while the latter is myopic. The former takes decisions based on expected returns over the long run. The latter takes them based on immediate returns (i.e. the best for the current population, regardless of next generations).

VI. Conclusions

This paper proposed a novel reinforcement learning approach to directed test generation for a crucial step of the design of a multicore system: the verification of the interaction between cores through shared memory. To be effective, that approach should rely on a proper representation for the state of a multicore enviroment, which requires the tracking of the history of action-observation pairs, and proper tailoring of observations to the target domain. This allows the agent-based generator to learn online how to be effective and reusable for *different* tasks of the *same* domain. Results have shown that, when encoded as unique signatures, observations capturing the *reads-from* relation led to faster coverage evolution and faster error diagnosis, as compared to related work. This makes our approach to directed test generation a promising alternative to conventional techniques that are not based on agents.

REFERENCES

[1] S. V. Adve and K. Gharachorloo, "Shared memory consistency models: a tutorial," *Computer*, vol. 29, no. 12, pp. 66–76, 12 1996.

[2] V. Nagarajan, D. Sorin, M. Hill, and D. Wood, *A Primer on Memory Consistency and Cache Coherence, 2nd Edition*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers, Feb. 2020.

[3] A. Adir and G. Shurek, "Generating concurrent test-programs with collisions for multi-processor verification," in *7th IEEE Int. High-Level Design Validation and Test Workshop*, 2002, pp. 77–82.

[4] A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, and A. Ziv, "Genesys-Pro: innovations in test program generation for functional processor verification," *IEEE Design Test of Computers*, vol. 21, no. 2, pp. 84–93, Mar 2004.

[5] A. Saha, N. Malik, B. O'Krafka, J. Lin, R. Raghavan, and U. Shamsi, "A simulation-based approach to architectural verification of multiprocessor systems," in *Proceedings International Phoenix Conference on Computers and Communications*, 1995, pp. 34–37.

[6] C. Manovit and S. Hangal, "Completely verifying memory consistency of test program executions," in *IEEE Int. Symposium on High-Performance Computer Architecture (HPCA)*, 2006, pp. 166–175.

[7] A. Roy, S. Zeisset, C. J. Fleckenstein, and J. C. Huang, *Fast and Generalized Polynomial Time Memory Consistency Verification*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, vol. 4144, pp. 503–516.

[8] O. Shacham, M. Wachs, A. Solomatnikov, A. Firoozshahian, S. Richardson, and M. Horowitz, "Verification of chip multiprocessor memory systems using a relaxed scoreboard," in *IEEE/ACM International Symposium on Microarchitecture (MICRO 41)*, 2008, pp. 294–305.

[9] M. Graf, G. A. G. Andrade, and L. C. V. dos Santos, "Evecheck: An event-driven, scalable algorithm for coherent shared memory verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 2, pp. 683–696, 2023.

[10] S. Fine and A. Ziv, "Coverage Directed Test Generation for Functional Verification Using Bayesian Networks," in *ACM Design Automation Conference (DAC)*, 2003, pp. 286–291.

[11] X. Qin and P. Mishra, "Automated generation of directed tests for transition coverage in cache coherence protocols," in *Design, Automation, and Test in Europe (DATE)*, 2012, pp. 3–8.

[12] Y. Lyu, X. Qin, M. M. Chen, and P. Mishra, "Directed test generation for validation of cache coherence protocols," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 1, pp. 163–176, Jan 2019.

[13] M. Elver and V. Nagarajan, "McVerSi: A test generation framework for fast memory consistency verification in simulation," in *IEEE Int. Symp. on High Perform. Computer Architecture (HPCA)*, 2016, pp. 618–630.

[14] G. A. G. Andrade, M. Graf, N. Pfeifer, and L. C. V. dos Santos, "A directed test generator for shared-memory verification of multicore chip designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 5295–5303, 2020.

[15] N. Pfeifer, B. V. Zimpel, G. A. G. Andrade, and L. C. V. dos Santos, "A Reinforcement Learning Approach to Directed Test Generation for Shared Memory Verification," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 3 2020, pp. 538–543.

[16] J. Alglave, L. Maranget, S. Sarkar, and P. Sewell, "Fences in weak memory models," in *Computer Aided Verification*, vol. 6174, Dec 2010, pp. 258–272.

[17] D. Lustig, A. Wright, A. Papakonstantinou, and O. Giroux, "Automated Synthesis of Comprehensive Memory Model Litmus Test Suites," in *ACM Int. Conf. on Arch. Support for Progr. Lang. and Op. Systems (ASPLOS)*, 2017, pp. 661–675.

[18] I. Wagner and V. Bertacco, "MCjammer: Adaptive Verification for Multi-core Designs," in *Design, Automation, and Test in Europe (DATE)*, 2008, pp. 670–675.

[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[20] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, p. 58–68, Mar. 1995. [Online]. Available: https://doi.org/10.1145/203330.203343

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *NIPS Deep Learning Workshop 2013*, 2013.

[22] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3–4, pp. 279–292, May 1992. [Online]. Available: http://www.springerlink.com/content/p120815501618373/fulltext.pdf

[23] L.-J. Lin, "Reinforcement learning for robots using neural networks," Ph.D. dissertation, Computer Science, USA, 1992, uMI Order No. GAX93-22750.

[24] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[25] T. Ball and J. Larus, "Efficient path profiling," in *Proceedings of the 29th Annual IEEE/ACM International Symposium on Microarchitecture. MICRO 29*, 1996, pp. 46–57.

[26] D. Lee and V. Bertacco, "Mtracecheck: Validating non-deterministic behavior of memory consistency models in post-silicon validation," in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 201–213.

[27] K. Gharachorloo, "Memory consistency models for shared-memory multiprocessors," Ph.D. dissertation, Stanford University, 1995.

[28] N. Binkert et al., "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, Aug 2011.

ChipAlign: Instruction Alignment in Large Language Models for Chip Design via Geodesic Interpolation

Chenhui Deng
NVIDIA
cdeng@nvidia.com

Yunsheng Bai
NVIDIA
yunshengb@nvidia.com

Haoxing Ren
NVIDIA
haoxingr@nvidia.com

***Abstract*—Recent advancements in large language models (LLMs) have expanded their application across various domains, including chip design, where domain-adapted chip models like ChipNeMo have emerged. However, these models often struggle with instruction alignment, a crucial capability for LLMs that involves following explicit human directives. This limitation impedes the practical application of chip LLMs, including serving as assistant chatbots for hardware design engineers. In this work, we introduce ChipAlign, a novel approach that utilizes a *training-free* model merging strategy, combining the strengths of a general instruction-aligned LLM with a chip-specific LLM. By considering the underlying manifold in the weight space, ChipAlign employs geodesic interpolation to effectively fuse the weights of input LLMs, producing a merged model that inherits strong instruction alignment and chip expertise from the respective instruction and chip LLMs. Our results demonstrate that ChipAlign significantly enhances instruction-following capabilities of existing chip LLMs, achieving up to a 26.6% improvement on the IFEval benchmark, while maintaining comparable expertise in the chip domain. This improvement in instruction alignment also translates to notable gains in instruction-involved QA tasks, delivering performance enhancements of 3.9% on the OpenROAD QA benchmark and 8.25% on production-level chip QA benchmarks, surpassing state-of-the-art baselines.**

I. Introduction

Recent years have seen remarkable breakthroughs in large language models (LLMs) across a wide range of applications, including text summarization, machine translation, and conversational AI [30]. Models like GPT [1], Gemini [19], Claude [4], and LLaMA [8] series have transformed numerous industries by automating complex tasks, enhancing decision-making processes, and enabling creative problem-solving that traditionally required human expertise. Alongside this success, there is a growing trend of adapting LLMs to specific domains to meet specialized needs. Domain-adapted LLMs have been developed for fields such as healthcare [22], finance [24], law [5], and climate [20], where nuanced understanding and specialized knowledge are essential for enhancing model performance within these domains.

In the realm of chip design, ChipNeMo stands out as a prominent example of domain-adapted LLMs [14]. Built on the LLaMA2-70B foundation model, ChipNeMo leverages domain-adaptive pretraining (DAPT) and finetuning (DAFT) to imbue the model with specialized knowledge in circuits, bugs, and electronic design automation (EDA) scripts. Following ChipNeMo, several customized LLMs have recently been developed for EDA tasks. He et al. introduced AutoMage, an LLM finetuned on LLaMA2 to specialize in EDA tool utilization [23]. Subsequently, Sharma et al. [17] and Pu et al [16]. developed LLMs tailored for QA and script generation tasks from OpenROAD.

However, these tailored LLMs typically exhibit diminished instruction alignment, a fundamental capability of general-purpose chat LLMs to follow human instructions, as demonstrated in Figure 2. This decline in instruction alignment limits the practical usability of chip LLMs, as they may struggle to respond effectively to user-directed commands, making them less versatile and reliable in real-world applications. For instance, when serving as single or multi-turn chatbots for hardware design engineers, it is critical that the chip

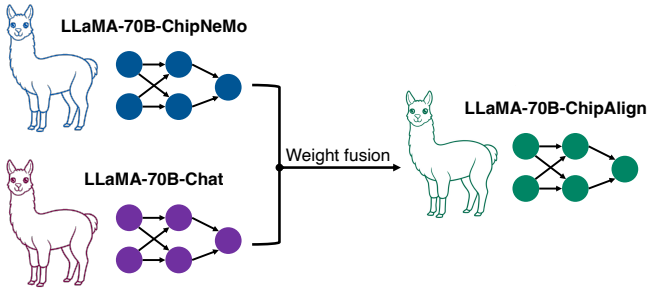


Figure 1: An illustration of model merging.

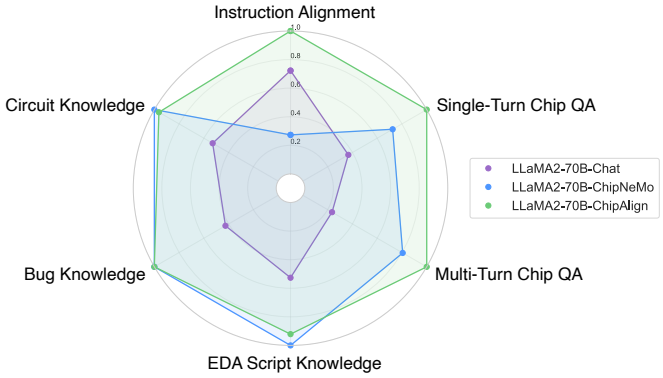


Figure 2: An overview of the capabilities of LLaMA2-70B variants: Chat, ChipNeMo, and ChipAlign on instruction alignment and chip domain benchmarks — Scores are normalized to $[0, 1]$ as per [12], with points closer to the outermost circle indicating better performance.

LLMs not only possess deep hardware knowledge but also strictly adhere to engineer instructions, such as "Please answer questions exclusively based on the provided context" or "Please provide a detailed and rigorous solution". Unfortunately, prior chip LLMs often compromise this ability after going through DAPT or DAFT, leading to less satisfactory responses as indicated in Figures 5 and 6.

To overcome the aforementioned challenge, a viable solution is multi-task learning, training an LLM on both chip domain-specific and instruction-following data. This approach equips the model with dual capabilities, combining chip design expertise with a strong instruction-following capability. Nonetheless, a significant obstacle to multi-task learning is the proprietary nature of high-quality instruction datasets, such as those used even for open-sourced models like the LLaMA series [8], rendering it difficult to obtain a well-aligned model.

In this work, we address the instruction alignment challenge from an orthogonal perspective. Instead of training on instruction data, we leverage a *training-free* approach based on model merging, which fuses the weights of LLMs specialized in different tasks to produce a unified model that excels across them, as shown in Figures 1. To

this end, we introduce ChipAlign, which enhances the instruction alignment of a chip LLM by merging its weights with those of a well-aligned general instruction model that is publicly available (e.g., LLaMA2-Chat or LLaMA3-Instruct). More concretely, we treat the weights of both instruction and chip LLMs as two points on a Riemannian manifold, a curved geometric space that allows for measuring distances and angles [3]. To construct a merged model that embodies both strong instruction alignment and chip design expertise, it should ideally reside near both input models on this manifold. This requirement implies positioning the model along the geodesic—the shortest path between the two points on the manifold. Accordingly, ChipAlign employs geodesic interpolation to generate a new model that effectively combines the strengths of both input models.

To demonstrate the viability and flexibility of ChipAlign, we conduct extensive experiments across several benchmarks, including the instruction-following evaluation (IFEval) benchmark [32] and various chip domain benchmarks such as the open-source OpenROAD QA benchmark [16] and proprietary production-level chip QA benchmarks. We utilize LLMs ranging from 8B to 70B parameters. Our results indicate that ChipAlign significantly enhances the instruction-following capabilities of chip LLMs, thereby improving their performance on practical chip QA benchmarks that involve specific instructions. We summarize our main technical contributions as follows:

• To our knowledge, we are the first to apply a model merging approach to domain-adapted LLMs in chip design. This is achieved by fusing the weights of a chip LLM with those of an instruction-aligned LLM, without requiring additional training.

• By considering the geometric properties of the weight space in LLMs, ChipAlign utilizes geodesic interpolation to produce a merged model. This facilitates a smoother fusion of input LLM weights, leading to consistent enhancements in performance over previous model merging methods across different LLM backbones.

• Owing to the geometric-aware merging scheme in ChipAlign, the merged model inherits both strong instruction alignment and chip expertise from the input LLMs. Compared to the state-of-the-art baseline ChipNeMo, ChipAlign achieves a significant enhancement in instruction alignment, showing a 26.6% improvement on the IFEval benchmark while maintaining comparable domain knowledge.

• Thanks to the enhanced instruction alignment, ChipAlign outperforms state-of-the-art chip LLMs in instruction-involved QA tasks, achieving improvements of 3.9% on the OpenROAD QA benchmark and 8.25% on production-level chip QA benchmarks.

II. Background

A. Domain-Adapted LLMs for Chip Design

In chip design, the need for domain-adapted LLMs has driven the development of several models tailored to hardware-related tasks. Liu et al. developed ChipNeMo starting with DAPT on 24 billion tokens drawn from chip design documents and code, using the LLaMA2-70B foundation model as a base. This pretraining phase employs the standard autoregressive language modeling objective to tailor the model to domain-specific data. Subsequently, the model underwent DAFT on approximately 57,000 samples, incorporating both domain-specific instructional data and open-source chat data from OASST [11]. Through DAPT and DAFT, ChipNeMo acquired specialized knowledge in the chip domain, leading to promising outcomes in various hardware design applications [14]. Later, He et al. developed AutoMage, an LLM finetuned on LLaMA2 for EDA tool usage [23], which led to the creation of ChatEDA, an autonomous agent customized for EDA design flow. More recently, Sharma et al. [17] and Pu et al. [16] have tailored LLMs for OpenROAD script generation and QA tasks, covering a broad spectrum of queries related to command usage, VLSI flow, installation guides, and GUI usage.

B. Instruction Alignment in Chip Design LLMs

DAPT and DAFT often drastically change the weights of LLMs to emphasize domain knowledge, resulting in a loss of the instruction-following capabilities originally present in general-purpose LLMs [9]. However, instruction alignment is crucial for real-world applications, such as a chatbot assistant for chip designers. In such settings, designers may seek guidance on design methodologies, troubleshooting steps, or explanations of specific design concepts, often phrased as direct instructions. Additionally, they may instruct the chatbot to respond solely based on a given context, which ensures the answer is grounded in relevant and context-specific information, as shown in Figures 5 and 6. Hence, the ability to understand and respond appropriately to these instructions is vital for the practical usability of a chip LLM, making instruction alignment an essential feature.

A straightforward approach to enhance the instruction alignment of chip LLMs involves multi-task learning, which simultaneously trains a model on chip domain-specific data and instruction-following data to effectively integrate both sets of capabilities. However, access to high-quality instruction data is limited, as datasets used by advanced models like GPT-4 and the LLaMA series remain proprietary. While open-source instruction datasets are valuable [11], they often lack the scale and diversity needed to train models effectively for complex instruction-following tasks. Besides, even when data is available, the costs associated with finetuning on large-scale instruction datasets are prohibitively high, particularly for models with billions of parameters.

C. Instruction Alignment via Model Merging

In contrast to multi-task learning, model merging is a training-free technique that directly fuses the weights of specialized LLMs to incorporate multiple capabilities without requiring access to the original training data [27]. Recent work has demonstrated that with a properly designed weight fusion scheme, model merging can achieve performance comparable to multi-task learning, making it an efficient method to equip LLMs with multiple capabilities [28].

In literature, Model Soup represents a pioneering method in this direction, averaging the weights of different LLMs to create a single model that generalizes well across tasks [21]. Following this idea, Ilharco et al. proposed task arithmetic that averages the weight differences (i.e., task vectors) between input LLMs and their common base model; the resulting average is then added back to the base model to produce the merged model [10]. Building on task arithmetic, TIES [26] enhances the approach by sparsifying the task vectors and incorporating a sign consensus algorithm prior to weight fusion. DELLA [6] further advances TIES via adaptively pruning less important weights with specific hyperparameters. However, all of these methods neglect the underlying geometric properties of LLM weights, which can lead to merged models with suboptimal performance. In contrast, ChipAlign aims to merge a chip LLM with a well-instructed general LLM through geodesic interpolation. This geometric-aware technique allows us to smoothly blend the weights of the original LLMs along the shortest path on a Riemannian manifold. As a result, ChipAlign produces a merged model that effectively combines chip domain knowledge with instruction alignment, outperforming previous model merging techniques as detailed in Table 1. Notably, while ChipAlign has potential applications in other domains, this work primarily focuses on hardware-related QA tasks.

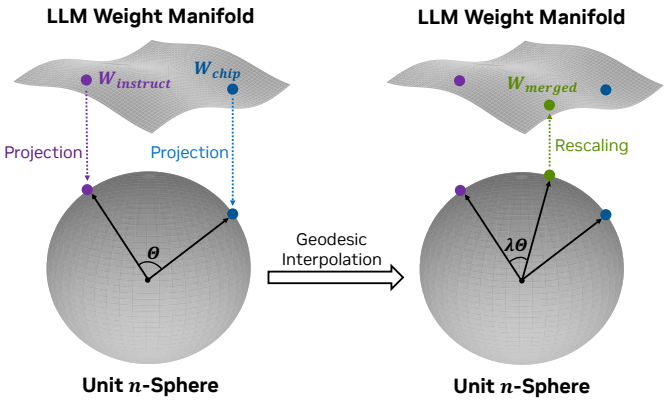


Figure 3: An overview of ChipAlign.

III. The Proposed Approach

Problem formulation – Let M_{chip} denote a chip LLM and M_{instruct} denote a general-purpose instruction LLM. Let $\{W^{(l)} | \, l = 1, ..., L\}$ denote the complete set of weights for an L-layer LLM, encompassing weights from the embedding layer, normalization layers, self-attention layers, and feed-forward layers. For each layer l, let $W^{(l)}_{\text{chip}} \in \mathbb{R}^{p \times q}$ and $W^{(l)}_{\text{instruct}} \in \mathbb{R}^{p \times q}$ represent the weight matrices of M_{chip} and M_{instruct}, respectively. Our goal is to develop a merging function f such that:

$$W^{(l)}_{\text{merge}} = f(\, W^{(l)}_{\text{chip}}, \, W^{(l)}_{\text{instruct}} \,)$$

The resulting $W^{(l)}_{\text{merge}} \in \mathbb{R}^{p \times q}$ serves as the weights for the l-th layer in the merged model M_{merge}, which aims to combine the strengths of both M_{chip} and M_{instruct}. For brevity, we omit the layer index l in the following sections unless explicitly mentioned. Notably, our problem formulation implicitly assumes that the input models share the same architecture, meaning their respective weight matrices W_{chip} and W_{instruct} are conformable for merging. This assumption generally holds in practice; for example, ChipNeMo is trained based on LLaMA2-70B-Base, which has the same architecture as the instruction model LLaMA2-70B-Chat that is publicly available.

Figure 3 provides an overview of our proposed approach, ChipAlign. Next, we are going to first present the motivation for using geodesic interpolation in our merging method in Section III-A. Then, Section III-B introduces ChipAlign that computes geodesic interpolation along a unit n-sphere. Finally, we analyze the complexity of ChipAlign in Section III-C. It is worth noting that ChipAlign operates under the assumption that both chip and instruction LLMs are already available. Details on how to obtain these LLMs are provided in Section IV-A.

A. Model Merging via Geodesic Interpolation

Neural network weights can be viewed as points on a high-dimensional Riemannian manifold [2], [3]. This perspective opens the door for leveraging powerful geometric tools, such as geodesic interpolation, to transition smoothly between two sets of model weights. Geodesic interpolation follows the shortest path on the manifold between two points, providing a structured way to transition through the weight space without compromising on important model properties. Such interpolation techniques can be particularly useful for merging LLMs that specialize in different tasks. Given two points on the manifold corresponding to LLM weights optimized for distinct objectives, geodesic interpolation enables us to find a new point on the manifold that lies near both models, thereby inheriting strengths from each. This approach allows us to combine properties such as chip domain knowledge and instruction alignment into a single LLM.

However, a significant challenge arises due to the computational intractability of performing geodesic interpolation in high-dimensional manifolds. To overcome this, we project the model weights onto a specific type of Riemannian manifold, the unit n-sphere, where geodesic interpolation can be performed in linear time using a canonical form, as discussed in Section III-B. This method provides an efficient way to merge model capabilities while respecting the geometric properties of the high-dimensional weight space.

B. Geodesic Interpolation on Weight Manifold

Our approach ChipAlign adopts geodesic interpolation between the weights of a chip LLM and an instruction-aligned LLM. To this end, we first formally define the unit n-sphere in the following.

Definition III.1. The *unit n-sphere* S^n is the set of all points in $(n+1)$-dimensional Euclidean space $\mathbb{R}^{n+1}$ that are at a unit distance from the origin:

$$S^n = \{w \in \mathbb{R}^{n+1} : \|w\| = 1\}$$

where $\| \cdot \|$ denotes the Euclidean norm.

To efficiently perform geodesic interpolation between the weights of these models, ChipAlign first projects the weight matrices W_{chip} and W_{instruct} onto a unit n-sphere as follows:

$$\text{Norm}_{\text{chip}} = \|W_{\text{chip}}\|_F, \quad \text{Norm}_{\text{instruct}} = \|W_{\text{instruct}}\|_F$$

$$\bar{W}_{\text{chip}} = \frac{W_{\text{chip}}}{\text{Norm}_{\text{chip}}}, \quad \bar{W}_{\text{instruct}} = \frac{W_{\text{instruct}}}{\text{Norm}_{\text{instruct}}}$$

where $\| \cdot \|_F$ denotes the Frobenius norm. According to Definition III.1, $\bar{W}_{\text{chip}} \in \mathbb{R}^{p \times q}$ and $\bar{W}_{\text{instruct}} \in \mathbb{R}^{p \times q}$ now reside on a unit n-sphere ($n = p \times q - 1$), the geodesic between them is represented by the arc connecting the two points on the sphere. This allows us to perform geodesic interpolation by interpolating along this arc, which can be efficiently achieved using the following lemma:

Lemma III.2. *Given $\bar{W}_{chip}$ and $\bar{W}_{instruct}$ lie on a unit n-sphere, the geodesic interpolation between them can be expressed as:*

$$\bar{W}_{merge} = \frac{sin(\lambda\Theta)}{sin(\Theta)} \bar{W}_{chip} + \frac{sin((1-\lambda)\Theta)}{sin(\Theta)} \bar{W}_{instruct}$$

where $\Theta = arccos(\bar{W}_{chip}, \bar{W}_{instruct})$ is the angle between $\bar{W}_{chip}$ and $\bar{W}_{instruct}$, and $\lambda \in [0, 1]$ determines the interpolation point along the geodesic, with $\lambda = 0$ corresponding to $\bar{W}_{instruct}$ and $\lambda = 1$ corresponding to $\bar{W}_{chip}$.

The proof for Lemma III.2 is available in [18]. Lemma III.2 allows us to generate a continuum of models between the two input models along the unit n-sphere, with λ determining the degree to which each model's weights are retained in the merged model.

Finally, to restore the magnitude of the original weight matrices, we rescale the interpolated weights back to the manifold by applying the Frobenius norms:

$$W_{\text{merge}} = \text{Norm}^{\lambda}_{\text{chip}} \cdot \text{Norm}^{1-\lambda}_{\text{instruct}} \cdot \bar{W}_{\text{merge}}$$

This process results in W_{merge}, the weight matrix for each layer in the merged model M_{merge}, effectively combining instruction alignment and chip design expertise in a single model.

Discussion – It is worth noting that an important advantage of ChipAlign is its ease of implementation, incorporating only a single hyperparameter λ. Our sensitivity analysis in Section IV-E reveals that $\lambda = 0.6$ leads to peak performance across different LLM backbones. This minimizes the burden of hyperparameter tuning,

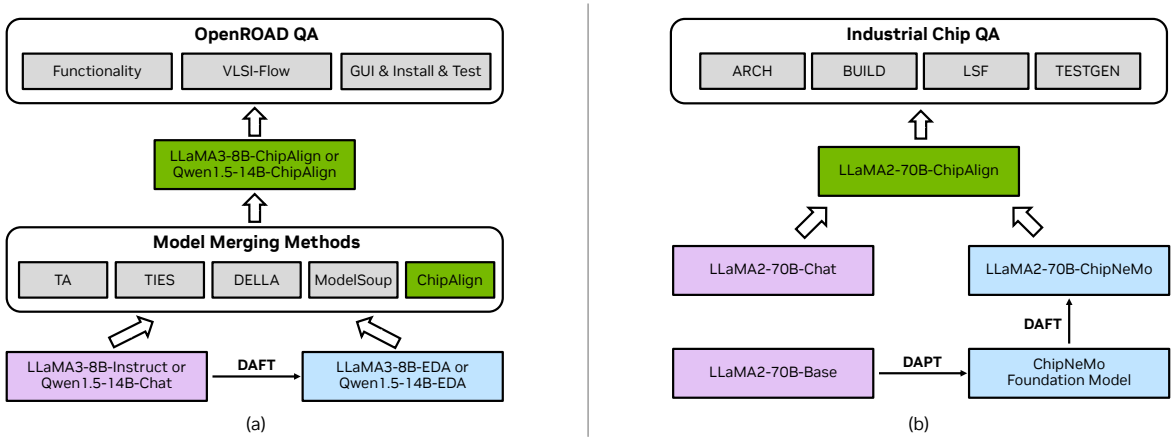


Figure 4: (a) An overview of developing merged models for the OpenROAD QA benchmark, featuring ChipAlign as the selected merging approach; (b) An overview of generating LLaMA2-70B-ChipAlign for industrial chip QA benchmarks.

making ChipAlign particularly easy to adopt for large-scale domain-adapted models. We further analyze the complexity of ChipAlign in Section III-C to highlight its efficiency advantages.

C. Complexity Analysis of ChipAlign

ChipAlign projects the weight matrices of input LLMs, each with n parameters, onto a unit n-sphere, requiring a single pass through the model weights in $O(n)$ time. It then performs geodesic interpolation along the arc between these weights and rescales the interpolated weights in $O(n)$ time. Consequently, the total time complexity for ChipAlign is $O(n)$. Similarly, its space complexity is $O(n)$, covering the storage of initial, projected, and final weights. This efficiency enables ChipAlign to handle billion-parameter scale LLMs with minimal computational overhead.

IV. Experiment

We have conducted a comprehensive evaluation of ChipAlign, comparing its performance against state-of-the-art (SoTA) baselines on the OpenROAD QA benchmark and industrial production-level chip QA benchmarks. To further evaluate ChipAlign's capabilities, we assess its instruction alignment on the IFEval benchmark and its domain knowledge on multiple-choice QA benchmarks covering EDA script generation, bug summarization, and circuit design. Finally, we conduct a sensitivity analysis on the hyperparameter λ in ChipAlign.

A. Experimental Setup

OpenROAD QA – As shown in Figure 4(a), we consider two well-aligned instruction LLMs that are publicly available: Qwen1.5-14B-Chat and LLaMA3-8B-Instruct. Following the methodology of Pu et al. [16] and Sharma et al. [17] to generate strong domain-adapted LLMs, we apply retrieval augmented DAFT to both models using around 2K context-query-answer training triplets from OpenROAD QA [29]. Specifically, we perform DAFT on each training QA pair along with its golden context, and adopt low-rank adaptation (LoRA) with a rank of 8 and an alpha of 16. We train the models over 20 epochs with a learning rate of 2×10^{-4} and a batch size of 1. Both models are trained on four nodes of a computing cluster, each node being equipped with eight A100 GPUs, each with 80GB of memory. This process results in the creation of Qwen1.5-14B-EDA and LLaMA3-8B-EDA. Once both the instruction and domain-adapted LLMs are available, we use ChipAlign with $\lambda = 0.6$ to fuse their weights, producing Qwen1.5-14B-ChipAlign and LLaMA3-8B-ChipAlign. Remarkably, the fusion process takes only 10 minutes on a CPU with 48 cores running at 2.5 GHz. Additionally, we compare ChipAlign with various popular model merging baselines such as task arithmetic (TA), TIES, DELLA, and Model Soup, adopting the recommended hyperparameters from their respective publications. Besides, we further compare ChipAlign against the general SoTA model GPT-4 Turbo and RAG-EDA, a Qwen1.5-14B-Chat based LLM highly customized for this benchmark [16]. To quantitatively assess model performance, we follow Pu et al. to adopt the ROUGE-L scoring method for comparing the generated LLM responses with the golden answers [13]. We have found that the ROUGE-L score is a more representative metric on this benchmark than either the BLEU [15] or UniEval [31] scores.

Industrial chip QA – As indicated by Figure 4(b), we select what is possibly the largest chip LLM, LLaMA2-70B-ChipNeMo, as our baseline. This model is developed by first undergoing DAPT on 24 billion tokens derived from chip design documents and code, using the LLaMA2-70B-Base model, followed by DAFT on 57K instruction pairs. More training details are available in [14]. Additionally, we choose LLaMA2-70B-Chat as the instruction model that is publicly available and has the same architecture as LLaMA2-70B-ChipNeMo. We set $\lambda = 0.6$ in ChipAlign for merging LLaMA2-70B-ChipNeMo and LLaMA2-70B-Chat to produce LLaMA2-70B-ChipAlign. Owing to the linear complexity of ChipAlign, the weight fusion process takes 43 minutes on a CPU equipped with 48 cores operating at 2.5 GHz. Besides, we utilize a GPT-4 based grader to assess the quality of each LLM response by comparing it with the golden answer. The grader assigns scores in $\{0, 25, 50, 75, 100\}$, where a higher score indicates better answer quality.

B. Evaluation on OpenROAD QA

We evaluate all models using 90 high-quality context-query-answer triplets from the OpenROAD QA benchmark, which all follow the same instruction shown in Figure 5. For each triplet, the context is derived either from the golden context corresponding to the given query or from the retrieved context via retrieval-augmented generation (RAG) using OpenROAD documentation. To establish a strong RAG pipeline, we employ the *bge-large-en-v1.5* for text embedding, *BM25* for lexical retrieval, and *bge-reranker-large* for re-ranking [25]. For the purpose of reproducibility, we set the temperature parameter to 0.0 for all models during response generation.

Table 1 showcases that ChipAlign surpasses all existing model merging methods in most cases, achieving a ROUGE-L score improvement of up to 6.4%. This empirical evidence confirms that our geometric-aware method produces a superior merged model compared to other

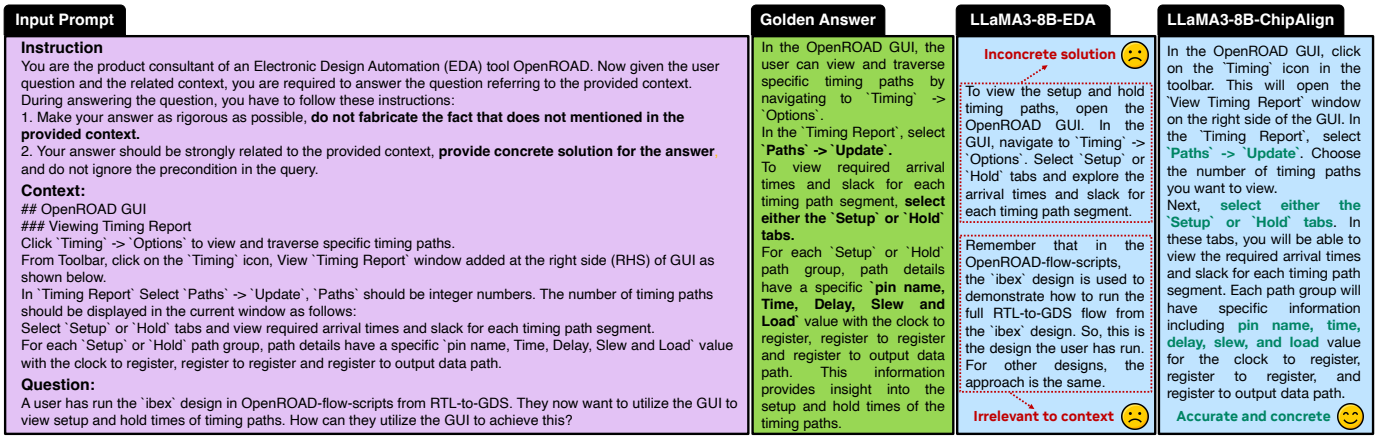


Figure 5: Comparison of model responses on the OpenROAD QA benchmark.

Table 1: ROUGE-L scores on the OpenROAD QA benchmark — * denotes the results sourced from [16].

	Golden Context				RAG Context			
Method	Functionality	VLSI Flow	GUI & Install & Test	All	Functionality	VLSI Flow	GUI & Install & Test	All
GPT-4 Turbo	0.280	0.338	0.367	0.315	0.250	0.274	0.331	0.276
RAG-EDA*	0.319	0.326	0.374	0.334	0.281	0.269	0.302	0.283
Qwen1.5-14B-Chat	0.276	0.318	0.354	0.305	0.244	0.264	0.301	0.263
Qwen1.5-14B-EDA	0.315	0.337	0.385	0.338	0.282	0.270	0.335	0.292
Qwen1.5-14B-TA	0.316	0.346	0.392	0.342	0.285	0.288	**0.358**	0.303
Qwen1.5-14B-TIES	0.300	0.352	0.368	0.329	0.255	0.279	0.338	0.281
Qwen1.5-14B-DELLA	0.295	0.343	0.382	0.328	0.254	0.278	0.304	0.272
Qwen1.5-14B-ModelSoup	0.317	0.359	0.389	0.345	0.295	0.278	0.358	0.306
Qwen1.5-14B-ChipAlign	**0.354**	**0.366**	**0.403**	**0.369**	**0.305**	**0.294**	0.354	**0.314**
LLaMA3-8B-Instruct	0.277	0.307	0.372	0.308	0.258	0.232	0.345	0.273
LLaMA3-8B-EDA	0.322	0.360	0.367	0.342	0.283	0.265	0.345	0.294
LLaMA3-8B-TA	0.284	0.362	0.372	0.325	0.267	0.292	0.344	0.292
LLaMA3-8B-TIES	0.286	0.332	0.376	0.319	0.256	0.281	0.347	0.285
LLaMA3-8B-DELLA	0.289	0.340	0.365	0.320	0.263	0.275	0.353	0.288
LLaMA3-8B-ModelSoup	0.333	0.370	**0.429**	0.365	0.288	0.286	0.368	0.307
LLaMA3-8B-ChipAlign	**0.362**	**0.385**	0.427	**0.383**	**0.304**	**0.300**	**0.392**	**0.325**

baselines. Moreover, the merged models generated by ChipAlign consistently outperform their corresponding EDA models across different LLM backbones, achieving performance improvements of up to 3.9%—a nontrivial margin for this benchmark. This improvement is attributed to ChipAlign's efficacy in enhancing instruction alignment of domain-adapted models, resulting in responses that better align with the provided instructions, as evidenced in Figure 5 and Table 3.

C. Evaluation on Industrial Chip QA

We have extended our evaluation of ChipAlign to include industrial production-level chip QA benchmarks, consisting of 39 practical questions from hardware design engineers across domains such as hardware architecture (ARCH), build processes (BUILD), job scheduling (LSF), and verification (TESTGEN). In addition to a single-turn setting where each question is treated independently, we also explore a multi-turn setting to simulate real-world scenarios where engineers may pose follow-up questions based on previous interactions. As illustrated in Figure 6, each input prompt includes a question, relevant contexts obtained through RAG, and multiple instructions, necessitating strong instruction alignment in LLMs. We set the temperature parameter to 0.0 for all model responses.

Table 2 showcases that the merged model LLaMA2-70B-ChipAlign consistently outperforms both source models (LLaMA2-70B-Chat and LLaMA2-70B-ChipNeMo) by a margin of up to 8.25%. Besides, Figure 6 illustrates that both LLaMA2-70B-Chat and LLaMA2-70B-ChipNeMo fail to follow the instructions, which mandate a model to answer questions using information from the relevant context. In contrast, LLaMA2-70B-ChipAlign strictly adheres to the provided instructions and generate correct answers supported by the context. This confirms ChipAlign's efficacy with the largest existing chip LLMs, which is further analyzed in Section IV-D.

D. Evaluation on Instruction Alignment and Domain Knowledge

In addition to evaluating ChipAlign on both the OpenROAD QA and industrial chip QA benchmarks, which demand a joint capability of instruction alignment and domain knowledge, we further assess these two capabilities of ChipAlign separately.

Specifically, we assess instruction alignment using IFEval, a well-known benchmark designed to test LLMs' ability to follow instructions. IFEval comprises 541 prompts featuring various general instructions, such as "List exactly 10 possible names ..." and "Think step-by-step and then give your answer ...". Accuracy for each LLM is calculated based on the number of prompts or instructions that are either strictly (Strict Accuracy) or loosely (Loose Accuracy) adhered to.

Our results presented in Table 3 showcase that LLaMA3-8B-ChipAlign significantly outperforms LLaMA3-8B-EDA in instruction-following accuracy and matches the performance of LLaMA3-8B-Instruct. More importantly, LLaMA2-70B-ChipAlign not only improves the accuracy of LLaMA2-70B-ChipNeMo by 26.6% on average, but also considerably surpasses the source instruction model,

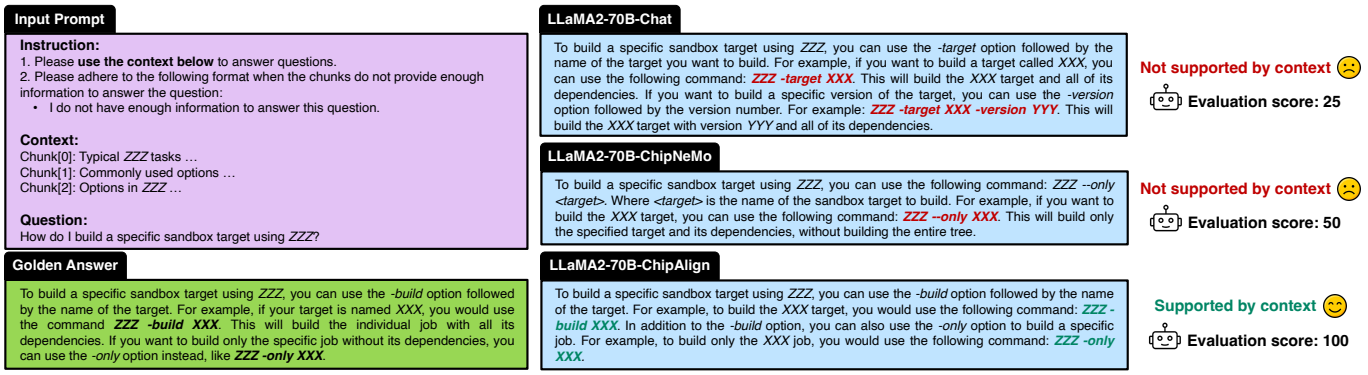


Figure 6: An example from the BUILD category of the industrial chip QA benchmark — Due to space limitations, only the first few words of each context chunk are shown, omitting full context. Besides, we redact the build tool and target names using *ZZZ* and *XXX*, respectively.

Table 2: GPT4-aided evaluation scores on industrial production-level chip QA benchmarks.

	Single Turn					Multi Turn				
Method	ARCH	BUILD	LSF	TESTGEN	All	ARCH	BUILD	LSF	TESTGEN	All
LLaMA2-70B-Chat	40.00	70.50	54.25	66.75	57.00	10.00	41.00	10.50	50.00	30.25
LLaMA2-70B-ChipNeMo	45.00	81.75	60.50	87.50	66.75	40.00	**81.75**	52.00	33.25	54.50
LLaMA2-70B-ChipAlign	**57.50**	**84.00**	**70.75**	**91.75**	**74.25**	**42.50**	75.00	**60.50**	**79.25**	**62.75**

Table 3: Instruction-following accuracy (%) on IFEval.

	Prompt Level		Instruction Level	
Method	Strict	Loose	Strict	Loose
LLaMA3-8B-Instruct	67.5	**75.4**	76.1	**82.6**
LLaMA3-8B-EDA	60.0	64.6	69.6	74.1
LLaMA3-8B-ChipAlign	**69.9**	74.1	**78.3**	81.9
LLaMA2-70B-Chat	43.2	50.1	55.2	61.5
LLaMA2-70B-ChipNeMo	27.9	32.0	39.8	44.4
LLaMA2-70B-ChipAlign	**54.3**	**61.0**	**65.0**	**70.3**

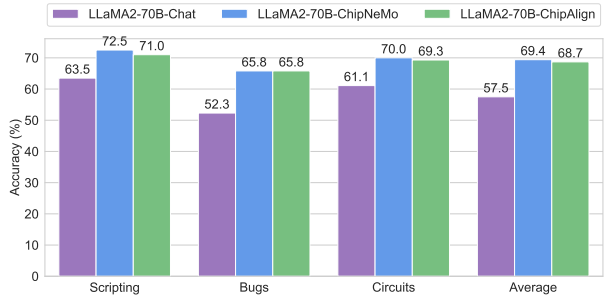


Figure 7: Results on multi-choice chip QA benchmark.

LLaMA2-70B-Chat. This enhanced performance can be attributed to the integration of the OASST instruction dataset [11] and the SteerLM alignment strategy [7] during the training of LLaMA2-70B-ChipNeMo, which imbued its weights with decent instructional knowledge complementary to that of LLaMA2-70B-Chat. Consequently, ChipAlign benefits from the combined instructional knowledge of LLaMA2-70B-ChipNeMo and LLaMA2-70B-Chat through the process of weight fusion. This fusion results in a merged model with stronger instruction alignment capabilities compared to both source models.

For evaluating pure chip domain knowledge, we utilize multi-choice chip QA benchmarks from [14] that contain no instructions. Figure 7 shows that ChipAlign performs on par with ChipNeMo across the domains of EDA scripts, bugs, and circuits, highlighting its ability to preserve domain knowledge after weight fusion. Furthermore, the comparative performance of LLaMA2-70B-Chat, LLaMA2-70B-ChipNeMo, and LLaMA2-70B-ChipAlign is visualized in Figure 2, providing a comprehensive overview of their respective capabilities.

E. Sensitivity Analysis on λ

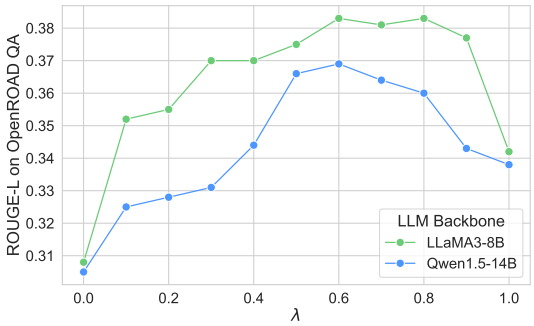


Figure 8: Sensitivity analysis on λ in ChipAlign.

As discussed in Section III, ChipAlign involves a single hyper-parameter λ that determines how closely the merged model aligns with either the instruction model M_{instruct} or the chip model M_{chip} along the geodesic. We conduct a sensitivity analysis of λ using both LLaMA3-8B and Qwen1.5-14B backbones on the OpenROAD QA benchmark. Notably, $\lambda = 0$ and $\lambda = 1$ correspond to the M_{instruct} and M_{chip}, respectively. As depicted in Figure 8, model performance initially increases rapidly from M_{instruct} (i.e., the leftmost point), peaks at $\lambda = 0.6$, and subsequently declines towards the performance level of M_{chip} (i.e., the rightmost point). Hence, we recommend setting $\lambda = 0.6$ as the default value in ChipAlign for practical applications.

V. Conclusion

This work introduces ChipAlign, a geometric-aware model merging approach to enhance instruction alignment in chip LLMs. ChipAlign treats the weights of a chip LLM and a general instruction LLM as two points on a Riemannian manifold and employs geodesic interpolation to create the merged model. Our results demonstrate that this merged model significantly improves instruction alignment in chip LLMs, yielding superior performance across various chip QA benchmarks.

REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Shun-ichi Amari. Information geometry and manifolds of neural networks. In *From Statistical Physics to Statistical Inference and Back*, pages 113–138. Springer, 1994.

[3] Shun-ichi Amari, Koji Kurata, and Hiroshi Nagaoka. Information geometry of boltzmann machines. *IEEE Transactions on neural networks*, 3(2):260–271, 1992.

[4] Anthropic. The claude 3 model family: Opus, sonnet, haiku. *Technical Report*, 2024.

[5] Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *arXiv preprint arXiv:2306.16092*, 2023.

[6] Pala Tej Deep, Rishabh Bhardwaj, and Soujanya Poria. Della-merging: Reducing interference in model merging through magnitude-based sampling. *arXiv preprint arXiv:2406.11617*, 2024.

[7] Yi Dong, Zhilin Wang, Makesh Narsimhan Sreedhar, Xianchao Wu, and Oleksii Kuchaiev. Steerlm: Attribute conditioned sft as an (user-steerable) alternative to rlhf. *arXiv preprint arXiv:2310.05344*, 2023.

[8] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[9] Sreyan Ghosh, Chandra Kiran Reddy Evuru, Sonal Kumar, Deepali Aneja, Zeyu Jin, Ramani Duraiswami, Dinesh Manocha, et al. A closer look at the limitations of instruction tuning. *arXiv preprint arXiv:2402.05119*, 2024.

[10] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. *arXiv preprint arXiv:2212.04089*, 2022.

[11] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] Yadong Li, Haoze Sun, Mingan Lin, Tianpeng Li, Guosheng Dong, Tao Zhang, Bowen Ding, Wei Song, Zhenglin Cheng, Yuqi Huo, et al. Baichuan-omni technical report. *arXiv preprint arXiv:2410.08565*, 2024.

[13] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[14] Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*, 2023.

[15] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[16] Yuan Pu, Zhuolun He, Tairu Qiu, Haoyuan Wu, and Bei Yu. Customized retrieval augmented generation and benchmarking for eda tool documentation qa. *arXiv preprint arXiv:2407.15353*, 2024.

[17] Utsav Sharma, Bing-Yue Wu, Sai Rahul Dhanvi Kankipati, Vidya A Chhabria, and Austin Rovinski. Openroad-assistant: An open-source large language model for physical design tasks. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, pages 1–7, 2024.

[18] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[19] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

[20] Saeid Ashraf Vaghefi, Dominik Stammbach, Veruska Muccione, Julia Bingler, Jingwei Ni, Mathias Kraus, Simon Allen, Chiara Colesanti-Senni, Tobias Wekhof, Tobias Schimanski, et al. Chatclimate: Grounding conversational ai in climate science. *Communications Earth & Environment*, 4(1):480, 2023.

[21] Mitchell Wortsman, Gabriel Ilharco, Samir Ya Gadre, Rebecca Roelofs, Raphael Gontijo-Lopes, Ari S Morcos, Hongseok Namkoong, Ali Farhadi, Yair Carmon, Simon Kornblith, et al. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, pages 23965–23998. PMLR, 2022.

[22] Chaoyi Wu, Weixiong Lin, Xiaoman Zhang, Ya Zhang, Weidi Xie, and Yanfeng Wang. Pmc-llama: toward building open-source language models for medicine. *Journal of the American Medical Informatics Association*, page ocae045, 2024.

[23] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[24] Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*, 2023.

[25] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighof. C-pack: Packaged resources to advance general chinese embedding. *arXiv preprint arXiv:2309.07597*, 2023.

[26] Prateek Yadav, Derek Tam, Leshem Choshen, Colin A Raffel, and Mohit Bansal. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36, 2024.

[27] Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *arXiv preprint arXiv:2408.07666*, 2024.

[28] Le Yu, Bowen Yu, Haiyang Yu, Fei Huang, and Yongbin Li. Language models are super mario: Absorbing abilities from homologous models as a free lunch. In *Forty-first International Conference on Machine Learning*, 2024.

[29] Tianjun Zhang, Shishir G Patil, Naman Jain, Sheng Shen, Matei Zaharia, Ion Stoica, and Joseph E Gonzalez. Raft: Adapting language model to domain specific rag. *arXiv preprint arXiv:2403.10131*, 2024.

[30] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[31] Ming Zhong, Yang Liu, Da Yin, Yuning Mao, Yizhu Jiao, Pengfei Liu, Chenguang Zhu, Heng Ji, and Jiawei Han. Towards a unified multi-dimensional evaluator for text generation. *arXiv preprint arXiv:2210.07197*, 2022.

[32] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

Security of Approximate Neural Networks against Power Side-channel Attacks

Aditya Japa*, Jack Miskelly**, Maire O'Neill**, Chongyan Gu**
*School of Computing, Engineering and Intelligent Systems, Ulster University, Derry, U.K.
**Centre for Secure Information Technologies, Queen's University Belfast, Belfast, U.K.
Email: a.japa@ulster.ac.uk, and c.gu@qub.ac.uk

***Abstract*—Emerging low-energy computing technologies, in particular approximate computing, are becoming increasingly relevant in key applications. A significant use case for these technologies is reduced energy consumption in Artificial Neural Networks (ANNs), an increasingly pressing concern with the rapid growth of AI deployments. It is essential we understand the security implications of approximate computing in an ANN context before this practice becomes commonplace. In this work, we examine the test case of approximate ANN processing elements (PE) in terms of information leakage via the power side channel. We perform a weight extraction correlation Power Analysis (CPA) attack under three approximation scenarios: overclocking, voltage scaling, and circuit level bitwise approximation. We demonstrate that as the degree of approximation increases the Signal to Noise Ratio (SNR) of power traces rapidly degrades. We show that the Measurement to Disclosure (MTD) increases for all approximate techniques. An MTD of 48 under precise computing is increased to at minimum 200 (bitwise approximate circuit at 25% approximation), and under some approximation scenarios >1024. i.e. an increase in attack difficulty of at least x4 and potentially over x20. A relative Security-Power-Delay (SPD) analysis reveals that, in addition to the across the board improvement vs precise computing, voltage and clock scaling both significantly outperform approximate circuits with voltage scaling as the highest performing technique.**

***Index Terms*—Neural network hardware, Power Side-channel attacks, Approximate computing, Artificial intelligence**

I. Introduction

Driven by the increasing global demand for high performance centralized compute resources, the total energy demands of datacentres and compute clusters has been growing at a breakneck pace in recent years. One of the key factors in this has been the sudden explosion of computationally intensive ANN based AI applications. For both environmental and economic reasons there is now a strong drive towards more energy efficient computing hardware in this context. One of the key techniques for low-energy computing is approximate computing (sometimes also called transprecise or inexact computing). The approximate computing paradigm, in essence, reduces energy consumption by removing the constraint that every output bit of a computation must be correct. Instead a controlled and tolerable degree of error is permitted. This, in turn, allows for increases in energy efficiency or throughput relative to an exact circuit. The corollary is that to take full advantage of approximate computing requires an error-tolerant application. ANNs, which possess an inherent error tolerance, are therefore one of the areas where approximation will have the widest impact.

When it comes to the secure information in ANNs there are two major areas of concern. The first is the information being processed by the network which in itself may be sensitive, especially as deployments become more common in areas like health and finance. The second are the weights and structure of the network itself. Model training is a high effort and resource intensive task. This makes the results of training, especially the weights, commercially sensitive. If an adversary can acquire them they can in effect steal the model, skipping the laborious training step but gaining all the benefits of it. In both cases there are clear incentives for attackers. Extracting this information as a purely remote attack is challenging. However, this changes when considering attackers with Side Channel Analysis (SCA) capability. In brief, SCA involves the analysis of characteristics such as power consumption, electromagnetic radiation emissions, and timing which are not intended to carry information but which nonetheless respond to operations in the system. Power draw is of particular significance in an ANN context, because there is a strong correlation between node weights, inputs, and power consumption.

SCA may seem like a difficult thing to exploit against the largely centralized compute clusters used for ANNs currently, but this state of affairs is unlikely to last. There is an increasing interest in edge deployments of ANNs to avoid the inherent inefficiencies and latency issues of centralized clusters. The latter point is especially significant for real time applications. Edge deployments inevitably expose devices to physical level attacks like SCA. When we consider that energy budget is even more critical for these systems, the future edge deployment of approximate ANNs (AANNs) seems highly probable. As approximation alters power characteristics, it seems intuitive that it will have an impact on SCA capability. It is essential we understand what exactly this impact will be in advance. While there is a substantial existing body of work on SCA against ANNs, and there has been some investigation of the security of approximate systems in a general sense, so far there has been no close analysis of the intersection of these technologies.

To rectify that, in this work we investigate how approximation of ANN Processing elements impacts their vulnerability to SCA. More specifically, we are considering an attacker with power monitoring capability who aims to extract the

This work was supported by EPSRC (UK) under the Grant EP/X009602/1.

weights from the network and thereby steal the IP of the network owner. The question is, will the approximation help or hinder the weight stealing attack and to what degree? In this work we will demonstrate that approximation significantly reduces the weight information leakage through the power side channel. Further, that the different forms of approximation offer markedly different levels of protection. This provides a valuable insight for future hardware designers aiming to combine these technologies.

The specific contributions are as follows:

- We provide a power analysis of approximate PEs under three approximation scenarios: clock scaling, voltage scaling, and bitwise approximate circuitry.
- We demonstrate, for the first time, that approximation of all types worsens the SNR of power traces from an attacker's perspective and that the rate of change of SNR relative to degree of approximation varies significantly between methods of approximation.
- We demonstrate how this leads to a higher requirement in terms of MTD for weight stealing attacks, with an increase in attack difficulty of at least x4 in the worst case and more than x20 in the best case.
- We provide a Security-Power-Delay (SPD) analysis of the different methods of approximation, which reveals significant differences in relative performance when security is considered.

II. Background

An ANN consists of layers of interconnected computational units (artificial neurons) which can also be called nodes. There are typically at least three layers: an input layer, one or more internal hidden layers, and a final output layer as shown in Fig.1. Each node represents a single neuron that produces a weighted sum. The nodes of each layer are connected to the nodes of the next layer with each such connection being assigned a weight. The nodes themselves implement an activation function which takes the full set of individual inputs and their weights and converts them to a single output. The number of nodes in each layer, the number of hidden layers, and the specific activation function used are implementation specific factors. Weights are determined via model training, in which the model is provided with labelled training data and a defined loss function (i.e. a function which quantifies the difference between predicted output and actual output). Weights are iteratively updated over multiple rounds of training until the loss function is sufficiently minimized. Training is resource intensive and may make use of proprietary training data. As such the final trained weights can have significant commercial value. Knowledge of the weights, layer structure, and activation function is sufficient to create an exact duplicate of the original model.

A. ANNs

Key hardware elements for ANNs are the implementation of the activation function for the nodes and the memory used to store the weight values. Activation functions are simply

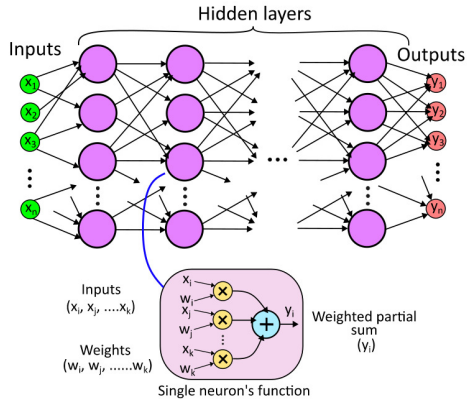


Fig. 1: ANN diagram with different layers

mathematical functions and can be implemented on general purpose hardware, but in accelerated architectures they often receive a dedicated hardware implementation for performance reasons. As the activation function is the primary computation of the network the throughput of the overall network is strongly tied to the throughput of the activation function in each node (the other major factor is the latency of memory accesses). Weights generally must be held in a memory of some kind if the ANN hardware is intended to be adjustable, but the distribution of this memory (per-connection, per-node, per-layer, etc.) can vary. The alternative is to embed the values directly as a physical property of the circuit, but in practice this approach is rare.

B. SCA of ANNs

From a security perspective there are several key areas where side channel leakage must be considered. First, the structure of the network in terms of number of nodes per layer, number of hidden layers, activation function, etc. i.e. the model architecture. These properties are difficult to conceal, and architectural reverse engineering attacks have been demonstrated using system resource allocation/memory access volume [1] [2], cache timing side channel [3], EM side channel [4] and power side channel [5] [6] [7]. This is not a threat limited to simplified cases: Yan et al. demonstrated such an attack using power analysis against complex, non-binarized networks on Nvidia's NVDLA architecture [8]. Knowledge of these properties is advantageous to an adversary in itself, because they both provide a grounding for performing weight-stealing attacks and allow the adversary to greatly reduce the cost of training their own model for a similar task. There is therefore a strong financial incentive for such attacks with or without a weight-stealing follow up attack.

Second, the weights of each connection in the network. If the exact weights can be acquired in addition to the structure being known, the adversary can exactly replicate (i.e. steal) the trained model. Batina et al. performed an EM side channel based attack on ARM based edge devices, which was able to extract not only weights but also the full architectural details [9]. Maji et al. performed a similar weight stealing

attack on ARM, ATmega, and RISC-V based devices using simple power analysis and timing information [10]. Yoshida et al. demonstrated both EM [11] and power analysis [12] attacks against models with encrypted weights implemented on FPGA. Dubey at al. similarly demonstrated the ability to fully extract weights from FPGA using power analysis [13] and proposed masking methods to counter such attacks. This is not a comprehensive list, but it suffices to demonstrate that exact weight extraction is a tangible threat, especially for edge devices.

Third, the data used in training and operation. Attacks which seek to learn the properties of input data based on model output are called model inversion attacks. These have been known in a general sense for almost ten years now [14], and specific proposals have been made which exploit both timing [15] and power [16] side channels to achieve the inversion. Property and membership inference attacks both aim to derive information about the original training data from the model. This form of attack has been demonstrated a number of times in a federated learning context (e.g. [17]–[20]) but the application of side channel analysis in this type of attack remains largely unexplored.

III. Experimental Setup for Evaluating Approximate ANNs

A. Approximate Computing in Neural Networks

Approximation as a technique for efficient ANNs is an area of active interest, with upwards of 30 works being published each year at the major design automation conferences alone [21], [22]. This is too broad a field to closely examine in the context of a short paper, but it suffices to know that there are two main forms of approximate computing used in this context: Voltage and Frequency Scaling (VFS) and circuit level bitwise approximation. Frequency scaling involves increasing the operating frequency (overclocking) for increased throughput. Voltage scaling reduces the operating voltage (undervolting) for reduced power consumption. In both cases the tradeoff is a set of semi-random errors at each layer based on the distribution of critical paths (the data paths with the highest propagation delays). In these methods, while the operating characteristics of the overall circuit are changed the hardware of the processing elements are the same as in an exact ANN. In contrast to these methods are bitwise approximate circuits, where the fundamental PEs are redesigned such that specific bits of the operation (the activation function in an ANN context) are approximated, while the remaining bits remain exact. With this method, error distribution is more controlled at the cost of requiring custom hardware. In a general sense, all of these methods can be viewed as increasing efficiency at the cost of introducing noise to the system. The key design choice is what error rate (i.e. how much noise) and distribution can be tolerated without an unacceptably high loss of model accuracy.

The goal of this work is to evaluate the *relative* leakage of different methods of approximation, rather than a specific ANN hardware implementation. As such, we have selected as our test case a relatively simple and readily approximable PE based on the design proposed in [23], consisting of two 4-bit multipliers and an 8-bit adder as can be seen in Fig. 2. This produces a weighted sum value as described in equation 1:

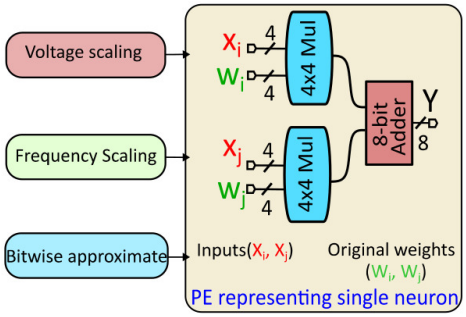


Fig. 2: Design of PE representing single neuron and applied approximate computing techniques.

$$Y = x_i \cdot w_i + x_j \cdot w_j \tag{1}$$

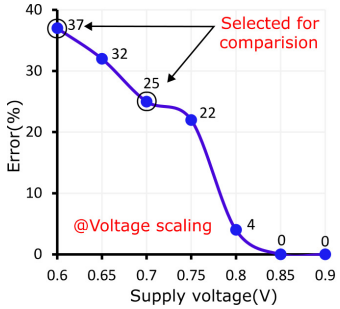
where Y is the weighted sum, xi, xj are the inputs and wi, wj are the associated weights. Multiple versions of this PE were implemented with varying levels of bitwise approximation from 0 bits (exact design) to 4 bits (50% approximation). More specifically, the bitwise approximation is achieved by replacing all full adders with approximate full adders (this includes those internal to the multipliers as well as the final adder). In all cases the least significant bits are the approximated bits. For VFS evaluation the voltage or frequency of the exact design was varied. As can be seen in Fig 3, to ensure like-for-like comparison two voltage and two frequency points were selected which produce the closest error rate to the 2 and 3 bit bitwise approximation levels. i.e. 25% and 37.5% error respectively. The overall degree of noise in each subsequent comparison varies by no more than 2% between methods. The desired points were determined empirically by stepping through the voltage and frequency ranges until two suitable points were found. The implementation is targeted at 45nm CMOS and simulated using Cadence EDA tools. Again, because the goal here is simply to evaluate the previously untested relative leakage of different approximation methods a simulated test of a single PE suffices to draw some initial conclusions about how these technologies may interact.

IV. CPA Attack on AANN

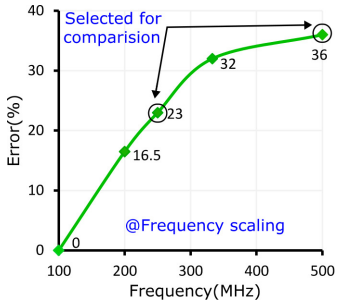
This section describes the weight-stealing CPA attack we use as our test case for leakage evaluation.

A. Attack Scenario

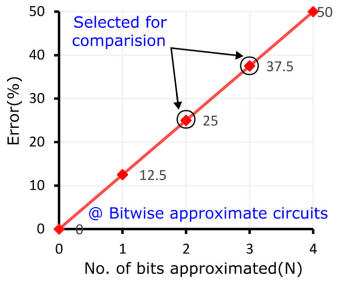
We assume an attack scenario with a relatively strong adversary. The adversary is assumed to have the ability to measure power traces from any internal operation without disrupting that operation. It is further assumed that the adversary can send arbitrary inputs to the ANN at will, and that the adversary has full knowledge of the hardware design - in particular the



(a) Varying supply voltage (voltage scaling technique)



(b) Varying clock frequency (frequency scaling technique)



(c) Increasing number of approximate bits in bitwise approximate circuit

Fig. 3: Percentage of error of PE under different approximation methods

structure of the PEs themselves. The only items of information concealed from the adversary are the specific weights of the deployed network and the original training data. The goal of the adversary is to steal those weight values. For a real world analogy to this threat model, consider a hostile customer who has purchased a naively implemented edge device (therefore, has access to the physical device and to any documentation) and who is attempting to extract a proprietary model from it.

B. Attack Mechanism

For the attack we use a Correlation Power Analysis (CPA) based approach, focusing on the multiply-add operation within the PE. The attack mechanism, visualized in Fig. 4, uses the following steps:

1) Two weight values (w_i and w_j) are selected.
2) Every possible combination of the input values x_i and x_j are applied in order to the PE using the fixed weights w_i and w_j. The power traces associated with each weighted sum execution is recorded and sampled to form a matrix, A, of size $m \times n$ (where m is the number of input combinations and n is the samples per trace).
3) On the other side, the PE algorithm is developed using python based Jupyter note book. An identical input sequence is applied with all possible combinations of weights and associated responses are recorded.
4) We use Hamming Distance (HD) to calculate hypothetical power consumption. The HD of two successive outputs of PE is calculated as shown in Eq. 2. This is repeated for each pair of outputs.

$$HD = Hamming\ Weight(Yi \oplus Y_{i+1}) \qquad (2)$$

5) The obtained hypothetical power consumption values are arranged in a matrix (H) with a size of $m \times k$ where m and k represent the number of inputs and number of weights respectively.
6) Finally, each column of H is correlated with all the columns of A, producing a correlation matrix (C) with size $k \times n$. The original weights of PE uniquely show the highest value in the correlation matrix compared to the other keys.

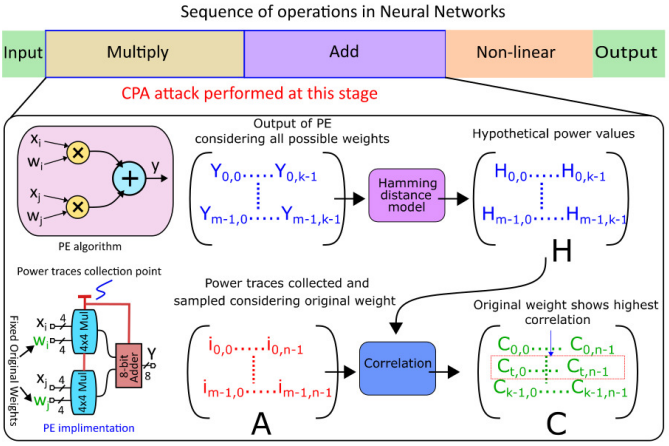


Fig. 4: CPA attack mechanism for retrieving weight information of processing element

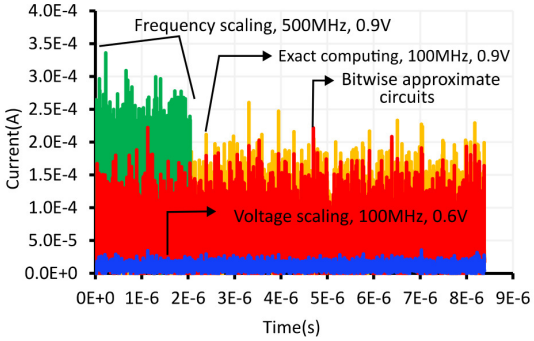
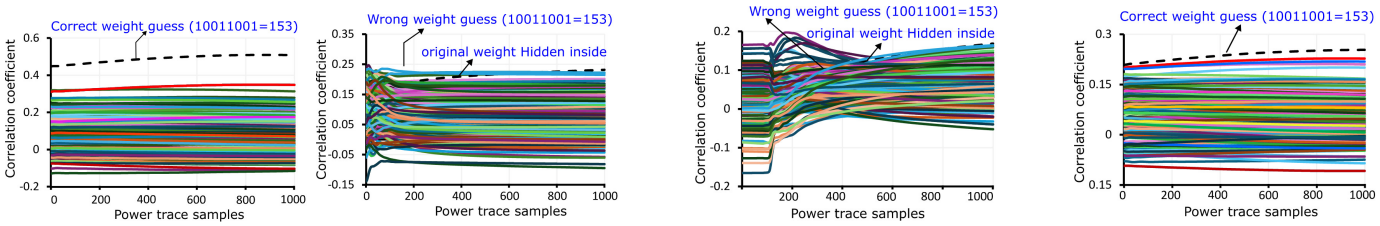


Fig. 5: Sample power traces of PE with different approximate computing techniques (37% in output response).

V. Results and Analysis

Fig. 5 shows the current measured with each form of approximation during an experiment where $w_i = 1001$, $w_j = 1001$, at a fixed error rate of 37%. This quite clearly illustrates the differences in the approaches from a power analysis perspective. While all three are producing the same degree of error relative to the exact case and working with the same weights and inputs, there is a drastic difference



(a) Exact PE (b) Voltage scaled PE (c) Frequency scaled PE (d) Bitwise approximate PE

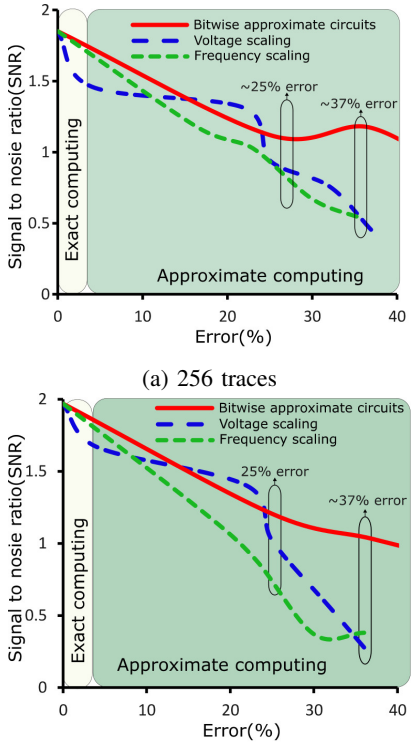
Fig. 6: Correlation coefficients of each technique using 256 power traces

in current. Bitwise approximation remains relatively close to the exact case over time with a moderate reduction in current overall. Frequency scaling is time-compressed and with drastically higher current draw, while voltage scaling exhibits a similarly drastic reduction.

Fig. 6 depicts the correlation coefficient results of all 256 possible weight values for different PE designs using 256 power traces of 1000 samples each. The level of approximation is 25% and the correlation coefficient of the original weight (w=10011001=153) is shown as a dotted line. It can be observed that in the exact case the attack is fairly trivial, giving a distinctly higher correlation with the correct weight (w=153). When we examine the approximate cases, it can be seen that each approximate PE exhibits a relatively lower correlation coefficient for the original weight compared to the exact PE design. Against voltage and frequency scaling the attack was unable to clearly distinguish the original weight. The attack on the bitwise approximated PE still succeeded, but the distinction between the original weight and the set of other weights is much less than in the exact case.

Clearly, there is a difference between these techniques not only in actual power consumption but also in the amount of useful information in the power trace. To get a clearer picture of what is going on here, we calculated the Signal to Noise Ratio (SNR) from an attack perspective. In this context the SNR is the ratio of the original weight correlation coefficient to the maximum value of the wrong weight guess which is expressed in [24]. i.e. SNR decides the level of difficulty in distinguishing between the original weight and the most likely wrong weight guess. High SNR is better for the attacker and is therefore undesirable. Fig. 7 shows how SNR changes with increasing approximation for each approximate technique. While it can be seen that, as expected from the previous results, all three have lower SNR at higher levels of approximation the behavior is not uniform. Voltage and frequency scaling continue to decline largely proportionally, while bitwise approximation tapers off. In addition, the reduction in SNR is more rapid in the voltage and frequency scaling cases. This accounts for the difference in the example attack shown in Fig. 6, as by the 25% error rate point the SNR of the voltage and frequency scaled PEs is already much lower than the bitwise approximated PE. With higher levels of approximation this gap only widens.

Fig. 8(a-g) shows the measurement to disclosure (MTD) for



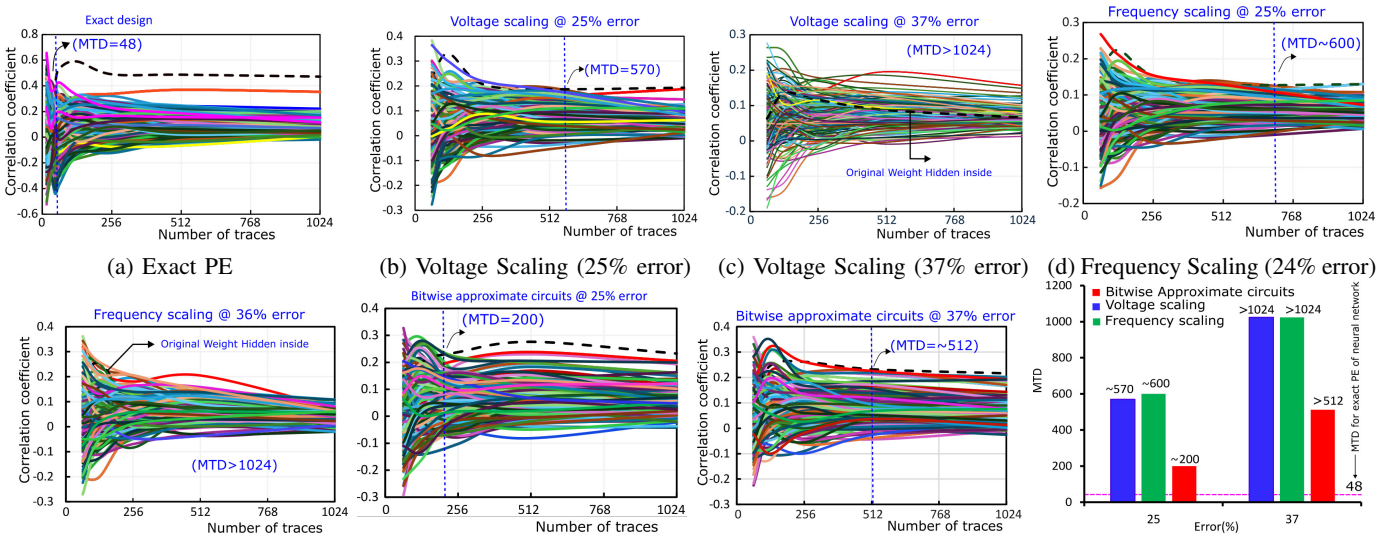
(a) 256 traces

(b) 512 traces

Fig. 7: SNR of PE design varying against output error

each test case. MTD is defined as the minimum number of power traces required to retrieve the original weight information [25], [26]. To calculate this, the attack was repeated using 256, 512, 768, and 1024 traces. The point where the dotted line (original weight) becomes distinct from the general body of potential weights is the MTD. In this test, the baseline value is 48 for an exact PE. Voltage scaling increases this to 570 at 25% error, and above 1024 at 37% error. Frequency scaling likewise increases to roughly 600 at 25% error and somewhere above 1024 at 37% error. As expected from the SNR analysis, bitwise approximation performs better than the exact case but worse than the VFS methods with MTD of 200 at 25% error and roughly 512 at 37% error.

If one considers only information leakage these results show a clear advantage to voltage and frequency scaling. However, the primary design goal of approximation is not security



(a) Exact PE (b) Voltage Scaling (25% error) (c) Voltage Scaling (37% error) (d) Frequency Scaling (24% error)

(e) Frequency Scaling (36% error) (f) Bitwise (~25% error) (g) Bitwise (~37% error) (h) Comparison for all PE designs

Fig. 8: MTD comparison of approximation method and error rate

TABLE I: Performance comparison of PE designs against CPA

Metrics	**Exact Computing**	**Approximate Computing Techniques**				**Comments**
		Approximation Level	***Voltage scaling***	***Frequency scaling***	***Bitwise approximation***	
MTD	~48	~25%	~570	~600	~200	~4× to ~12× improvement
SPD(fJ)-1	~0.2		~19.5	~11.5	~5	~5× to ~20× improvement
MTD	~48	~37%	>1024	>1024	>512	~11× to ~21× improvement
SPD(fJ)-1	~0.2		~6.9	~6.5	~3.1	~15× to ~35× improvement

but rather efficiency and performance. To better evaluate the performance of each method in this context this work redefines/introduces the security, power, and delay (SPD) metric. The SPD is a figure-of-merit metric that combines the benefits of approximate computing related to security and the system's power consumption and delay [25], [26]. The SPD is the ratio of MTD and the product of power delay as shown in equation 3

$$SPD = \frac{MTD}{Power\ Consumption * Delay} \tag{3}$$

Fig. 9 compares the SPD for each method based on the measurements taken during our experiments. It is evident that the exact PE design shows lower SPD due to the lower MTD, higher power consumption, and higher delay values. Orthogonal to this, approximate computing techniques significantly improve the SPD of PE designs. With a ~25% approximation, PE designs achieve ~20×, ~12× and ~5× higher SPD with voltage scaling, overclocking and approximate circuits respectively when compared with the exact counterpart. Similarly, with a ~37% approximation, PE designs achieve ~35×, ~33×, and 16× higher SPD with voltage scaling, overclocking and approximate circuits respectively. Among the three techniques, the voltage scaling technique achieves higher SPD and approximate circuits techniques show relatively lower SPD. The results have been summarized in Table 1.

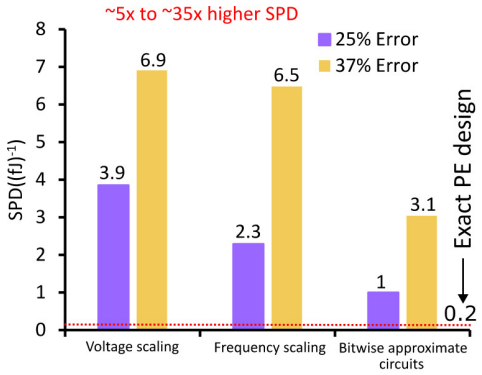


Fig. 9: SPD comparison of PE designs with different approximate computing techniques

VI. Conclusion

In this work we have shown that, in an ANN context, approximate computing techniques applied to the fundamental processing elements have a significant and benign impact on information leakage through the power side channel with a minimum increase in attack complexity of $\times 4$ and upwards of $\times 35$ possible. Further, that the magnitude of this effect varies significantly depending on the method and degree of approximation. This provides useful insight for the design of secure and efficient ANNs in the future.

REFERENCES

[1] X. Hu, L. Liang, S. Li, L. Deng, P. Zuo, Y. Ji, X. Xie, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Deepsniffer: A dnn model extraction framework based on learning architectural hints," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '20, (New York, NY, USA), p. 385–399, Association for Computing Machinery, 2020.

[2] K. Patwari, S. M. Hafiz, H. Wang, H. Homayoun, Z. Shafiq, and C.-N. Chuah, "Dnn model architecture fingerprinting attack on cpu-gpu edge devices," in *2022 IEEE 7th European Symposium on Security and Privacy (EuroSP)*, pp. 337–355, 2022.

[3] Y. Liu and A. Srivastava, "Ganred: Gan-based reverse engineering of dnns via cache side-channel," in *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW'20, (New York, NY, USA), p. 41–52, Association for Computing Machinery, 2020.

[4] Ł. Chmielewski and L. Weissbart, "On reverse engineering neural network implementation on gpu," in *Applied Cryptography and Network Security Workshops: ACNS 2021 Satellite Workshops, AIBlock, AIHWS, AIoTS, CIMSS, Cloud S&P, SCI, SecMT, and SiMLA, Kamakura, Japan, June 21–24, 2021, Proceedings*, pp. 96–113, Springer, 2021.

[5] S. Tian, S. Moini, A. Wolnikowski, D. Holcomb, R. Tessier, and J. Szefer, "Remote power attacks on the versatile tensor accelerator in multi-tenant fpgas," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 242–246, IEEE, 2021.

[6] Y. Zhang, R. Yasaei, H. Chen, Z. Li, and M. A. Al Faruque, "Stealing neural network structure through remote fpga side-channel analysis," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4377–4388, 2021.

[7] V. Meyers, D. Gnad, and M. Tahoori, "Reverse engineering neural network folding with remote fpga power analysis," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 1–10, IEEE, 2022.

[8] X. Yan, X. Lou, G. Xu, H. Qiu, S. Guo, C. H. Chang, and T. Zhang, "Mercury: An automated remote side-channel attack to nvidia deep learning accelerator," in *2023 International Conference on Field Programmable Technology (ICFPT)*, pp. 188–197, 2023.

[9] L. Batina, S. Bhasin, D. Jap, and S. Picek, "CSI NN: Reverse engineering of neural network architectures through electromagnetic side channel," in *28th USENIX Security Symposium (USENIX Security 19)*, (Santa Clara, CA), pp. 515–532, USENIX Association, Aug. 2019.

[10] S. Maji, U. Banerjee, and A. P. Chandrakasan, "Leaky nets: Recovering embedded neural network models and inputs through simple power and timing side-channels—attacks and defenses," *IEEE Internet of Things Journal*, vol. 8, no. 15, pp. 12079–12092, 2021.

[11] K. Yoshida, T. Kubota, M. Shiozaki, and T. Fujino, "Model-extraction attack against fpga-dnn accelerator utilizing correlation electromagnetic analysis," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pp. 318–318, 2019.

[12] K. Yoshida, T. Kubota, S. Okura, M. Shiozaki, and T. Fujino, "Model reverse-engineering attack using correlation power analysis against systolic array based neural network accelerator," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.

[13] A. Dubey, R. Cammarota, and A. Aysu, "Maskednet: The first hardware inference engine aiming power side-channel protection," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 197–208, 2020.

[14] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An {End-to-End} case study of personalized warfarin dosing," in *23rd USENIX security symposium (USENIX Security 14)*, pp. 17–32, 2014.

[15] Z. Chen, M. Tang, and J. Li, "Inversion attacks against cnn models based on timing attack," *Security and Communication Networks*, vol. 2022, no. 1, p. 6285909, 2022.

[16] L. Wei, B. Luo, Y. Li, Y. Liu, and Q. Xu, "I know what you see: Power side-channel attack on convolutional neural network accelerators," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, (New York, NY, USA), p. 393–406, Association for Computing Machinery, 2018.

[17] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 619–633, 2018.

[18] M. Chase, E. Ghosh, and S. Mahloujifar, "Property inference from poisoning," *arXiv preprint arXiv:2101.11073*, 2021.

[19] C. Fu, X. Zhang, S. Ji, J. Chen, J. Wu, S. Guo, J. Zhou, A. X. Liu, and T. Wang, "Label inference attacks against vertical federated learning," in *31st USENIX security symposium (USENIX Security 22)*, pp. 1397–1414, 2022.

[20] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE symposium on security and privacy (SP)*, pp. 691–706, IEEE, 2019.

[21] G. Armeniakos, G. Zervakis, D. Soudris, and J. Henkel, "Hardware approximate techniques for deep neural network accelerators: A survey," *ACM Comput. Surv.*, vol. 55, Nov. 2022.

[22] F. Regazzoni and I. Polian, "Side channel attacks vs approximate computing," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, GLSVLSI '20, (New York, NY, USA), p. 321–326, Association for Computing Machinery, 2020.

[23] A. Balaji, S. Ullah, A. Das, and A. Kumar, "Design methodology for embedded approximate artificial neural networks," in *Proceedings of the 2019 Great Lakes Symposium on VLSI*, GLSVLSI '19, (New York, NY, USA), p. 489–494, Association for Computing Machinery, 2019.

[24] S. D. Kumar, H. Thapliyal, and A. Mohammad, "Finsal: A novel finfet based secure adiabatic logic for energy-efficient and dpa resistant iot devices," in *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pp. 1–8, 2016.

[25] I. M. Delgado-Lozano, E. Tena-SÁnchez, J. NÚÑez, and A. J. Acosta, "Design and analysis of secure emerging crypto-hardware using hyperfet devices," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 2, pp. 787–796, 2021.

[26] E. Tena-Sánchez and A. J. Acosta, "Logic minimization and wide fan-in issues in dpl-based cryptocircuits against power analysis attacks," *International Journal of Circuit Theory and Applications*, vol. 47, no. 2, pp. 238–253, 2019.

E-morphic: Scalable Equality Saturation for Structural Exploration in Logic Synthesis

Chen Chen*
HKUST(GZ)
cchen099@connect.hkust-gz.edu.cn

Guangyu Hu*†
HKUST
ghuae@connect.ust.hk

Cunxi Yu
University of Maryland, College Park
cunxiyu@umd.edu

Yuzhe Ma
HKUST(GZ)
yuzhema@hkust-gz.edu.cn

Hongce Zhang†
HKUST(GZ)
hongcezh@hkust-gz.edu.cn

***Abstract*—In technology mapping, the quality of the final implementation heavily relies on the circuit structure after technology-independent optimization. Recent studies have introduced equality saturation as a novel optimization approach. However, its efficiency remains a hurdle against its wide adoption in logic synthesis. This paper proposes a highly scalable and efficient framework named E-morphic. It is the first work that employs equality saturation for resynthesis after conventional technology-independent logic optimizations, enabling structure exploration before technology mapping. Powered by several key enhancements to the equality saturation framework, such as direct e-graph-circuit conversion, solution-space pruning, and simulated annealing for e-graph extraction, this approach not only improves the scalability and extraction efficiency of e-graph rewriting but also addresses the structural bias issue present in conventional logic synthesis flows through parallel structural exploration and resynthesis. Experiments show that, compared to the state-of-the-art delay optimization flow in ABC, E-morphic on average achieves 12.54% area saving and 7.29% delay reduction on the large-scale circuits in the EPFL benchmark.**

I. Introduction

Typically, technology mapping employs graph or tree covering algorithms to map an And-Inverter Graph (AIG) into standard cells. However, technology mapping based on structural covering faces a challenge known as structural bias [1]. Namely, the quality of result (QoR) is largely dependent on the structure of subject graphs [2]. If the initial structure is poor, the final mapping will also be sub-optimal, even with the use of heuristics or iterative recovery methods.

To address the structural bias problem, previous research introduced lossless synthesis [3] to explore a few structural choices during mapping. It employs heuristic rewriting to produce functional equivalent nodes that are then combined into a single subject graph with choices, which is subsequently used to derive the mapped netlist and researches [1] [4] [5] targeted at mitigating structural bias heuristically, they cannot avoid this issue completely.

Recently, equality saturation has emerged as a promising technique, which uses a non-destructive rewriting approach to achieve better Pareto-optimal design space exploration and has been widely applied in the EDA field [6]. The applications include program optimization [7], compiling optimization [8], [9], datapath optimization [10], multiplier optimization [11], etc. Among these prior works, E-Syn [12] first applied e-graph rewriting to Boolean logic optimization and demonstrated the potential of e-graphs in exploring optimal logic structure. However, due to the complexity of e-graph-based optimization, these previous works have to be on a limited scale, with no more than 40,000 e-nodes [10]. When the graph scale significantly grows, the two main steps in equality saturation, which are rewriting (to expand solution space) and extraction (to pick a good solution), would inevitably suffer from excessively long runtime. Therefore, it is still an unanswered question whether e-graphs can play an effective role in large-scale Boolean logic optimization scenarios.

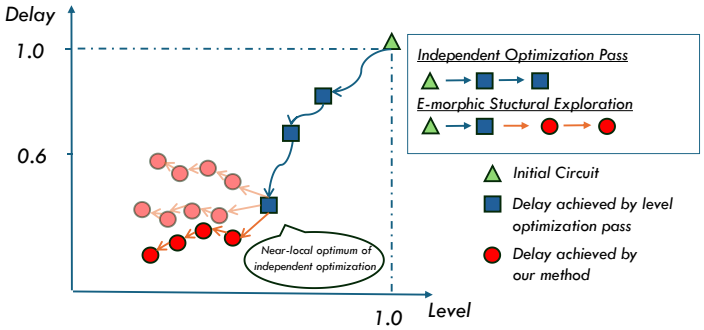


Fig. 1: E-morphic: parallel structural exploration for delay finetuning improves the circuit performance.

As we work to scale up equality saturation for logic synthesis, we acquire the following two key insights. (1) The method [1] [3] faces a primary challenge: because ABC inherently performs local rewriting [13], previous approaches do not maintain equivalence classes during the rewriting process. Instead, they create equivalence classes by detecting equivalent nodes through simulation and SAT checking across multiple circuits after performing local rewrites against the original circuit. This approach maintains only one additional equivalent node per equivalence class in the most commonly used ABC operator dch and has a restricted ability to make global structural changes. However, e-graphs utilize congruence closure-based [14] non-destructive rewriting, fundamentally aiming to maintain a vast

This work is supported in part by the National Natural Science Foundation of China under Grant No. 62304194, and by Guangzhou Municipal Science and Technology Project under Grant No. 2023A03J0013.

*Both authors contributed equally to this research.

*†Corresponding author.

number of equivalence classes. **By performing only a small number of iterations of e-graph-based rewriting, e-graphs can generate significantly more equivalence classes than previous methods, enabling larger structural transformations. Additionally, using fewer iterations ensures that independent optimization targets such as node sizes and logic levels are not adversely affected.** Through a case study presented in Figure 1, we observed that as technology-independent optimizations approach a near-local optimum, our tool, `E-morphic`, employs parallel e-graph-based resynthesis to mitigate structural bias issues. This approach facilitates the exploration of more optimal solutions for technology mapping targets. (2) Although fewer iterations already generate an excessive number of equivalence classes and nodes, the presence of too many equivalent nodes within each class severely impacts the time required for e-graph extraction. **Therefore, it is important to have an efficient and effective e-graph extraction method to explore various structural choices within a reasonable time. To this end, we incorporate a series of new features into the equality saturation framework, including solution space pruning and simulated annealing. Solution space pruning effectively reduces the over-abundance of redundant equivalent nodes in each equivalence class, significantly decreasing the time required for e-graph extraction while simulated annealing enables the escape from local minima when selecting from equivalent terms.**

Bearing these ideas in mind, we build an equality saturation framework named `E-morphic` for practical Boolean logic synthesis problems. It uses equality saturation in a fast and scalable manner to address the efficiency and scalability issues. The word "morphic" in its name suggests its intended role in logic synthesis — exploring diverse logic structures to improve the post-mapping design quality.

The contributions of this paper are as follows:

- To our best knowledge, this is the first work that adopts e-graph rewriting to address the structural bias issue of technology mapping in logic synthesis.
- It brings up several important updates to the equality saturation framework, such as DAG-to-DAG conversion, solution space pruning, simulated annealing, and multi-batch parallel computing for e-graph extraction. These techniques notable enhance the efficiency of equality saturation in logic synthesis and could be helpful for other e-graph applications.
- We implement a scalable and effective logic optimization framework and achieve 12.54% area saving and 7.29% delay reduction compared to a competitive delay-optimal synthesis flow on the large-scale circuits in the EPFL benchmark.

II. Background

A. E-Graph and Equality Saturation

An e(quivalence)-graph is a data structure used to represent a set of equivalent terms (expressions). Figure 2 gives an example of an e-graph where each green node is called an e-node that represents a term from a given language. The red-dotted boxes are equivalent classes (e-classes) where e-nodes in the same class represent equivalent expressions. E-graph also maintains congruence relations[1] over e-classes.

TABLE I: Examples of rewriting rules

Class	Pattern (LHS)	Transformation (RHS)
Commutativity	$a * b$	$b * a$
	$a + b$	$b + a$
Associativity	$(a * b) * c$	$a * (b * c)$
	$(a + b) + c$	$a + (b + c)$
Distributivity	$a * (b + c)$	$a * b + a * c$
	$(a + b) * (a + c)$	$a + (b * c)$
	$(a * b) + (a * c)$	$a * (b + c)$
Consensus	$(a * b) + ((\neg a) * c) + b * c$	$(a * b) + (\neg a) * c$
	$((a + b) * ((\neg a) + c)) * (b + c)$	$(a + b) * ((\neg a) + c)$
De-Morgan	$\neg(a * b)$	$\neg a + \neg b$
	$\neg(a + b)$	$(\neg a) * (\neg b)$

For a given input term, we may rewrite it into other equivalent forms following a set of rules (such as those listed in Table I). This process of finding equivalent terms is called equality saturation. Given a term $\mathcal{T}$, an initial e-graph $\mathcal{E}$ is constructed. Then we can iteratively find patterns that match with the left-hand-side of rewriting rules and record the right-hand-side patterns that will be added to the graph. Specifically, the rewriting does not remove the original term but only adds information to $\mathcal{E}$. Therefore, it is non-destructive and the ordering of rewriting will not affect the outcome. This solves the so-called phase ordering problem in optimization. The rewriting iterations may terminate when no more rule is applicable (namely, the graph is saturated) or when a certain exiting condition is met. Then an optimized term can be extracted from the final e-graph where the optimality of terms is measured by a cost function defined w.r.t. the whole graph.

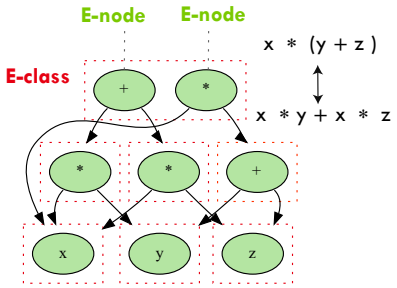


Fig. 2: This e-graph represents two equivalent expressions. Each green node is an e-node, and each red-dotted box is an e-class. Edges connect e-nodes to child e-classes.

B. E-Graph Rewriting in Logic Synthesis

E-graph has demonstrated its potential in many optimization problems [8]–[11], [15], while the prior work `E-Syn` [12] first applied equality saturation in logic synthesis. `E-Syn` is based on an existing e-graph processing framework `egg` [16], as presented by Figure 3. Its workflow is as follows: ❶ **Preprocessing:** `E-Syn` takes advantage of the ABC logic synthesis tool to transform input Verilog or AIG into the equation format. Then `E-Syn` converts the equation format into the S-expression, which is a data structure for nested lists commonly used in Lisp-like programming languages and can be converted into e-graphs using the built-in functionality of `egg`. ❷ **Rewriting:** The `egg`

[1] If two terms a and b are equivalent, any function applications over a and b would also be equivalent ($f(a) \equiv f(b)$).

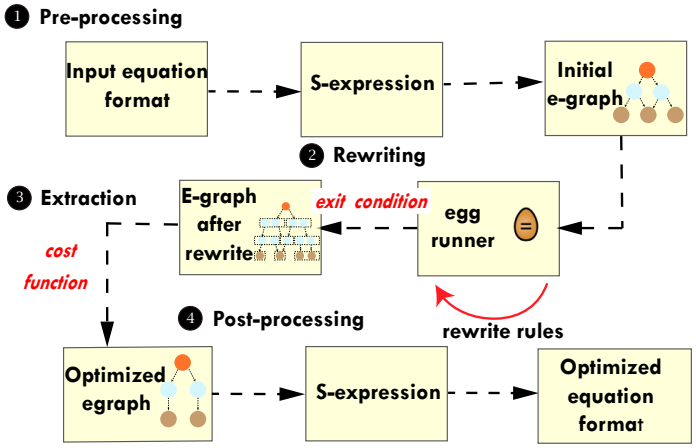


Fig. 3: The workflow of E-Syn [12]

runner will iteratively apply a set of rewriting rules until the e-graph is saturated or an exit condition is met, such as when the time limit or maximum iteration is reached. ❸ **Extraction:** It selects the best term based on a cost function. To achieve overall algorithmic efficiency, a commonly used approach is to traverse the e-graph bottom-up and greedily select the e-node with the lowest cost among the equivalent ones in the same e-class. ❹ **Post-processing:** E-Syn finally converts the optimized e-graph back into the equation format, which can be mapped to gate-level circuits using synthesis tools. Then the performance metrics of the circuit will be reported.

III. Methodology

E-morphic effectively addresses several significant challenges associated with previous works [12], [17], which struggles with problems like inefficient intermediate representations and restricted cost functions, leading to scalability issues. Our solution overcomes these limitations through critical advancements, which we will discuss in the following subsections.

A. E-morphic Overview

Our framework, E-morphic, introduces a novel and efficient method for applying e-graphs in logic synthesis. We identify a unique role for e-graph-based optimization, between technology-independent optimization and technology mapping. E-morphic offers several key advantages:

1) **Structural Exploration before Technology Mapping:** Parallel E-graph-based resynthesis using fewer rewriting iterations after technology-independent optimization enables efficient structural exploration to address the structural bias issue.
2) **Novel Extraction Method:** Integrating Solution Space Pruning and Simulated Annealing-Based Algorithms with Multi-Threaded Parallel Computing for Efficient E-graph Extraction and Faster Evaluation.
3) **Efficient Implementation:** Our framework incorporates several efficiency-enhancing features: (a) direct DAG-to-DAG conversion between circuits and e-graphs, eliminating the overhead of intermediate S-expressions, and (b) machine learning-based cost estimation for rapid quality assessment.

Figure 5 illustrates the workflow of E-morphic. The input circuit first undergoes conventional technology-independent optimization (❶). Our efficient DAG-to-DAG conversion (❷) then initializes the e-graph. The highlight of our approach lies in the extraction and evaluation phase (❸), with our novel simulated annealing algorithm, parallel processing, and dual cost evaluation methods (ABC-based quality-prioritized and GNN-based rapid estimation).

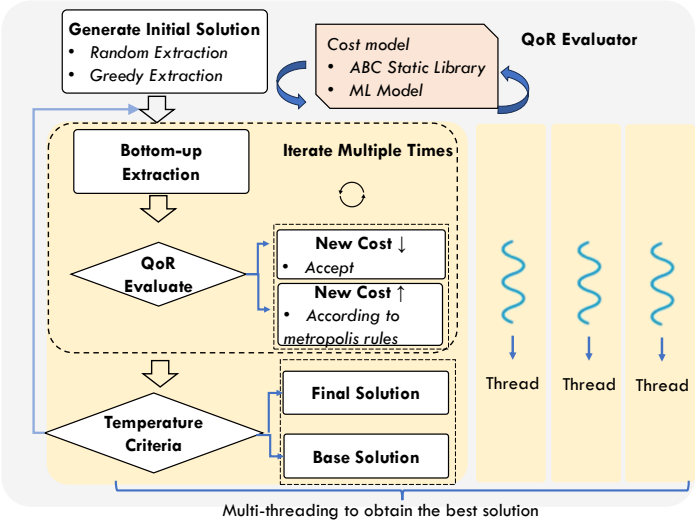


Fig. 4: Simulated-annealing-based extraction algorithm.

B. Efficient Extraction methods

1) Simulated Annealing e-graph Extraction

The extraction phase in e-graph-based optimization is crucial for obtaining high-quality results. However, extraction from equality saturated graphs is known to be NP-hard [18], and traditional greedy algorithms often fail to reflect QoR after circuit technology mapping. To address this challenge, we have developed a novel extraction algorithm based on simulated annealing (SA) [19], a probabilistic technique for approximating the global optimum of a given function. In addition, we integrate Solution Space Pruning and Multi-Threaded Parallel Computing to achieve efficient e-graph extraction and facilitate broad structural exploration.

Our SA extractor combines the principles of bottom-up extraction with simulated annealing to efficiently explore the solution space of an e-graph. This hybrid approach aims to overcome local optima and find high-quality circuit representations. The algorithm (illustrated in Figure 4) operates as follows:

We begin by generating an initial solution using a basic extraction method, such as greedy depth-minimization or random selection. The main loop of the algorithm then iteratively improves this solution while gradually decreasing the temperature parameter of simulated annealing, with specific cooling schedules detailed in Section IV-A. At each iteration, we generate a neighboring solution, evaluate its cost, and decide whether to accept or reject it based on the cost difference and current temperature. This process allows for occasional uphill moves, enabling the algorithm to escape local optima.

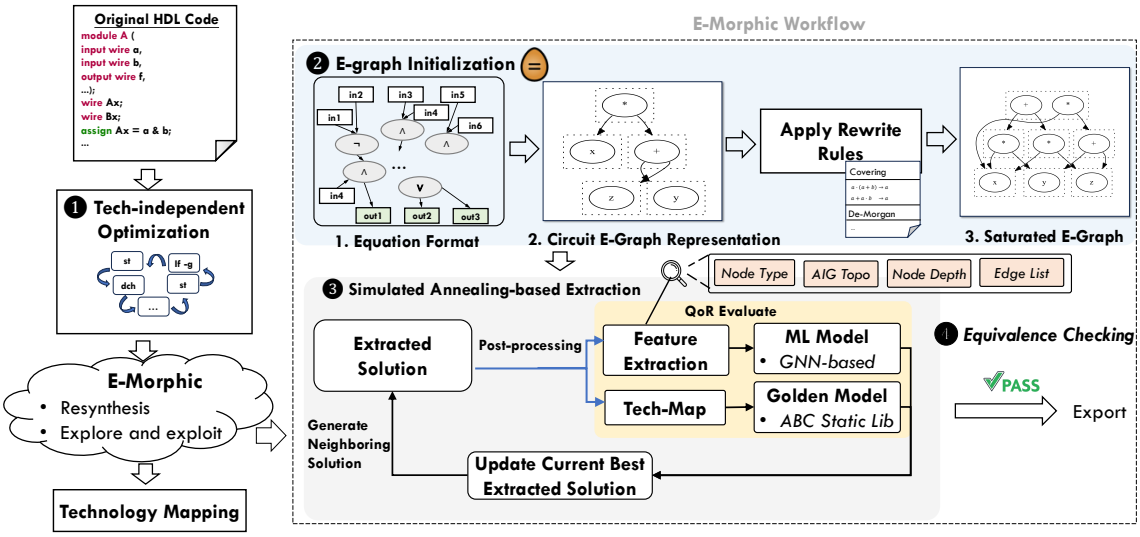


Fig. 5: E-morphic overview.

The neighbor generation process, illustrated in Algorithm 1, starts from leaf nodes and propagates changes upwards through the e-graph. For each node, we decide whether to update its selection based on a combination of cost improvement and random chance. This randomness allows for exploration of different parts of the solution space while ensuring that the solution quality does not degrade below the initial solution.

Algorithm 1: Generate Neighboring Solution

Input: Current solution $\mathcal{O}_{current}$, e-graph $\mathcal{E}$, cost function $\mathcal{C}$, random probability p_{random}
Output: New solution $\mathcal{O}_{new}$

$\mathcal{O}_{new} \leftarrow \mathcal{O}_{current}$;
$traversalQueue \leftarrow$ LeafNodes($\mathcal{E}$);
$Costs_map \leftarrow \{ \infty$, for all e-classes$\}$;
while *$traversalQueue$ is not empty* **do**
  $enode \leftarrow traversalQueue$.pop();
  $class \leftarrow$ GetClass($enode$);
  $prev_cost \leftarrow Costs_map[eclass]$;
  $CS \leftarrow$ all children of $enode$;
  if *$\mathcal{C}$ is a "sum cost"* **then**
    $new_cost \leftarrow C(enode) + \sum_{i \in CS} Costs_map[i]$;
  end
  if *$\mathcal{C}$ is a "depth cost"* **then**
    $new_cost \leftarrow C(enode) + \max_{i \in CS}\{Costs_map[i]\}$;
  end
  if $prev_cost = \infty$ ***or*** $(new_cost < prev_cost$ ***and*** $random() \geq p_{random})$ **then**
    $\mathcal{O}_{new}$.choose($eclass$, $enode$);
    $Costs_map[eclass] \leftarrow new_cost$;
    $traversalQueue$.extend(GetParents($eclass$));
  end
end
return $\mathcal{O}_{new}$;

2) Solution Space Pruning for Extraction

Additionally, we employ solution space pruning techniques to accelerate the extraction process. Traditional bottom-up extraction algorithms often traverse every e-node in the e-graph, recalculating costs even when changes are unlikely to yield improvements. In contrast, our method maintains a traversal queue in Algorithm 1 that only records the e-nodes with costs less than or equal to the minimum cost in each e-class, effectively skipping the remaining e-nodes with higher costs, thereby significantly filtering out the numerous redundant e-nodes introduced in each e-class due to commutative and associative laws as shown in Figure 6. In addition, the minimum cost of each e-class is maintained by a hashmap called $Costs_map$, which caches the optimal costs per e-class. This approach avoids redundant computations by preventing the re-evaluation of e-node costs when the costs of their child nodes remain unchanged.

By combining these techniques, we achieve a more efficient and effective extraction process that is particularly well suited for large and complex e-graphs encountered in large circuit optimization scenarios.

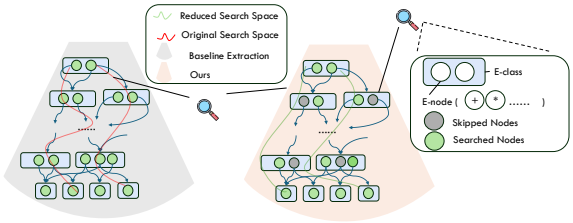


Fig. 6: Solution space pruning.

3) Multithreaded Parallel Cost Evaluation

In the QOR evaluation phase of our simulated annealing algorithm as referenced in Figure 4, we support multi-threading to enhance efficiency. Leveraging initial solutions derived from bottom-up extraction, we concurrently run multiple annealing processes across different threads. This parallel execution allows us to perform QOR fine-tuning on multiple solution sets simultaneously, facilitating extensive resynthesis for a variety of structural transformations. Ultimately, we map all the parallel-generated solutions and select the one with the best QOR results.

C. Dual-Model Approach for Cost Estimation

To balance optimization speed and result quality, we employ two complementary cost models in our extraction process:

1) **Runtime-prioritized mode:** This mode primarily uses a Graph Neural Network (GNN) model for rapid estimation. The GNN quickly evaluates circuit characteristics based on node type, topology order, and connection relationships. While less accurate, it provides fast feedback to guide the simulated annealing process, making it particularly useful for exploring many candidate solutions.
2) **Quality-prioritized mode:** This mode performs a fast but rough mapping to obtain a QoR estimation. It converts the extracted circuit to an equation file, processes it using `ABC` with a standard cell library, and applies technology mapping and optimizations. The resulting circuit delay serves as the primary cost metric, ensuring high-quality evaluations at the cost of longer runtime.

By combining these modes, our approach efficiently navigates the complex solution space of e-graphs, balancing speed and accuracy in cost estimation for various circuit optimization tasks.

D. Efficient Implementation

Building upon our novel approach for structural exploration and extraction, we have developed several key techniques to enhance the efficiency of our framework. These improvements address the major bottlenecks in previous e-graph-based logic synthesis methods, allowing `E-morphic` to handle larger and more complex circuits.

1) Direct DAG-to-DAG Conversion

Previous work in e-graph-based logic synthesis [12] relied on S-expressions as the intermediate representation between circuits and e-graphs. However, S-expressions, being nested lists derived from flattening abstract syntax trees (ASTs), are not ideal for representing circuit graphs. Shared nodes must be duplicated in S-expressions, leading to significant space and time overhead during circuit-to-e-graph conversion and potentially degrading the overall quality of results (QoR). To address this limitation, we have developed a more efficient intermediate format that serves as a bridge for conversion. It eliminates the need for complex parsing and reconstruction steps, significantly reducing computational overhead as illustrated in Figure 8. This format is essentially a serialization of the initial e-graph, using unique identifiers to represent nodes and their relationships. Figure 7 provides an example of this representation. For an equation format input, each assignment corresponds to a node in the initial e-graph, which is referred to by an `id` in our DSL. The one-to-one correspondence between circuit elements and their e-graph counterparts avoids the exponential growth in representation size of S-expressions.

2) Flexible Cost Model Integration

Our framework provides an interface for integrating various cost models, including machine learning-based approaches. We don't limit on the specifics of the particular model, this flexibility allows us to leverage state-of-the-art techniques for rapid quality assessment. Later, Section IV-D will demonstrate the use of a GNN model to provide rapid cost estimation to guide the simulated annealing process.

```
"egraph" : {
 "3": {"id": 3, "nodes": [{"Symbol": "a" }],"parents": [7,8]},
 ... // node id:(b, id:4), (c, id: 5)
 "7": {"id": 7, "nodes": [{"AND": [3,4]}],  "parents": [6,9]},
 "8": {"id": 8, "nodes": [{"AND": [3,5]}],  "parents": [9]},
 ... } // other nodes
```

Fig. 7: The intermediate format for efficient e-graph—circuit conversion in pre- and post-processing.

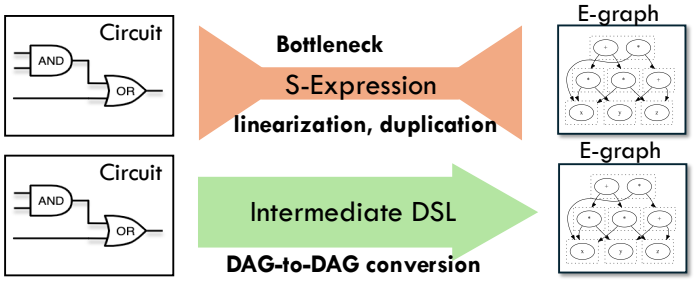


Fig. 8: Prior and our new e-graph–circuit conversion.

By combining these efficiency-enhancing techniques, `E-morphic` achieves significant speed-ups, enabling the application of e-graph-based optimization to a wider range of circuit sizes and complexities.

IV. Experiments

A. Experiment Setting

We implement the `E-morphic` framework in `Rust` to interface with the existing `egg` library for e-graphs. All experiments are performed on a Ubuntu 20.04.4 LTS server with dual Intel Xeon Platinum 8375C processors and 256GB memory. The test circuits all come from the EPFL benchmark suite [20], containing 2 small-, 2 medium- and 6 large-scale cases. Throughout these experiments, we use ASAP $7nm$ [21] technology library to evaluate the post-mapping QoR. E-graph rewriting takes 5 iterations, which already produces a substantial number of equivalence classes to facilitate structural exploration. In the quality-prioritized mode, we utilize 4 threads, with the annealing exit condition set to 4 iterations. In the first iteration, the temperature T is set to 2000 as a high temperature T_1 encourages acceptance of inferior solutions, helping to avoid local optima. For the 2nd and 3rd iterations, the temperature is defined as $T_n = T_{n-1} \times \frac{|\text{new cost} - \text{old cost}|}{n \times 10000}$ (n is the number of iterations), ensuring progression into the pseudogreedy local-search stage. In the 4th iteration, the temperature is adjusted to $T_n = T_{n-1} \times \frac{|\text{new cost} - \text{old cost}|}{n}$, allowing for further enhancements in solution quality. For the runtime-prioritized mode, we utilize 6 threads to balance the performance loss of the cost model. All the circuits optimized through `E-morphic` are verified for equivalence using the `cec` command in `ABC`.

B. Efficient DAG-to-DAG Conversion

One bottleneck of the prior work `E-Syn` is the conversion between circuit and e-graphs. Here we compare the circuit–e-graph mutual conversion time using the benchmark circuits. As can be seen from Table III, in most cases, our direct DAG-to-DAG conversion takes only a fraction of a second. Whereas,

TABLE II: QoR and runtime comparison between E-morphic and the baseline.

Circuit	*SOP Balancing Baseline*				*SOP Balancing + E-morphic w/o ML model*				*SOP Balancing + E-morphic w ML model*			
	Area (μm^2)	Delay (ps)	lev	runtime (s)	Area (μm^2)	Delay (ps)	lev	runtime (s)	Area (μm^2)	Delay (ps)	lev	runtime (s)
hyp	370305.16	253779.06	10929	1496.23	386483.84	**240435.12**	10948	2102.52	**366751.38**	**248781.94**	11033	1646.86
div	56873.2	25378.59	1258	108.04	**34475.98**	**24747.77**	1300	280.79	**48386.24**	**24333.65**	1298	147.77
mem_ctrl	29433.64	1281.99	61	76.5	**29219.49**	**1056.09**	**59**	113.17	**29365.29**	**1081.48**	**60**	97.09
log2	41154.32	5233.86	231	161.49	**32840.69**	**5033.46**	240	289.79	41234.11	**5153.74**	233	208.30
multiplier	35799.38	2632.08	147	54.68	**31443.34**	**2371.24**	**130**	105.02	**35002.03**	**2599.37**	147	87.15
sqrt	53262.02	78627.99	3574	112.27	**33852.66**	**72796.83**	3578	270.21	**48909.25**	**76432.02**	3579	143.43
square	21240.14	834.01	69	25.12	21418.84	**792.09**	69	50.22	21532.44	**794.16**	70	37.46
arbiter	4464.75	288.18	23	12.58	4587.45	**270.98**	**22**	23.34	4532.63	**284.27**	24	20.18
sin	16052.7	3712.41	119	34.95	**15737.77**	**3567.12**	**113**	62.31	**15737.77**	**3685.34**	**118**	50.83
adder	1206.99	584.53	57	1.96	1282.11	**536.42**	57	10.2	1285.61	**546.86**	57	7.07
GEOMEAN	25274.02	5620.01	292	60.32	**22104.32**	**5210.55**	**287**	126.25	**24660.84**	**5390.13**	295	91.42
Improvements					**12.54%**	**7.29%**			**2.42%**	**4.09%**		

TABLE III: Comparison of e-graph–circuit conversion with a time-limit of 3600 seconds and 8 GB memory-limit. "Forward" means circuit to e-graph conversion and "backward" refers to e-graph to circuit conversion. TO represents "timeout." MO stands for "out-of-memory."

Design		**E-Syn [12]**		**E-morphic (this work)**	
Name	**#. e-node**	**Forward** (s)	**Backward** (s)	**Forward** (s)	**Backward** (s)
hyp	420897	TO & MO	N.A.*	**8.2**	**4.6**
div	101860	TO & MO	N.A.*	**1.57**	**1.2**
mem_ctrl	84701	TO & MO	N.A.*	**1.44**	**0.8**
log2	54532	TO & MO	N.A.*	**0.93**	**0.71**
multiplier	50761	TO	N.A.*	**0.84**	**0.75**
sqrt	41234	TO	N.A.*	**0.70**	**0.53**
square	35685	TO	N.A.*	**0.59**	**0.55**
arbiter	23619	59.53	65.56	**0.38**	**0.12**
sin	8948	TO	N.A.*	**0.15**	**0.13**
adder	2548	4.25	53.4	**0.04**	**0.04**
GEOMEAN	-	-	-	**0.65**	**0.46**

* Not available due to the prior e-graph to circuit conversion failure.

E-Syn hardly scales to over 10^4 e-nodes, resulting in time-out in most cases. The results indicate that our conversion method is minimally affected by the circuit size. Even when applied to circuits over 10^5 e-nodes, the conversion can still be performed in a remarkably short runtime.

C. Enhancing Conventional Delay-oriented Synthesis

As delay-oriented optimization is crucial to meet timing constraints in logic synthesis, in this experiment, we compare our approach against a competitive delay-oriented logic optimization flow published in [22], which has been adopted by industrial users. The optimization sequence is as follows: ***(st; if -g -K 6 -C 8)(st; dch; map)***^{4}. It takes advantage of SOP balancing [22], priority-cut-based mapping [23] and resynthesis after mapping [1]. We argue that it is nontrivial to outperform this flow with further optimization. For comparison, the E-morphic flow applies the same ***(st; if -g -K 6 -C 8)(st; dch; map)***^{3} first. Before the final round of ***(st; dch; map)***, we make use of e-graph rewriting to obtain a set of structurally diverse solutions. The results are shown in Table II. Compared to the conventional delay-oriented optimization flow, E-morphic achieves delay reduction on all designs, averaging 7.29%, along with an area saving of nearly 12.54% while the runtime overhead is moderate. Due to space limit, we omit the result of E-Syn [12] as it already times out for all large-scale circuits in this benchmark.

D. GNN-based Cost Model for Faster Extraction

To further reduce the runtime overhead of E-morphic, one can plug in a machine learning model for rapid cost evaluation. Here we demonstrate the use of state-of-the-art HOGA [24] model. We firstly train it on circuits from the OpenABC-D benchmark [25] synthesized by the aforementioned flow to adapt its prediction towards the setting in this experiment. Each design module in the OpenABC-D benchmark generates 100 different structural samples, resulting in a training set of around 40,000 samples. The labels are obtained using the same ASAP 7nm cell library for technology mapping. The delay prediction achieves a Mean Absolute Percentage Error (MAPE) of 25.2% and a Kendall's τ of 0.62. Using this prediction model, we present logic synthesis results in the last column of Table II, which shows an averaged runtime saving of nearly 28%.

E. Runtime Analysis

We plot the proportion of time consumption of each step in Figure 9. It can be seen that the time spent on the existing technology-independent optimization and mapping in the delay-oriented flow [22] accounts for a large portion of the total runtime, which is irrelevant to the usage of e-graph optimization. While the time for the additional operations in E-morphic (mainly conversion and simulated annealing) is moderate, demonstrating the efficiency of our framework. Particularly, this efficiency is more pronounced in large-scale circuits, where the proportion of our time overhead is significantly lower.

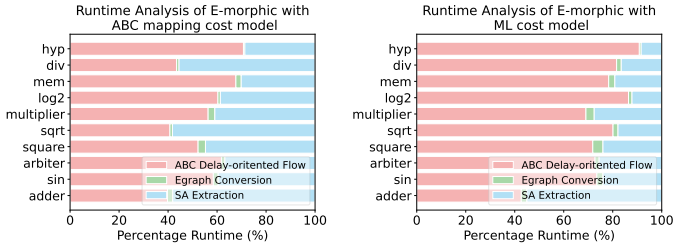


Fig. 9: Runtime breakdown of E-morphic logic synthesis

V. Conclusion

This paper proposes a scalable framework named E-morphic, which uses equality saturation for structure exploration before technology mapping. It features efficient e-graph-circuit conversion, solution-space pruning and simulated annealing for extraction. Additionally, it allows the integration of ML prediction models to further reduce the runtime. The techniques presented in this paper will not only benefit applications of equality saturation in logic synthesis, but also other e-graph-based large-scale optimization problems.

REFERENCES

[1] S. Chatterjee, A. Mishchenko, R. Brayton, X. Wang, and T. Kam. Reducing structural bias in technology mapping. In *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, pages 519–526, 2005.

[2] Eric Lehman, Yosinori Watanabe, Joel Grodstein, and Heather Harkness. Logic decomposition during technology mapping. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):813–834, 1997.

[3] Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. Improvements to technology mapping for lut-based fpgas. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 41–49, 2006.

[4] Leon Stok, Andrew J. Sullivan, and Mahesh A. Iyer. Wavefront technology mapping. In *1999 Design, Automation and Test in Europe (DATE '99), 9-12 March 1999, Munich, Germany*, page 531. IEEE Computer Society / ACM, 1999.

[5] Eric Lehman, Yosinori Watanabe, Joel Grodstein, and Heather Harkness. Logic decomposition during technology mapping. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 16(8):813–834, 1997.

[6] Ecenur Ustun, Cunxi Yu, and Zhiru Zhang. Equality saturation for datapath synthesis: A pathway to pareto optimality. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–2. IEEE, 2023.

[7] Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. Equality saturation: A new approach to optimization. In *POPL*, page 264–276, New York, NY, USA, jan 2009. ACM.

[8] Jianyi Cheng, Samuel Coward, Lorenzo Chelini, Rafael Barbalho, and Theo Drane. SEER: Super-optimization explorer for HLS using e-graph rewriting with MLIR. *ASPLOS*, 2024.

[9] Yan Pi, Hongji Zou, Tun Li, Wanxia Qu, and Hai Wan. ESFO: Equality saturation for FIRRTL optimization. In *GLSVLSI*, 2023.

[10] Samuel Coward, George A Constantinides, and Theo Drane. Automating constraint-aware datapath optimization using e-graphs. In *DAC*, 2023.

[11] Andy Wanna, Samuel Coward, Theo Drane, George A. Constantinides, and Milos D. Ercegovac. Multiplier optimization via e-graph rewriting. *ACSSC*, 2023.

[12] Chen Chen, Guangyu Hu, Dongsheng Zuo, Cunxi Yu, Yuzhe Ma, and Hongce Zhang. E-Syn: E-graph rewriting with technology-aware cost functions for logic synthesis. In *DAC*, 2024.

[13] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. Dag-aware AIG rewriting a fresh look at combinational logic synthesis. In Ellen Sentovich, editor, *Proceedings of the 43rd Design Automation Conference, DAC 2006, San Francisco, CA, USA, July 24-28, 2006*, pages 532–535. ACM, 2006.

[14] Aaron R Bradley and Zohar Manna. *The calculus of computation: decision procedures with applications to verification*. Springer Science & Business Media, 2007.

[15] Ecenur Ustun, Ismail San, Jiaqi Yin, Cunxi Yu, and Zhiru Zhang. Impress: Large integer multiplication expression rewriting for FPGA HLS. In *FCCM*, pages 1–10. IEEE, 2022.

[16] Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *POPL*, 2021.

[17] Samuel Coward, George A Constantinides, and Theo Drane. Automatic datapath optimization using e-graphs. In *ARITH*, pages 43–50. IEEE, 2022.

[18] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[19] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[20] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *IWLS*, number CONF, 2015.

[21] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. ASAP7: A 7-nm FinFET predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

[22] Alan Mishchenko, Robert Brayton, Stephen Jang, and Victor Kravets. Delay optimization using sop balancing. In *Proceedings of the International Conference on Computer-Aided Design*, ICCAD '11, page 375–382. IEEE Press, 2011.

[23] Alan Mishchenko, Sungmin Cho, Satrajit Chatterjee, and Robert Brayton. Combinational and sequential mapping with priority cuts. In *ICCAD*, pages 354–361. IEEE, 2007.

[24] Chenhui Deng, Zichao Yue, Cunxi Yu, Gokce Sarar, Ryan Carey, Rajeev Jain, and Zhiru Zhang. Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits. In *DAC*, 2024.

[25] Animesh Basak Chowdhury, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Openabc-d: A large-scale dataset for machine learning guided integrated circuit synthesis, 2021.

Holistic Design towards Resource-Stringent Binary Vector Symbolic Architecture

Shijin Duan[1], Nuntipat Narkthong[1], Yukui Luo[2], Shaolei Ren[3], Xiaolin Xu[1]
[1]*Northeastern University*, [2]*Binghamton University*, [3]*University of California, Riverside*
{duan.s, narkthong.n, x.xu}@northeastern.edu, yluo11@binghamton.edu, shaolei@ucr.edu

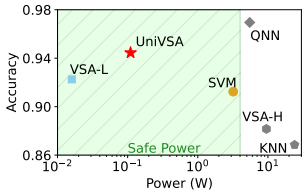
***Abstract*—Classification tasks on ultra-lightweight devices demand devices that are resource-constrained and deliver swift responses. Binary Vector Symbolic Architecture (VSA) is a promising approach due to its minimal memory requirements and fast execution times compared to traditional machine learning (ML) methods. Nonetheless, binary VSA's practicality is limited by its inferior inference performance and a design that prioritizes algorithmic over hardware optimization. This paper introduces UniVSA, a co-optimized binary VSA framework for both algorithm and hardware. UniVSA not only significantly enhances inference accuracy beyond current state-of-the-art binary VSA models but also reduces memory footprints. It incorporates novel, lightweight modules and design flow tailored for optimal hardware performance. Experimental results show that UniVSA surpasses traditional ML methods in terms of performance on resource-limited devices, achieving smaller memory usage, lower latency, reduced resource demand, and decreased power consumption.**

I. Introduction

TinyML frameworks are essential for edge devices, aiming for low latency, minimal hardware overhead, and high throughput [1]–[3], particularly in brain-computer interfaces (BCIs). Traditional ML like SVMs [4] and linear classifiers [5] are still favored for BCI due to their lightweight implementations, while in modern ML, even the binary neural networks (BNNs) fail to meet the power constraints of implanted BCI devices [4], [6]. Thus, there's a vital need for ML models that are both **lightweight** and **performant** for advancing tinyML applications on resource-stringent devices. We contend that the emerging binary vector symbolic architecture (VSA) possesses these characteristics. Binary VSA encodes objects and values as binary vectors and executes operations through logical operations, significantly enhancing performance on resource-limited devices [7]–[10].

Currently, the state-of-the-art (SOTA) training approach for lightweight binary VSA is low-dimensional computing (LDC) [11]. Binary VSA models optimized by LDC strategy can have vector dimension $D \approx 100$, with only minimal loss in inference accuracy compared with SOTA high-dimensional models with $D = 10,000$ [12]. The low-dimension binary VSA results in kilobyte-scale model sizes and power consumption within safe limits on BCIs, laying the groundwork for applying binary VSA models in resource-stringent environments.

However, current binary VSA implementations still face other challenges in resource-constrained devices such as BCIs. Specifically, binary VSA models could underperform compared to traditional ML approaches in certain BCI tasks. For instance, an LDC-trained binary VSA model achieves an accuracy 5% lower than that of an SVM for the EEGMMI task (Table II). Additionally, there is a notable saturation in accuracy with increasing vector dimensions, such as around $D = 128$ in [11]. Innovations in binary VSA design are essential to overcome these accuracy constraints within power limitations. Furthermore, past binary VSA hardware implementations are often developed separately from the model design, as independent accelerators [9], [11], without considering the hardware overhead and its suitability for the intended application. A binary VSA designed solely with performance under consideration may be suboptimal or even unsuitable for specific hardware, especially with severe resource constraints.



	Memory(KB)	Latency(ms)
SVM	$\times 10^3 - 10^4$	$\times 10^2$
KNN	–	$\times 10^2$
QNN	$\times 10^3 - 10^4$	$\times 10^2$
VSA-H	$\times 10^3 - 10^4$	$\times 10^{-2}$
VSA-L	$\times 1 - 10$	$\times 10^{-3}$
UniVSA	$\times \mathbf{1 - 10}$	$\times \mathbf{10^{-2}}$

Fig. 1: Comparison between UniVSA, high-dimensional VSA (noted as VSA-H [9], [12]), low-dimensional VSA (LDC [11]), and other lightweight ML models (including QNN [13], BNN [14], SVM [15], and KNN [16]).

In this work, we address crucial challenges by proposing a universal binary VSA framework, namely *UniVSA*, highlighting its underlying algorithm-hardware co-optimization mechanisms. Importantly, our approach does not only involve algorithmic optimization to define an improved binary VSA model, but also specifies the hardware implementation based on the well-trained model. On the algorithm side, we emphasize a previously overlooked issue in binary VSA — the absence of interaction between features when generating vector representations. To remedy this, we introduce a binary feature extraction module that establishes relationships between features. Additionally, we enrich the binary VSA models with other efficient designs. In terms of hardware, we propose modules for UniVSA which consist of essential primitives and a controller for execution scheduling. In Fig. 1, we highlight the general comparison between UniVSA and other lightweight ML methods.

Our main contributions are three-fold as summarized below:

- **Algorithm-optimization: Model Design.** We propose an improved binary VSA model in our UniVSA framework. This model has more capability and flexibility in the

configuration to improve the inference accuracy.

- **Hardware-optimization: Implementation.** We specify modules to accelerate UniVSA. Implementing UniVSA employs parallelization and pipelining within critical modules to minimize hardware overhead. Meanwhile, resource usage is carefully managed to facilitate parallelism for reduced latency while preventing excessive power consumption.
- **Evaluation:** We evaluate UniVSA compared with other lightweight ML methods under resource-stringent scenarios. These results validate the algorithmic optimization of UniVSA, and demonstrate the lightweight and high efficiency of UniVSA hardware implementation.

II. Preliminaries

A. Binary Vector Symbolic Architecture

For a sample x with N features and M available values for each feature, binary VSA generates bipolar vector sets $\mathbf{F} = \{\boldsymbol{f}_i\} \in \{-1, 1\}^{N \times D}, i = [1, N]$ and $\mathbf{V} = \{\boldsymbol{v}_m\} \in \{-1, 1\}^{M \times D}$, $m = [1, M]$ to represent feature positions and values, respectively. Note for continuous values, they are discretized into M intervals to suit the VSA computing. The sample x is encoded as a bipolar vector $\boldsymbol{s}$ as

$$\boldsymbol{s} = \text{sgn}\left(\sum_{i=1}^{N} \boldsymbol{f}_i \circ \boldsymbol{v}_{x_i}\right) \tag{1}$$

where $\text{sgn}(\cdot)$ is the sign function to binarize the accumulation result. We set $\text{sgn}(0) = 1$ as a tiebreaker. A toy example of binary VSA encoding is illustrated in the upper part of Fig. 2, explaining how a sample is encoded as a bipolar vector.

B. Binary VSA on Classification Tasks

For a classification task, binary VSA generates a bipolar vector for each category (i.e., class). Assuming that there are C classes, the training of a binary VSA model is to derive a class vector set $\mathbf{C} = \{\boldsymbol{c}\} \in \{-1, 1\}^{C \times D}$. During inference, a query sample is encoded as binary vector $\boldsymbol{s}$ using Eq. 1. Then, the query sample vector will be compared with all class vectors, for which the one with the highest similarity will be determined as the predicted label:

$$\text{label} = \arg\max_{j} \ \text{SIM}(\boldsymbol{s}, \boldsymbol{c}_j) \tag{2}$$

$\text{SIM}(\cdot)$ can be any similarity measurements, where Hamming distance and dot-product are commonly used in binary VSA. In summary, binary VSA behaves as follows for a query input x:

$$x \xrightarrow{x_i \to \boldsymbol{v}_{x_i}} [\boldsymbol{v}_{x_1}, ..., \boldsymbol{v}_{x_N}] \xrightarrow{\text{sgn}(\sum \boldsymbol{f} \circ \boldsymbol{v})} \boldsymbol{s} \xrightarrow{\arg\max \mathbf{C}\boldsymbol{s}} \text{label} \tag{3}$$

The lower part of Fig. 2 demonstrates an example of the dot-product metric as the similarity measurement.

C. Low-Dimensional Training on Binary VSA

Considering the necessity of low dimension to implement binary VSA on resource-constrained devices, we explain the low-dimensional computing (LDC) strategy [11] for binary VSA training. LDC maps the binary VSA model (i.e., Eq. 1 and Eq. 2) onto a partial BNN, which is then optimized for vector generation:

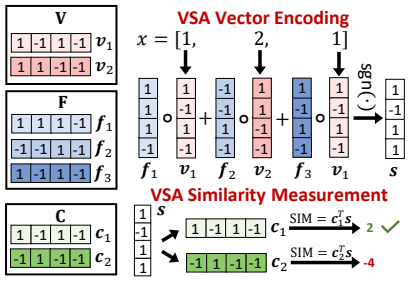


Fig. 2: Encoding x using binary VSA (upper) and measuring similarity between x and stored class vectors $\mathbf{C}$ (lower). Assume x has three features and two available values, thus $N = 3, M = 2$; there are two categories $C = 2$.

Value Projection. A *unique* neural network, namely *ValueBox* (VB), is proposed to project an feature value x_i to a bipolar vector $\boldsymbol{v}_{x_i}$, mimicking the look-up step "$x_i \to \boldsymbol{v}_{x_i}$" in Eq. 3. VB is composed of multi-layer perceptron (MLP) followed by binarization, i.e., $\boldsymbol{v}_x = \text{VB}(x) = \text{sgn}(\text{MLP}(x))$. During binary VSA implementation, $\mathbf{V} = \{\boldsymbol{v}_m\}$ is collected by evaluating all available values $m = [1, M]$ through $\text{VB}(m)$.

Vector Encoding. LDC interprets the vector encoding (i.e., the second step in Eq. 3) as a specially structured binary layer. In this view, $\boldsymbol{v}_x$ are regarded as the layer input, and $\mathbf{F} = \{\boldsymbol{f}_i\}$ are the binary weights.

Similarity Measurement. The equivalence of the Hamming distance and the dot-product is proved in [11]. Therefore, the similarity measurement can be achieved with a binary dense layer (at the third step in Eq. 3), where the binary weights correspond to the class vector set $\mathbf{C}$.

After optimizing this partial BNN, the feature vector set $\mathbf{F}$ and class vector set $\mathbf{C}$ are directly extracted from the binary layers, while the value vector set $\mathbf{V}$ is derived from VB. During inference, only $\mathbf{V}$, $\mathbf{F}$, and $\mathbf{C}$ are required in VSA process (Eq. 1 and 2); partial BNN is only utilized for training. Although LDC offers a strategy to reduce vector dimensions, binary VSA still falls short in resource-constrained applications due to *performance constraints* and *undeveloped design flow*.

III. UniVSA: Model Design

For resource-stringent scenarios, binary VSA should meet two critical requirements: ***ultra-lightweight*** (i.e., low dimension) and ***high accuracy***. We extend the binary VSA model in terms of these goals, with three-fold enhancements: ① Discriminated Value Projection – the features with less importance for classification can be represented with fewer elements; ② Binary Feature Extraction – a binary convolutional layer is introduced to construct interactions between features; ③ Soft Voting – ensemble strategy is embedded. We demonstrate the overview of our UniVSA model in Fig. 3.

A. Binary VSA Model under UniVSA Framework

We take the electrocorticography (ECoG) signal [17] as an example for the model input, which is one typical format

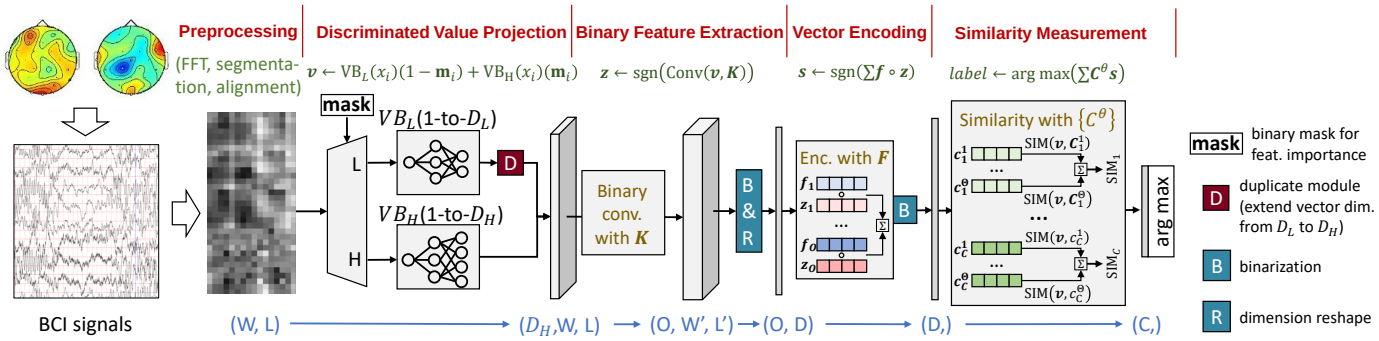


Fig. 3: The UniVSA model design. Symbols in parentheses show the vector/matrix dimensions for each stage.

of implanted BCI signals. Specifically, it is preprocessed and evenly divided into W sliding windows with overlap, where each window contains a signal snippet of length L [4], i.e., the input is shaped as (W, L), in total $N = W \times L$ features.

1) Discriminated Value Projection: Current binary VSA models consider all input features with the same importance. However, some time or frequency intervals of an ECoG signal are simply irrelevant or noisy information. We define a feature's importance as whether it has obvious impacts on accuracy for classification. Specifically, an input-wise binary **mask** is generated through the feature subset selection strategy [18]. The features corresponding to 1 in **mask** represent high importance, while others with 0 are low-importance. The input passes through a discriminator defined by **mask**, where values under high-importance features go to VB_H and under low-importance ones go to VB_L (in Fig. 3). Specifically, VB_L produces bipolar vectors of lower dimension than those from VB_H.

2) Binary Feature Interaction by Convolution: The features are independently encoded in current binary VSA models (Eq. 1), while interactions between features are overlooked; accordingly, we propose to use convolution to extract relationships between features. Given the strict resource and power constraints in resource-stringent scenarios, we employ binary convolution for the feature extraction on binary value vectors.

3) Vector Encoding: We follow the LDC training to apply the encoding procedure (Eq. 1). As aforementioned in Sec. II-C, the binary weights in this layer correspond to feature vector set $\mathbf{F}$. Yet, unlike other binary VSA models where $\boldsymbol{f}_i \in \mathbf{F}$ is generated for each feature position, the $\boldsymbol{f}_i$ in our UniVSA model represents the channel position of the binary convolution output. Consequently, the output of encoding $\boldsymbol{s}$ is the binary vector representation of the input ECoG signal.

4) Similarity Measurement with Soft Voting: In LDC training [11], the similarity measurement layer generates low-dimension class vectors, by optimizing a binary dense layer. However, more bias (also known as underfitting) will be introduced as the vector dimension decreases [19]. We upgrade this design, by using multiple binary dense layers in parallel, as an ensemble layer for classification. Therefore, a sample is evaluated by multiple similarity layers with average probabilities. Assuming Θ similarity layers, we have Θ sets of class vectors $\mathbf{C}$ generated. Eq.2 can be rewritten as

$$\text{label} = \arg\max_j \frac{1}{\Theta} \sum_{\theta=1}^{\Theta} \text{SIM}(\boldsymbol{s}, \boldsymbol{c}_j^\theta) = \arg\max_j \sum_{\theta=1}^{\Theta} \mathbf{C}^\theta \boldsymbol{s} \quad (4)$$

By jointly considering the similarity results of every dense layer, the output produces the category prediction on the class with the highest similarity.

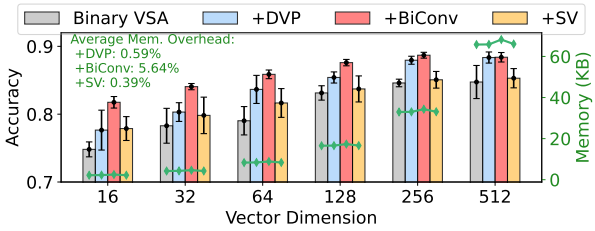


Fig. 4: The ablation study of the three-fold enhancements over binary VSA. The inference accuracy (bar) of binary VSA is significantly improved with little memory footprint (line).

B. Preliminary Assessment on UniVSA Model

We validate the positive impact of discriminated value projection (DVP), binary feature extraction (BiConv), and soft voting (SV) on inference accuracy and their memory footprint for resource-stringent scenarios, in Fig. 4. The evaluation is based on the EEGMMI dataset [20]. BiConv consistently improves the accuracy of binary VSA across all vector dimensions and help ensures a stable training process, evidenced by low accuracy deviations. This underscores *the necessity to incorporate feature interaction* in binary VSA models. DVP is less stable and initially underperforms BiConv when vector dimensions are low, but performs better as the vector dimension increases, even comparable to BiConv. SV underperforms the other two methods, yet still positively impacts binary VSA accuracy at lower vector dimensions, helping *alleviate underfitting issues* in low-capacity binary VSA models. By harnessing the unique strengths of these methods, UniVSA achieves a notable enhancement in binary VSA inference. In particular, the result of the memory footprint shows that these extensions have tiny overhead for the model size, i.e., $+0.59\%$ on DVP, $+5.64\%$ on BiConv, and $+0.39\%$ on SV, considering the overall kilobyte-scale memory requirement.

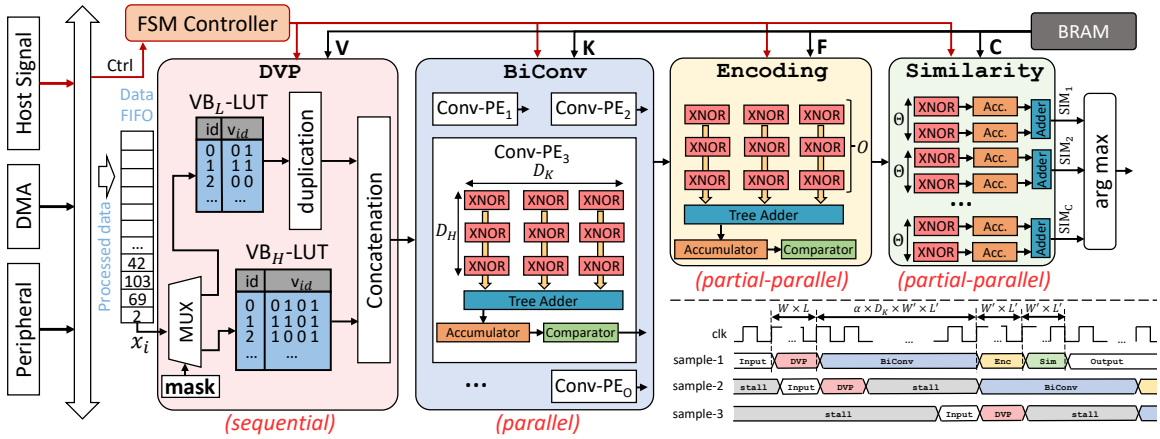


Fig. 5: The hardware overview of UniVSA. We indicate the execution scheduling under each module in parentheses, and demonstrate the overall pipelined scheduling of UniVSA for streaming inputs at the bottom right, where $\alpha = \max\{D_K, log_2 D_H\}$ is the execution time for a single convolution iteration.

IV. UniVSA: Hardware Implementation

A. Module Design

The hardware implementation of UniVSA contains four modules: **binary convolution, encoding, similarity measurement**, and **central controller**. Besides, we employ auxiliary circuits for **discriminated value projection** (DVP) and intermediate data buffering. We illustrate the hardware architecture in Fig. 5.

Discriminated Value Projection. The input features go through VB$_H$/VB$_L$ (decided by their feature importance **mask**) to derive the value vector for each feature. Since `DVP` operates significantly faster than `BiConv` (bottom right of Fig. 5). Parallelism on `DVP` would increase hardware overhead without reducing the latency of UniVSA. Thus, we do sequential execution on `DVP` to minimize resource usage and power consumption. A data FIFO is integrated at the `DVP` input to sequentially feed data elements.

Binary Convolution. The binary convolution layer extracts implicit features in the binary value vectors. As notation, value vectors have size (D_H, W, L) and the binary kernel $\mathbf{K}$ has size (O, D_H, D_K, D_K), where O is the output channel number and D_K is the kernel size. The binary convolution is accelerated by a *double buffering* strategy [21] to preload the next data block, without waiting for the accomplishment of current convolution. During convolution, we parallelize the computation by splitting kernels into O parts with size (D_H, D_K, D_K). Each part participates in computing to determine the binary value for one channel. Therefore, considering the output feature map with size (O, W', L'), $W' \times L' \times D_K$ iterations are required.

Encoding and Similarity Measurement. We flatten the output of binary convolution into shape $(O, W' \times L')$, to prepare for encoding. The encoding input is XNORed with feature vectors $\mathbf{F}$ and summed along the O-axis through an adder tree. The binarization is then performed to produce a sample vector $\boldsymbol{s}$ of dimension $W' \times L'$. Similarly, $\boldsymbol{s}$ is XNORed with Θ sets of class vectors $\mathbf{C}$ for the similarity measurement. For each category, the XNORed results are summed up to derive the final similarity. Only partial parallelism is applied on `Encoding` and `Similarity` along dimension O and Θ, to balance the vector operation and resource usage. Akin to `DVP`, their latency can be covered by `BiConv` under streaming inputs.

B. Hardware Consideration

Given the limited resource in tiny devices, we consider the **Memory** and **Resource** to quantify the hardware efficiency. The memory is determined by vector groups **V**, **F**, **K**, and **C**. With the input size (W and L), number of available values (M), number of categories (C), value vector dimension (D_H/D_L), kernel size (D_K) and output channel (O) in convolution, and number of voters (Θ), the required memory is

$$\begin{aligned}\textbf{Memory} = {} & M \times (D_H + D_L) + O \times D_H \times D_K^2 \\ & + W \times L \times O + W \times L \times \Theta \times C\end{aligned} \tag{5}$$

Since `BiConv` dominate resource usage (in Fig. 6), we mainly consider their resource for configuration. Without loss of generality, we define the resource overhead as

$$\textbf{Resource} \approx \beta \times D_K \times O \times D_H \tag{6}$$

where β is a coefficient indicating that the resource usage increment is approximately proportional to the kernel size and input/output channel of binary convolution.

We treat the memory footprint and resource usage as a penalty for UniVSA configurations, to balance the accuracy and the hardware overhead in the model design. The hardware overhead is normalized by taking the basis $\mathbf{M}_0$ and $\mathbf{R}_0$ with configuration $(D_H, D_L, D_K, O, \Theta, M) = (4, 2, 3, 64, 1, 256)$:

$$\mathcal{L}_{HW} = \lambda_1 \frac{\textbf{Memory}}{\mathbf{M}_0} + \lambda_2 \frac{\textbf{Resource}}{\mathbf{R}_0} \tag{7}$$

Here, λ_1 and λ_2 are scaling factors to align with the inference accuracy and differentiate these two hardware overheads.

V. Evaluation

A. Experiment Setup

Datasets. We evaluate the UniVSA framework on real-world BCI classification tasks and VSA tasks (in Table I):

TABLE I: Benchmark configurations.

	Domain	# of Classes	Input Size (W, L)	Model Config. $(D_H, D_L, D_K, O, \Theta)$
EEGMMI	Time	2	(16,64)	(8, 2, 3, 95, 1)
BCI-III-V	Frequency	3	(16,6)	(8, 1, 3, 151, 3)
CHB-B	Frequency	2	(23,64)	(8, 2, 3, 16, 3)
CHB-IB	Frequency	2	(23,64)	(4, 1, 5, 16, 1)
ISOLET	Time	26	(16,40)	(4, 4, 3, 22, 3)
HAR	Time	6	(16,36)	(8, 4, 3, 18, 3)

EEGMMI [20], BCI-III-V [22], CHB-B [23], CHB-IB [23], ISOLET [24], HAR [25]. The first four datasets are collected in electroencephalography (EEG) signals for BCI devices. EEGMMI records volunteers' activities from 64 channels, and BCI-III-V is collected for mental imagery activities. We follow the preprocessing in [26] and [22] for each dataset, respectively. The CHB (balanced and imbalanced version) dataset is for seizure detection. In addition, we include two tasks commonly evaluated for VSA models. ISOLET is a voice recording of 26-letter speaking, and HAR is an activity gesture recording from accelerometers and gyroscopes. Inputs for UniVSA are discretized to 256 levels in advance [11], i.e., $M = 256$, and shaped as 2-D of size (W, L).

Model Design. We determine the configuration of UniVSA model by considering both the accuracy and hardware overhead, i.e., $obj = (Acc - \mathcal{L}_{HW})$ with $\lambda_1 = \lambda_2 = 0.005$. The optimization procedure on configuration can also be referred to [27]. To search for the optimal configuration for each task, we utilize the evolutionary search with elitist preservation [28]. In Table I, we demonstrate the optimal searched configurations to all benchmarks. The UniVSA models are trained on Python 3.7 with NVIDIA A10 GPU acceleration.

Hardware Implementation. We construct the UniVSA on Ultra96-V2 SoC equipped with a ZU3EG FPGA [29]. Communication of control signals, data input, and final output between the CPU and the FPGA is conducted through `AXI_HPM_LPD` [30] interface. To optimize resource usage and scheduling, we develop the hardware implementation in Verilog using Vivado 2022.2.

B. Performance of Software Design

In Table II, we present the comparison of the accuracy and memory footprint between UniVSA and other efficient methods, including SOTA high-dimensional binary VSA training strategies (i.e., LeHDC [12] with $D = 10,000$), low-dimensional binary VSA training (i.e., LDC with $D = 128$ [11]), LDA (32-bit float), SVM (16-bit float with RBF kernel), and KNN (K=5). Although deep learning approaches were also explored, we exclude dense comparisons with them because of their failure on resource and power budget [2].

Among traditional ML models, SVM generally has better performance, notwithstanding its significantly larger model size ($\times 255$ on LDA). Although SVM is a favored method for BCIs [4], it could perform poorly on certain tasks, such as BCI-III-V and HAR, according to our evaluations. On the other hand, binary VSA models trained by LDC can have

TABLE II: Model comparison between UniVSA and other lightweight methods on *accuracy* and *memory*. Memory footprint (KB) is shown in parentheses.

	LDA	**KNN**	**SVM**	**LeHDC [12]**	**LDC [11]**	**UniVSA**
EEGMMI	0.7004 (8.19)	0.8262 (−)	0.8766 (11223.04)	0.7980 (1602.50)	0.8279 (16.54)	0.8971 (13.59)
BCI-III-V	0.8599 (1.15)	0.9888 (−)	0.8971 (510.22)	0.8235 (443.75)	0.9370 (1.71)	0.9545 (3.57)
CHB-B	0.9067 (11.78)	0.9744 (−)	0.9819 (1990.14)	0.8992 (2162.50)	0.9669 (23.71)	0.9774 (4.51)
CHB-IB	0.9142 (11.78)	0.9488 (−)	0.9729 (3612.29)	0.8675 (2162.50)	0.9639 (23.71)	0.9684 (3.67)
ISOLET	0.9410 (66.56)	0.9140 (−)	0.9602 (5048.32)	0.9489 (1152.50)	0.9133 (10.78)	0.9282 (8.36)
HAR	0.7625 (13.82)	0.5582 (−)	0.7852 (6743.81)	0.9523 (1047.50)	0.9256 (9.44)	0.9338 (3.14)
average	0.8475 (18.88 KB)	0.8685 (−)	0.9124 (4.24 MB)	0.8816 (1.29 MB)	0.9225 (15.05 KB)	0.9445 (8.31 KB)

comparable SVM inference performance on average while consuming significantly less memory, i.e., 0.5% of the memory required by SVM. However, LDC underperforms SVM on some tasks, e.g., 4.87% lower on EEGMMI and 4.69% lower on ISOLET, echoing the accuracy challenges in Sec. I. In addition, high-dimensional binary VSA models (LeHDC) achieve better accuracy on certain tasks, yet require memory on the MB scale.

Our proposed UniVSA framework on average outperforms other ML methods in terms of accuracy. When compared to LDC, UniVSA shows superior accuracy across all tasks, underscoring its improved model design and inference performance. Additionally, UniVSA's design strategy, which considers hardware overhead, results in a smaller memory footprint than LDC for most evaluated tasks. Notably, the memory footprint of UniVSA on CHB tasks is much lower than that of LDC. This is because the memory overhead (Eq.5) of binary VSA is dominated by the size of feature vectors $\mathbf{F}$, i.e., $(O, W \times L)$ for UniVSA and $(D, W \times L)$ for LDC, where the searched $O = 16$ in UniVSA is much smaller than $D = 128$ in binary VSA trained from LDC. While UniVSA does not always outperform all other ML methods, with SVM frequently delivering the highest performance, it does offer consistently high accuracy. The performance of SVM, however, is task-dependent and it underperforms on certain tasks. In contrast, UniVSA generally delivers close-to-best results, except for the ISOLET task, demonstrating its generality in lightweight classification tasks.

C. Performance of Hardware Implementation

Framework Comparison. We compare the hardware implementation of UniVSA with ML methods including SVM [32] and KNN [16], and SOTA binary VSA hardware implementations, LookHD [9] with $D = 2000$ and LDC [11] with $D = 64$. Additionally, we consider lightweight NN approaches, i.e., BNN [14]/QNN [13]. We choose the comparison by the closest FPGA architecture in the previous work so that the lowest variations in resource allocation and power consumption are induced during implementation. The results are summarized in Table III.

① Compared with conventional ML methods and NN models, UniVSA can achieve much lower hardware overhead, with only

TABLE III: The hardware performance of UniVSA against other models. Values in parentheses are estimated. ISOLET is selected for UniVSA since it has the closest input size to other binary VSA models.

	FPGA Arch.	Input Size / Classes	Frequency (MHz)	Memory (KB)	Latency (ms)	Power (W)	LUTs ($\times 10^3$)	BRAMs	DSPs
SVM [31]	Virtex-5	(20,20) / –*	84	(406)	14.29	3.2	31.85	131	59
KNN [16]	Stratix IV	64 / 2	131.42	–	69.12	24	135	–	80
BNN [14]	Zynq-ZU3EG	(3,32,32) / 10	250	–	(0.36)	4.1	51.44	212	126
QNN [13]	Zynq-ZU3EG	(3,224,224) / 1000	250	(1450)	(24.33)	5.5	51.78	159	360
LookHD [9]	Kintex-7	617 / 26	200	(165)	–	(9.52)	165	175	807
LDC [11]	Zynq-ZU3EG	784 / 10	200	6.48	0.004	(0.016)	0.75	5	1
UniVSA	Zynq-ZU3EG	(16,40) / 26	250	8.36	0.044	0.11	7.92	1	0

* The SVM implementation is evaluated on a detection task, not classification.

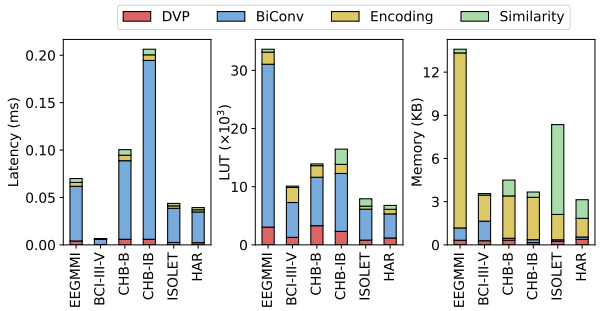


Fig. 6: The hardware overhead of each stage in UniVSA.

TABLE IV: The hardware performance of UniVSA on all tasks. The throughput is estimated for streaming inputs to UniVSA.

Benchmark	Latency (ms)	Power (W)	LUTs ($\times 10^3$)	BRAMs	DSPs	Throughput ($\times 10^3$)
EEGMMI	0.070	0.45	33.62	3	0	17.34
BCI-III-V	0.007	0.18	10.10	1	0	184.84
CHB-B	0.100	0.34	13.92	1	0	12.06
CHB-IB	0.206	0.21	16.46	1	0	5.30
ISOLET	0.044	0.11	7.92	1	0	27.78
HAR	0.039	0.10	6.78	1	0	30.85

0.1∼0.5× resource usage. UniVSA also consumes order-of-magnitude lower power and latency than these ML methods. On the other hand, as indicated in an FPGA-based SVM survey [15], all the collected SVMs have power consumption greater than **1.5W**. Taking into account this solid line, except for low-dimensional binary VSA (i.e., LDC) and UniVSA, other models in Table III are not suitable for BCI tasks. Note that BNN and QNN possibly have better inference performance than binary VSA models, especially on complex classification tasks. However, we propose UniVSA as a binary VSA framework to conduct easy classification tasks while requiring ultra-high efficiency, where UniVSA can achieve comparable inference accuracy and a minimal hardware budget.

② Compared with other binary VSA work, such as LookHD [9] and LDC [11], UniVSA shows comprehensive superiority over LookHD, yet consumes more power and resources than LDC. Nevertheless, considering that UniVSA can achieve better accuracy and lower memory footprint (Table II), and the resource usage is still acceptable (less than SVM [32]) by BCIs, we account for this as a feasible trade-off to better support classification tasks on resource-stringent devices.

Ablation Study. We further demonstrate the hardware overhead of the computing stages in UniVSA, in Fig. 6. The `BiConv` layer consumes the most resources and execution time in all tasks, far more than other stages. This supports our previous discussion that `BiConv` dominates the UniVSA execution, which is the motivation that we sequentialize the `DVP`, `Encoding`, and `Similarity` stages to minimize the resource usage, rather than optimize their latency using parallelism. Moreover, the memory footprint of `BiConv` is very low, since only the kernel $\mathbf{K}$ is stored with a small dimension (O, D_H, K, K); yet, $\mathbf{F}$ with size $(O, W \times L)$ or $\mathbf{C}$ with size $(\Theta, C, W \times L)$ occupies most of the memory footprint when the input size or classes is large.

Hardware Performance on Benchmark. We provide the hardware performance for all tasks in Table IV. Overall, all tasks demonstrate power consumption of less than 0.5W and latency under 0.2ms on the FPGA. The throughput analysis under streaming inputs demonstrates the efficiency of UniVSA, that all tasks can achieve over 5,000 throughput, sufficient for BCI applications. With pipeline, the execution time is close to the `BiConv` latency. Comparison between tasks reveals a strong correlation between hardware overhead and factors such as data input size and convolution channels. This finding bolsters our co-optimization strategy for UniVSA model design, taking hardware into consideration.

VI. Conclusion

In this paper, we present an end-to-end algorithm-hardware co-optimization framework, UniVSA, solving the challenges in current binary VSA design on resource-stringent devices. UniVSA advances the binary VSA by model designing and hardware acceleration with a completed process. Our evaluation demonstrates the effectiveness of UniVSA on benchmarks selected under resource-stringent scenarios, achieving the best inference accuracy on average and lowest memory footprint against other ML models. Further evaluation also demonstrates the efficiency and lightweight of UniVSA.

Acknowledgment

Shijin Duan, Nuntipat Narkthong, and Xiaolin Xu were supported in part by the US NSF under Grants CNS-2326597, CNS-2239672, and a Cisco Research Award. Shaolei Ren was supported in part by the US NSF under CNS-2326598.

REFERENCES

[1] L. Dutta and S. Bharali, "Tinyml meets iot: A comprehensive survey," *Internet of Things*, vol. 16, p. 100461, 2021.

[2] S. Aggarwal and N. Chugh, "Review of machine learning techniques for eeg based brain computer interface," *Archives of Computational Methods in Engineering*, vol. 29, no. 5, pp. 3001–3020, 2022.

[3] Z. Lv, L. Qiao, Q. Wang, and F. Piccialli, "Advanced machine-learning methods for brain-computer interfacing," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 5, pp. 1688–1698, 2020.

[4] I. Karageorgos, K. Sriram, J. Veselỳ, M. Wu, M. Powell, D. Borton, R. Manohar, and A. Bhattacharjee, "Hardware-software co-design for brain-computer interfaces," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 391–404.

[5] S. Kumar, A. Sharma, and T. Tsunoda, "An improved discriminative filter bank selection approach for motor imagery eeg signal classification using mutual information," *BMC bioinformatics*, vol. 18, pp. 125–137, 2017.

[6] G. Qiao, S. Hu, T. Chen, L. Rong, N. Ning, Q. Yu, and Y. Liu, "Stbnn: Hardware-friendly spatio-temporal binary neural network with high pattern recognition accuracy," *Neurocomputing*, vol. 409, pp. 351–360, 2020.

[7] P. Kanerva, "Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors," *Cognitive computation*, vol. 1, no. 2, pp. 139–159, 2009.

[8] G. Karunaratne, M. Le Gallo, G. Cherubini, L. Benini, A. Rahimi, and A. Sebastian, "In-memory hyperdimensional computing," *Nature Electronics*, pp. 1–11, 2020.

[9] M. Imani, Z. Zou, S. Bosch, S. A. Rao, S. Salamat, V. Kumar, Y. Kim, and T. Rosing, "Revisiting hyperdimensional learning for fpga and low-power architectures," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 221–234.

[10] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1–52, 2023.

[11] S. Duan, X. Xu, and S. Ren, "A brain-inspired low-dimensional computing classifier for inference on tiny devices," in *tinyML Research Symposium 2022*, 2022.

[12] S. Duan, Y. Liu, S. Ren, and X. Xu, "Lehdc: Learning-based hyperdimensional computing classifier," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1111–1116. [Online]. Available: https://doi.org/10.1145/3489517.3530593

[13] Y. Yang, Q. Huang, B. Wu, T. Zhang, L. Ma, G. Gambardella, M. Blott, L. Lavagno, K. Vissers, J. Wawrzynek *et al.*, "Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas," in *Proceedings of the 2019 ACM/SIGDA international symposium on field-programmable gate arrays*, 2019, pp. 23–32.

[14] Y. Zhang, J. Pan, X. Liu, H. Chen, D. Chen, and Z. Zhang, "Fracbnn: Accurate and fpga-efficient binary neural networks with fractional activations," in *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2021, pp. 171–182.

[15] S. Afifi, H. GholamHosseini, and R. Sinha, "Fpga implementations of svm classifiers: A review," *SN Computer Science*, vol. 1, pp. 1–17, 2020.

[16] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient knn algorithm implemented on fpga based heterogeneous computing system using opencl," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 167–170.

[17] A. H. Shoeb and J. V. Guttag, "Application of machine learning to epileptic seizure detection," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 975–982.

[18] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1-2, pp. 273–324, 1997.

[19] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[20] P. PhysioBank, "Physionet: components of a new research resource for complex physiologic signals," *Circulation*, vol. 101, no. 23, pp. e215–e220, 2000, https://archive.physionet.org/pn4/eegmmidb/.

[21] X. Tan, X.-W. Shen, X.-C. Ye, D. Wang, D.-R. Fan, L. Zhang, W.-M. Li, Z.-M. Zhang, and Z.-M. Tang, "A non-stop double buffering mechanism for dataflow architecture," *Journal of Computer Science and Technology*, vol. 33, pp. 145–157, 2018.

[22] J. R. Millan, "On the need for on-line learning in brain-computer interfaces," in *2004 IEEE international joint conference on neural networks (IEEE Cat. No. 04CH37541)*, vol. 4. IEEE, 2004, pp. 2877–2882, https://www.bbci.de/competition/iii/desc_V.html.

[23] A. H. Shoeb, "Application of machine learning to epileptic seizure onset detection and treatment," 2009, https://archive.physionet.org/pn6/chbmit/.

[24] D. Dua and C. Graff, "UCI machine learning repository," 2017, https://archive.ics.uci.edu/ml/datasets/isolet.

[25] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." p. 3, 2013, https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones#.

[26] X. Zhang, L. Yao, X. Wang, J. Monaghan, D. Mcalpine, and Y. Zhang, "A survey on deep learning-based non-invasive brain signals: recent advances and new frontiers," *Journal of neural engineering*, vol. 18, no. 3, p. 031002, 2021. [Online]. Available: https://github.com/xiangzhang1015/Deep-Learning-for-BCI

[27] L. Yang, Z. Yan, M. Li, H. Kwon, L. Lai, T. Krishna, V. Chandra, W. Jiang, and Y. Shi, "Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[28] D.-C. Dang, T. Friedrich, T. Kötzing, M. S. Krejca, P. K. Lehre, P. S. Oliveto, D. Sudholt, and A. M. Sutton, "Escaping local optima using crossover with emergent diversity," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 484–497, 2017.

[29] Xilinx, "Zynq ultrascale+ mpsoc data sheet: Overview," 2021. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds891-zynq-ultrascale-plus-overview.pdf

[30] XILINX. (2017) Vivado axi reference guide. [Online]. Available: https://docs.xilinx.com/v/u/en-US/ug1037-vivado-axi-reference-guide

[31] C. Kyrkou, T. Theocharides, and C.-S. Bouganis, "An embedded hardware-efficient architecture for real-time cascade support vector machine classification," in *2013 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, 2013, pp. 129–136.

[32] M. Qasaimeh, A. Sagahyroon, and T. Shanableh, "Fpga-based parallel hardware architecture for real-time image classification," *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp. 56–70, 2015.

iTaskSense: Task-Oriented Object Detection in Resource-Constrained Environments

SungHeon Jeong, Hamza Errahmouni Barkam, Hyunwoo Oh, Hanning Chen, Tamoghno Das, Zhen Ye, Mohsen Imani
University of California, Irvine, USA
{sungheoj, herrahmo, hyunwooo, hanningc, tamoghnd, zye14, m.imani}@uci.edu

***Abstract*—Task-oriented object detection is increasingly essential for intelligent sensing applications, enabling AI systems to operate autonomously in complex, real-world environments such as autonomous driving, healthcare, and industrial automation. Conventional models often struggle with generalization, requiring vast datasets to accurately detect objects within diverse contexts. In this work, we introduce iTask, a task-oriented object detection framework that leverages large language models (LLMs) to generalize efficiently from limited samples by generating an abstract knowledge graph. This graph encapsulates essential task attributes, allowing iTask to identify objects based on high-level characteristics rather than extensive data, making it possible to adapt to complex mission requirements with minimal samples.**

iTask addresses the challenges of high computational cost and resource limitations in vision-language models by offering two configuration models: a distilled, task-specific vision transformer optimized for high accuracy in defined tasks, and a quantized version of the model for broader applicability across multiple tasks. Additionally, we designed a hardware acceleration circuit to support real-time processing, essential for edge devices that require low latency and efficient task execution. Our evaluations show that the task-specific configuration achieves a 15% higher accuracy over the quantized configuration in specific scenarios, while the quantized model provides robust multi-task performance. The hardware-accelerated iTask system achieves a 3.5x speedup and a 40% reduction in energy consumption compared to GPU-based implementations. These results demonstrate that iTask 's dual-configuration approach and situational adaptability offer a scalable solution for task-specific object detection, providing robust and efficient performance in resource-constrained environments.

I. Introduction

Task-oriented object detection [1], [2] is crucial for a wide range of applications, from autonomous driving and surveillance to healthcare and industrial automation [2], [3]. Unlike general object detection, task-oriented detection focuses on identifying and localizing objects within a specific contextual frame, allowing AI systems to make informed decisions and initiate context-driven actions [4]. Meeting the demands of task-oriented detection often requires models to adapt to complex mission requirements, which are frequently defined by abstract characteristics rather than explicit labels. This dependency on nuanced contextual understanding challenges traditional models, which typically rely on large, annotated datasets to achieve high accuracy [5]. Furthermore, implementing these models in real-world scenarios introduces additional constraints, such as computational and memory efficiency, particularly in resource-constrained environments like edge devices.

Another challenge is that traditional vision-language models, such as MDETR [6], used to solve tasks [7] are very resource-intensive, consuming high amounts of memory and power. This is particularly evident in intelligent sensing applications, where edge devices must operate efficiently with limited computational resources and potentially minimal data samples [8]. To address these demands, we present *intelligent Task Oriented Aritficial Intelligence (*iTask*)*, a task-oriented object detection framework designed to excel in complex missions and adapt effectively to a wide range of real-world tasks.

Consider a complex mission scenario where a system must identify any object that could be used to open a bottle of beer. In a traditional machine learning context, a model would need vast amounts of labeled data to capture all potential objects capable of performing this action, such as bottle openers, lighters, or even certain types of spoons [2]. However, real-world scenarios often require generalizing from very few samples, which is nearly impossible for traditional methods without comprehensive datasets. iTask addresses this challenge by using a large language model (LLM) to generate an abstract knowledge graph that encapsulates the essential attributes required to perform the task, such as "firm leverage" or "rigid edge". This high-level understanding allows iTask to define these characteristics and identify objects that meet them, enabling generalization even with minimal data samples.

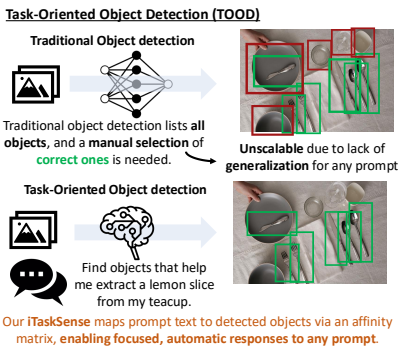


Fig. 1: **iTask framework for task-oriented object detection**: Unlike traditional detection, iTask uses an affinity matrix for prompt-based, context-aware responses, enhancing precision and scalability across domains while tackling high computational demands and complex object semantics.

For example, in a scenario where the goal is to extract a lemon slice from a glass, iTask can analyze objects that share attributes relevant to the task, as shown in Figure 1. While a knife, fork, or spoon might be typical choices, an unexpected tool like a pen could also serve the purpose, yet it may not appear in conventional datasets unless trained with vast samples. By leveraging the LLM-generated knowledge graph, iTask can identify and rank objects based on their feature alignment with task requirements, selecting the most suitable option even in cases where there is no perfect match or multiple candidates.

This capability is further enhanced through iTask's situational awareness, which enables real-time mission adaptability. When deployed on edge devices, iTask can interact dynamically with its environment to analyze content based on specific mission objectives. For instance, if an edge device is instructed to "identify objects capable of providing leverage," iTask can assess available objects and select those that match the task-defined attributes . This real-time adaptability not only reduces dependency on extensive datasets but also significantly improves resource efficiency by focusing on task-relevant data, optimizing computational load, and conserving energy.

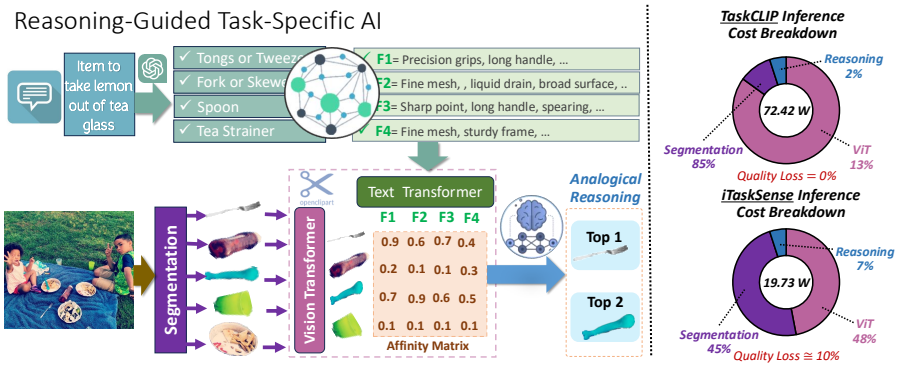


Fig. 2: **The** iTask **framework for reasoning-guided task-specific object selection and efficient hardware acceleration:** Given the prompt, segmented objects are encoded and matched to functional features (F1–F4) via an affinity matrix. Analogical reasoning ranks items (e.g., forks, tweezers) based on task-relevant attributes. **Hardware optimization for taskCLIP inference:** Smaller CLIP models reduce cost but may lose quality; knowledge distillation improves their accuracy. A unified ASIC design accelerates segmentation (convolution-based) and ViT (attention-based) tasks on the same hardware, achieving significant computational efficiency.

To overcome the limitations of traditional vision-language models in intelligent sensing, iTask is built with two configurations tailored for different mission requirements. The first configuration uses knowledge distillation to create a smaller, task-specific vision transformer that maximizes detection performance for specific tasks. While smaller models reduce computational costs, they often lose quality compared to larger models. iTask addresses this challenge by employing knowledge distillation techniques to make smaller CLIP models as accurate as their larger counterparts, ensuring high-quality performance without the associated cost overhead. The second configuration employs quantization to compress the model, enabling versatility across multiple tasks with reduced computational demands [9], [10]. These configurations allow iTask to provide efficient task-oriented detection in a wide variety of real-world scenarios while maintaining adaptability and responsiveness.

Additionally, a hardware circuit was designed to accelerate iTask, enabling rapid processing near the sensor and meeting the real-time demands of intelligent sensing environments. Since segmentation and vision transformers (ViT) represent fundamentally different network architectures—segmentation being convolution-based and ViT attention-based—specific hardware accelerators are required for each. To address this challenge, we propose a unified ASIC architecture that accelerates both ViT and segmentation tasks on the same hardware platform, significantly improving computational efficiency for taskCLIP inference. This design is essential for edge-based AI applications, where high-speed processing and minimal latency are critical. Through this combined approach of model optimization, situational adaptability, and hardware acceleration, iTask delivers a scalable, efficient solution tailored for resource-constrained environments where robust task-specific detection is essential.

The contributions of this paper are threefold: (1) the introduction of iTask, a framework that integrates LLM-generated knowledge graphs for task-specific generalization with minimal samples, achieving a 15% improvement in accuracy over traditional quantized configurations in specific scenarios; (2) dual configuration models, including knowledge distillation and quantization, for efficient task-specific and multi-task object detection; and (3) a hardware acceleration circuit that enables real-time, low-latency performance in intelligent sensing applications, delivering a 3.5x speedup and a 40% reduction in energy consumption compared to standard GPU-based implementations These advancements make iTask a scalable, efficient solution for complex, task-oriented detection in resource-constrained environments.

II. Related Work

A. Vision-Language Models

Vision-Language Models (VLMs) integrate visual and textual data for multimodal representation, bridging visual content with textual queries. VinVL, for example, achieves high performance in multi-modal tasks, utilizing a vision-language-enhanced object detection model [11]. MiniVLM adapts VLM principles for resource-constrained environments using a Two-stage Efficient Extractor (TEE) and a compact transformer-based fusion module inspired by EfficientDet and MiniLM [12]–[14]. Despite their strengths, VLMs remain on the critical path for decision-making, where costly and non-interpretable outputs present challenges. Guided by MiniVLM's efficiency, we design iTask to combine intelligent sensing with LLMs, addressing task-oriented detection challenges in edge computing scenarios while maintaining interpretability and computational efficiency.

B. Task-Oriented Object Detection

Task-oriented object detection [1], [2] focuses on identifying objects relevant to specific tasks, requiring robust reasoning to address complex demands. Task-aligned One-stage Object Detection (TOOD) enhances classification and localization by generating task-interactive features to reduce spatial misalignment [15]. Similarly, Wang et al. [16] developed a task-oriented degraded image enhancement network, achieving precision improvements of up to 1.6% on degraded datasets like URPC2020 [17]. However, achieving efficiency in TOOD remains challenging due to the reliance on large, compute-intensive models. To address these issues, our work integrates TOOD with intelligent sensing and lightweight LLMs to enhance detection precision, efficiency, and generalization in resource-constrained edge environments, while leveraging hardware optimizations for task-specific inference.

C. Intelligent Sensors

Intelligent sensors have become a transformative innovation in autonomous systems by enabling near-sensor data processing, which reduces latency, enhances privacy, and minimizes bandwidth through selective data acquisition and compression. Huang et al. [18] demonstrated notable energy savings using near-sensor AI models that process only frames of interest. Similarly, Zhang et al. [19] showed that adaptive thresholds improve compression and reconstruction accuracy,

while Lu et al. [20] achieved dynamic threshold adjustments, enhancing performance. Despite these advancements, intelligent sensing faces significant challenges in algorithm-hardware co-optimization [21]. Our framework addresses these challenges by identifying regions of interest (ROIs) that enable lightweight sensors to filter irrelevant data efficiently. Combined with task-oriented object detection and VLMs, this approach significantly improves the energy efficiency and adaptability of intelligent sensing systems. To achieve real-time detection and reasoning in edge environments with limited resources, we design and implement a domain-specific ASIC accelerator to enhance our model's performance.

D. Hardware Design for Intelligent Sensing

Hardware design is critical for intelligent sensing applications, where energy efficiency and accuracy are constrained by sensor type, data resolution, and model complexity. Taha et al. [22] developed a low-power wireless transmission architecture that activates only critical sensors, leveraging near-sensor deep learning to balance efficiency and accuracy. Similarly, Ballard et al. [23] optimized spectral encoding for hyperspectral satellites, significantly reducing costly data transmission overheads. Huang et al. [18] designed a near-sensor object detection model capable of capturing frames of interest, eliminating the need for high-bandwidth transmission. However, hardware limitations often prevent running multiple algorithms like segmentation and ViT on the same platform. To overcome this, our work introduces a unified ASIC design that supports both convolution-based segmentation and attention-based ViT tasks. This architecture enables efficient taskCLIP inference by combining task-oriented detection with energy-efficient hardware acceleration, ensuring scalability and adaptability in dynamic edge environments.

III. Framework

A. Core Design of iTask for Efficient Task-Driven Detection

Ignited by prior research [1], iTask initially integrates the open-vocabulary capability of OpenAI CLIP with the reasoning power of LLM. Subsequently, it emphasizes model compression along with hardware and software co-design for deploying the framework on edge. It employs a multi-stage process to bridge the semantic gap between task affordances and visual object features, optimizing both accuracy and efficiency for edge computing scenarios.

Figure 2 presents the framework of iTask. The first stage of the iTask framework involves general object detection using pre-trained models to identify potential object candidates. In this step, bounding boxes and object features are generated for all items in the scene. To ensure effective reasoning and selection, iTask integrates an LLM-guided parsing mechanism to extract task-specific affordance attributes, such as "precision grip" or "long handle," relevant to the given prompt. These attributes are encoded as visual features, enabling a unified embedding space for images and text.

In the second stage, task-specific object selection is performed using a transformer-based aligner module. Inspired by TaskCLIP, the aligner recalibrates embeddings from the vision-language model (VLM), addressing misalignment between object images (e.g., nouns) and their visual attributes (e.g., adjective phrases). The aligner's multi-layer design incorporates self-attention and cross-attention mechanisms to refine the embeddings, ensuring they accurately represent task-specific affordances. This recalibration improves the semantic alignment and ensures high-quality reasoning-driven selection of objects.

Finally, iTask employs a trainable scoring function to process the affinity matrix generated by embedding alignment. This scoring function ranks objects based on their suitability for the specified task,

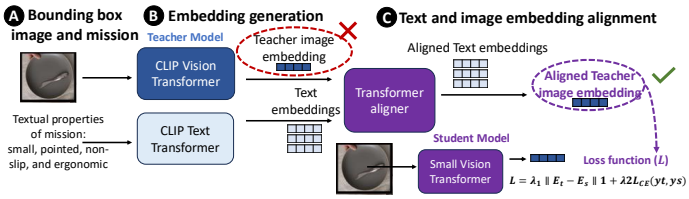


Fig. 3: Diagram of the iTask framework with knowledge distillation. The teacher model (CLIP ViT) generates image and text embeddings distilled to a smaller ViT model. A transformer aligner enhances alignment for task-specific object detection, addressing low performance when using pre-aligned image embeddings.

enabling robust task-oriented detection. By leveraging the inherent capabilities of VLMs and combining them with reasoning mechanisms, iTask achieves scalable and efficient detection, outperforming single-stage approaches in terms of generalizability and adaptability.

B. Lightweight Adaptation of TaskCLIP

To address the computational demands of the iTask framework, we introduce a strategy for lightweight adaptation, optimizing the performance of large vision-language models (VLMs) like TaskCLIP for resource-constrained environments. This approach employs two key techniques, affinity mimicking and weight inheritance, to compress and accelerate the model while maintaining robust task-oriented detection capabilities. Affinity mimicking allows the lightweight model to replicate the cross-modal feature alignment of a larger VLM. The image-text affinity scores generated by the large model, A_{large}, are distilled into the lightweight model, A_{light}, using the following loss function:

$$\mathcal{L}_{\text{affinity}} = \|A_{\text{large}} - A_{\text{light}}\|_F^2,$$

where $\|\cdot\|_F$ denotes the Frobenius norm. This ensures that the lightweight model retains essential semantic understanding for task-specific detection. To improve initialization, critical weights from the large model, θ_{large}, are transferred to the lightweight model, θ_{light}, using a selective masking strategy:

$$\theta_{\text{light}} = M \odot \theta_{\text{large}},$$

where M is a learnable mask identifying important weights, and $\odot$ denotes element-wise multiplication. By integrating these techniques, we significantly reduce the model size and computational complexity of iTask while preserving its task-specific detection performance. As shown in the inference cost breakdown on the right side of Figure 2, the proposed approach reduces reasoning and segmentation costs, enabling efficient and accurate task-driven detection across diverse real-world applications.

While smaller CLIP models reduce computational cost, they may suffer from quality degradation; however, knowledge distillation effectively enhances their accuracy. To further support this efficiency, a unified ASIC design accelerates both segmentation (convolution-based) and ViT (attention-based) tasks on the same hardware. This hardware optimization achieves significant computational efficiency by seamlessly integrating the distinct requirements of these tasks, making TaskCLIP inference highly suitable for resource-constrained environments such as edge computing systems.

C. Knowledge Distillation for Task-Specific Precision

To address the computational demands of the iTask framework while maintaining high task-specific performance, we propose a knowledge distillation technique that uses EfficientNet as a starting point to train a lightweight student model as a substitute for the vision transformer in CLIP, as shown in Figure 3. This approach targets the challenge of achieving the quality of large CLIP models without

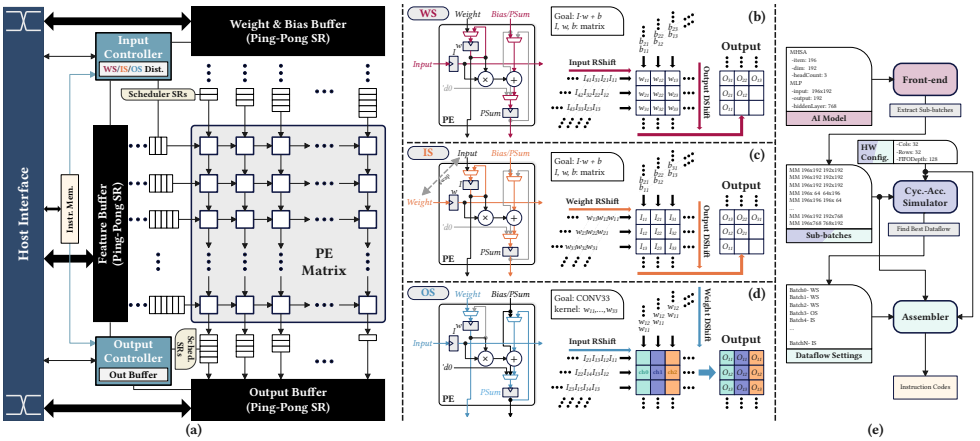


Fig. 4: The hardware/software stack for iTask acceleration. (**a**) The proposed systolic-array-based hardware accelerator architecture. (**b**) Weight stationary dataflow implemented with our PE design. (**c**) Input stationary dataflow implemented with our PE design. (**d**) Output stationary dataflow implemented with our PE design. (**e**) The compiler stack for the proposed accelerator architecture.

incurring their computational costs. Our distilled model balances accuracy and efficiency effectively, making it suitable for resource-constrained environments.

In this setup, the original CLIP vision transformer serves as the teacher model, guiding the training of a compact student model. The student model employs EfficientNet's compound scaling methodology [24], which systematically balances network depth, width, and resolution to achieve both efficiency and accuracy. This ensures that the student captures essential visual patterns while aligning semantically with textual embeddings. The distillation process minimizes a combined loss function:

$$\mathcal{L} = \lambda_1 \|E_t - E_s\|_1 + \lambda_2 \mathcal{L}_{\text{CE}},$$

where E_t and E_s are the embeddings from the teacher and student models, respectively, and $\mathcal{L}_{\text{CE}}$ is the cross-entropy loss between the output distributions of the two models. The L1 loss preserves the semantic alignment between teacher and student embeddings, while the cross-entropy loss ensures consistency in task-specific predictions. To stabilize training and reduce representational gaps, knowledge is transferred progressively in multiple stages. This approach ensures that the student model iteratively refines its representations while avoiding overfitting the teacher's embeddings. As depicted in Figure 3, a task-specific transformer aligner further refines the student model's embeddings to enhance task-related affordances. This recalibration addresses subtle visual details crucial for precise object detection in resource-constrained environments.

IV. Hardware Acceleration

Accelerating iTask benefits when utilizing unified hardware architecture accelerating both CNN and ViT. This requires a balance between general-purpose and domain-specific nuances. The existing parallel processing hardware, such as GPU and FPGA, have negatively biased nuances in terms of iTask acceleration. The former is biased toward general-purpose nuance, enabling scalability but lacking energy efficiency. The latter can organize the dataflow more flexibly but lacks computing power due to its intrinsic architectural limitations, thus, optimized to to accelerate specific architectures. Moreover, current domain-specific accelerators, whether FPGA-based or ASIC-based, typically concentrate solely on either CNN or VIT acceleration, rendering them ill-suited to support state-of-the-art vision language models. To solve all these challenges, we designed the hardware accelerator for ASIC implementation, expanding the scalability to diverse task-oriented applications for iTask.

Fig. 4 presents the hardware/software stack designed to accelerate the iTask. The stack consists of the systolic array-based hardware accelerator, and the domain-specific compiler to generate the instruction codes executed on the accelerator.

A. Hardware Accelerator

For most computation kernels in ViT and CNN workloads, we designed the systolic array architecture with 8-bit MAC in each PE, which enhances data reusability and overcomes memory-bound limitations. The unified PE design in Fig. 4(b)-(d) enables to support three dataflows, including the weight stationary (WS), input stationary (IS), and output stationary (OS), with proper organization of multiplexers (MUXes) in weight and partial sum (PSum) registers. This ensures the optimal dataflow execution for each computation layer, practically accelerating diverse ViT and CNN configurations.

Our architecture organized the input buffers (feature, weight, bias) as shift registers (SRs) with a ping-pong strategy, enabling seamless feeding to the systolic array (see Fig. 4(a) PE Matrix). This avoids using SRAM macros in ASIC design, as SRAM would present the critical-path delay due to its addressing circuit, making it a bottleneck in our design, which adopts 8-bit operations to ensure high-speed operations. This not only increases the maximum frequency but also the energy efficiency, as the ping-pong SRs do not require much memory compared to the SRAM buffer. Another notable characteristic is the adoption of scheduler SRs (see Fig. 4(a)) in both I/O buffers. The scheduler SRs automatically generate the sequence required for the systolic array computation, which is presented in Fig 4(b)-(d) for each dataflows, to implement ViT and CNN operations from the original matrix data structure. This eliminates the host computer's data re-ordering workloads, including pre- and post-processing.

B. Compiler Stack

The compiler stack, illustrated in Fig. 4(e), organizes a three-stage operation. The front-end part extracts the lower-level computation layers, e.g., matrix multiplication or convolution, from the high-level machine learning model architectures, including CNN and ViT. This enables the swift diversification of the model's hyperparameters.

The computation layers extracted from the front-end are cast to the cycle-accurate simulator with systolic array configuration (M×N). The simulator executes each computation layer with supporting dataflows,

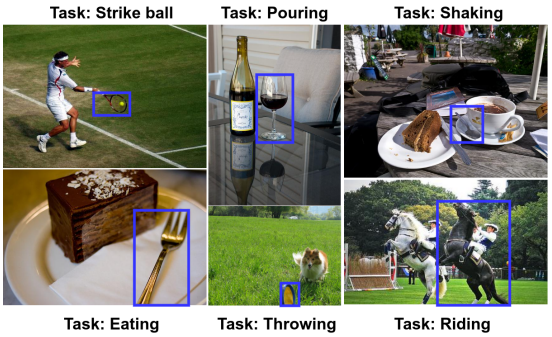


Fig. 5: Examples of task-oriented object detection with iTask, where specific objects are identified based on the task context. Tasks include 'Strike ball,' 'Pouring,' 'Shaking,' 'Eating,' 'Throwing,' and 'Riding,' with detected objects highlighted in blue. This approach enables context-aware object recognition tailored to task-specific requirements.

deriving the optimal dataflow for each layer. This simulator is developed upon Scalesim [25], modifying the cycle calculation to apply our accelerator's data-path using ping-pong SR FIFO.

Finally, the assembler produces the instruction code binary to execute given machine learning tasks using specific hardware configurations. With the optimized dataflow for each layer from the simulator executed using the same configurations, the optimal code for the ML workload is generated.

V. Evaluation

This section presents the evaluation of the iTask framework across various task-oriented object detection tasks, highlighting the performance of our original configuration and quantized and knowledge distillation configurations. We use AP@0.5 as our primary metric for evaluating task-oriented object detection, as it offers a suitable balance between precision and recall, especially relevant for task-specific applications that prioritize accurate identification over general object detection. AP@0.5 means at a 50% Intersection over the Union (IoU) threshold, is chosen as the primary evaluation metric due to its robustness in balancing precision and recall for task-specific object detection. The machine learning side experiments were conducted on an NVIDIA RTX 4090, with hardware synthesis performed through 28nm CMOS technology using Synopsys Design Compiler, specifically 0.81v @ 125C corner. The accelerator is designed with Chisel [26] for swift deployment across diverse systolic array configurations. In this evaluation, we selected the 128 $\times$ 128, a widely used configuration in other works. Table III presents the configurations and results. We also analyze the energy and speed our hardware accelerator achieves in task-oriented scenarios. In task-oriented applications, the focus is often on detecting objects that meet specific criteria rather than maximizing accuracy across all detected instances. AP@0.5 allows us to prioritize objects that align closely with each task's intended functionality or affordances, making it ideal for scenarios where the utility of an object (e.g., can it hold flowers?) is more critical than precise localization. Higher IoU thresholds, while beneficial for general detection, may penalize task-oriented models by undervaluing objects that partially meet task requirements, whereas AP@0.5 provides a balanced metric for functional correctness in such cases.

A. iTask *Configuration Performance*

Table I showcases the performance of state-of-the-art (SOA) models for task-oriented object detection on COCO tasks, including our iTask implementation and its variations. Among these, iTask* (YOLO) achieves the highest average performance (45.01), demonstrating superior task-specific accuracy while maintaining competitive results across

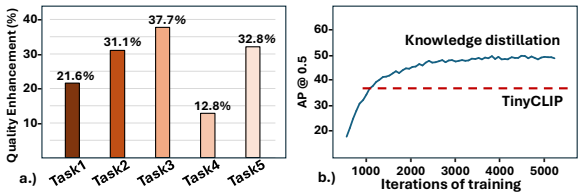


Fig. 6: (a) The graph shows the quality improvement of the student model implemented using knowledge distillation compared to tinyCLIP, (b) convergence of TinyCLIP and Knowledge distillation at Task 1.

all tasks. This establishes our iTask framework as a leader in the SOA for task-oriented object detection.

Our iTask implementation includes two configurations: a region-based RCNN approach and an efficient YOLO-based single-shot detector. iTask (YOLO) achieves an average performance of 44.51, while iTask* (YOLO) further improves accuracy with task-specific tuning. These results highlight the robustness and adaptability of iTask across diverse detection scenarios.

With iTask setting the SOA, the next step focuses on quantized versions of the model to reduce computational cost, memory usage, and inference time. The goal is to create lightweight, fast, and accurate configurations suitable for deployment in resource-constrained environments such as edge devices and real-time applications, extending the impact of iTask to practical use cases.

The lightweight version, iTask Tiny, introduces a trade-off between accuracy and computational efficiency. Across tasks, iTask Tiny configurations exhibit a quality loss ranging from approximately 8% to 12% relative to iTask (YOLO), depending on the specific quantization settings. For example, more aggressive quantization configurations like (8-4) experience higher performance degradation, while less aggressive configurations such as (10-16) achieve reduced quality loss. Despite this degradation, iTask Tiny retains sufficient accuracy for many task-specific applications, providing a highly efficient alternative for deployment in resource-constrained environments.

This quality loss reflects the inherent challenges in balancing lightweight design and task-specific accuracy. However, iTask Tiny demonstrates that, through careful quantization and optimization, lightweight models can still deliver effective task-oriented object detection while significantly reducing computational requirements. These results highlight the potential for further refinement of quantized models to bridge the gap between efficiency and performance for real-world systems.

B. *Knowledge Distillation Results*

Figure 5 illustrates examples of task-oriented object detection using iTask for the lightweight version of our model. Specific objects are identified based on the context of various tasks, such as *'Strike ball,' 'Pouring,' 'Shaking,' 'Eating,' 'Throwing,'* and *'Riding.'* Detected objects are highlighted in blue, demonstrating the model's ability to perform context-aware object recognition tailored to task-specific requirements. This contextual adaptability underscores the effectiveness of the lightweight iTask framework in aligning object detection with task-specific affordances.

Moving to the quantitative analysis, Figure 6 provides deeper insights into the performance improvements achieved through knowledge distillation. Subfigure (a) compares the quality enhancement of EfficientViT, implemented using knowledge distillation, against TinyCLIP. The results clearly show that knowledge distillation leads to a significantly higher ceiling in terms of performance, with EfficientViT outperforming TinyCLIP in achieving task-specific quality

TABLE I: Comparison of different models across tasks with iTask variations.

Task	GGNN [27]	TOIST [28]	TOIST* [28]	iTask (RCNN) [1]	iTask (YOLO) [1]	iTask* (YOLO) [1]
1	36.6	44.0	45.8	44.23	44.23	48.31
2	29.8	39.5	40.0	43.71	43.71	46.23
3	17.2	23.5	26.9	31.65	31.64	30.71
4	43.6	52.8	58.3	65.54	65.54	62.02
5	21.0	23.0	32.5	37.46	37.46	37.74
Mean	29.64	36.56	40.7	44.52	44.51	45.01

TABLE II: Comparisons to the prior AI accelerators.

	A^3 [29]	SpAtten [30]	Sanger [31]	ViTCoD [32]	ReDAS [33]	This Work
Target AI Model	Transformer	Sparse Transformer	Sparse Transformer	Sparse ViT	General Purpose	ViT + CNN
Technology (nm)	40	40	55	28	28	28
Precision	INT9	INT12	4 & 16 bits	Not specified	INT8	INT8
Frequency (MHz)	1000	1000	500	500	1150	1850
Area (mm^2)	2.08	18.71	16.9	3.0	20.77	6.14
Power (mW)	110.422	2600*	2760	323.9	-	8330
Peak Performance (TOPS)	0.22	2.00	2.56	0.256	18.84	30.31
Compute Density (TOPS / mm^2)	0.11	0.11	0.15	0.09	0.90	4.93
Energy Efficiency (TOPS / W)	1.99	0.77	0.93	0.79	-	3.64

* Execluded the DRAM power for fair comparison.

TABLE III: Hardware Specifications.

Parameter	Array Size	Total Area	Max Power	Max Frequency	Max Throughput
Value	128×128	$6.14\ mm^2$	$8.33\ W$	$1.85\ GHz$	30.31 TOPS

TABLE IV: Model configurations for hardware evaluation

Label	ViT	CNN	Param. Count (M)
A	TinyClip8	Resnet-18	19.96
B	TinyClip8	Faster R-CNN*	21.54
C	EfficientViT-m0	YOLOv4-tiny	24.41
D	TinyClip8	YOLOv4-tiny	24.13
E	EfficientViT-m5	Resnet-18	59.41
F	TinyClip32	Faster R-CNN*	74.67
G	EfficientViT-m5	YOLOv4-tiny	63.59
H	TinyClip32	YOLOv4-tiny	77.27

*Resnet-50 backbone (by fourth bottleneck block) used.

improvements. For instance, while both models improve over time, EfficientViT exhibits a sharper and more consistent quality enhancement, demonstrating its capability to bridge the gap between lightweight design and task-specific accuracy.

Subfigure (b) highlights the convergence behavior of TinyCLIP and the knowledge-distilled EfficientViT for Task 1. Knowledge distillation demonstrates not only faster convergence but also superior final performance compared to TinyCLIP. This result underscores the effectiveness of leveraging a teacher-student paradigm to enhance the compact model's capacity for complex visual understanding.

Overall, knowledge distillation enables lightweight models like EfficientViT to achieve task-specific accuracy comparable to larger models while maintaining efficiency. This approach ensures a robust and scalable solution for edge computing scenarios, as it balances computational constraints with the precision required for task-oriented object detection.

C. Energy Efficiency and Speedup Analysis

To evaluate compute performance, we configured the ViT/CNN pairs within the iTask framework as detailed in Table IV, with corresponding compute latencies presented in Fig. 7. The results demonstrate that our accelerator efficiently accommodates diverse configurations, spanning varying parameter counts and model architectures. This flexibility facilitates the selection of optimal configurations tailored to specific tasks and constraints, enhancing scalability and leveraging the strengths of both ViT and CNN configurations across a range of deployment scenarios.

Table II compares the state-of-the-art (SOTA) AI accelerators, showing our accelerator has better compute density and energy efficiency

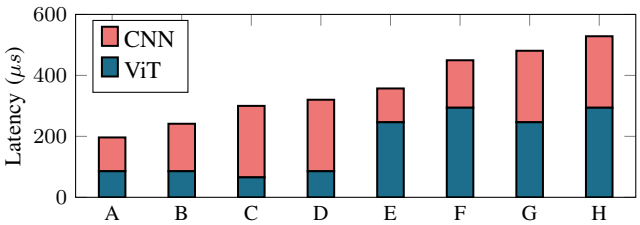


Fig. 7: The latency breakdowns on CNN/ViT architectures executed on the hardware accelerator.

than other works. This result originates from the dedicated design discipline for the iTask acceleration, including only the necessary modules and replacing on-chip SRAM with ping-pong SRs. At the same time, other accelerators adopted hardware sparse attention support to reduce the required throughput itself by pruning [30]–[32] or aware of general-purpose acceleration for most AI-related computations [33]. Another reason is that the newer semiconductor technology node, $28nm$, is used to synthesize our accelerator.

VI. Conclusions

In conclusion, iTask delivers a powerful framework for task-oriented object detection by integrating hardware acceleration, flexible model configurations, and situational awareness. Our hardware acceleration circuit significantly reduces latency and energy costs, making edge deployment feasible for real-time applications. The dual configurations—knowledge distillation for task-specific accuracy and quantization for versatile, multi-task generalization—offer adaptable performance that meets diverse task requirements without excessive computational demands. By enabling intelligent sensing, iTask selectively processes relevant data, optimizing resource use and providing an efficient, context-aware solution tailored for resource-constrained environments, from autonomous systems to real-time surveillance.

Acknowledgements

This work was supported in part by the DARPA Young Faculty Award, the National Science Foundation (NSF) under Grants #2127780, #2319198, #2321840, #2312517, and #2235472, the Semiconductor Research Corporation (SRC), the Office of Naval Research through the Young Investigator Program Award, and Grants #N00014-21-1-2225 and #N00014-22-1-2067, Army Research Office Grant #W911NF2410360. Additionally, support was provided by the Air Force Office of Scientific Research under Award #FA9550-22-1-0253, along with generous gifts from Xilinx and Cisco.

REFERENCES

[1] H. Chen, W. Huang, Y. Ni, S. Yun, Y. Liu, F. Wen, A. Velasquez, H. Latapie, and M. Imani, "Taskclip: Extend large vision-language model for task oriented object detection," *arXiv preprint arXiv:2403.08108*, 2024.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *CoRR*, vol. abs/2005.12872, 2020.

[3] G. Pantazopoulos, J. Bruyere, M. Nikandrou, T. Boissier, S. Hemanthage, B. K. Sachish, V. Shah, C. Dondrup, and O. Lemon, "Vica: Combining visual, social, and task-oriented conversational ai in a healthcare setting," in *Proceedings of the 2021 International Conference on Multimodal Interaction*, ICMI '21, (New York, NY, USA), p. 71–79, Association for Computing Machinery, 2021.

[4] J. Sawatzky, Y. Souri, C. Grund, and J. Gall, "What Object Should I Use? - Task Driven Object Detection," in *CVPR*, 2019.

[5] J. Tang, G. Zheng, J. Yu, and S. Yang, "Cotdet: Affordance knowledge prompting for task driven object detection," 2023.

[6] A. Kamath, M. Singh, Y. LeCun, G. Synnaeve, I. Misra, and N. Carion, "Mdetr-modulated detection for end-to-end multi-modal understanding," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1780–1790, 2021.

[7] C. Xie, Z. Zhang, Y. Wu, F. Zhu, R. Zhao, and S. Liang, "Described object detection: Liberating object detection with flexible expressions," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[8] Y. D. Kwon, R. Li, S. I. Venieris, J. Chauhan, N. D. Lane, and C. Mascolo, "Tinytrain: Resource-aware task-adaptive sparse training of dnns at the data-scarce edge," 2024.

[9] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019.

[10] K. Wu, H. Peng, Z. Zhou, B. Xiao, M. Liu, L. Yuan, H. Xuan, M. Valenzuela, X. S. Chen, X. Wang, H. Chao, and H. Hu, "Tinyclip: Clip distillation via affinity mimicking and weight inheritance," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 21970–21980, October 2023.

[11] P. Zhang, X. Li, X. Hu, J. Yang, L. Zhang, L. Wang, Y. Choi, and J. Gao, "Vinvl: Revisiting visual representations in vision-language models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5579–5588, 2021.

[12] M.-A. Tnani, P. Subarnaduti, and K. Diepold, "Efficient feature learning approach for raw industrial vibration data using two-stage learning framework," *Sensors*, vol. 22, no. 13, p. 4813, 2022.

[13] M. Tan, R. Pang, and Q. V. Le, "Efficientdet: Scalable and efficient object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10781–10790, 2020.

[14] J. Wang, X. Hu, P. Zhang, X. Li, L. Wang, L. Zhang, J. Gao, and Z. Liu, "Minivlm: A smaller and faster vision-language model," *arXiv preprint arXiv:2012.06946*, 2020.

[15] C. Feng, Y. Zhong, Y. Gao, M. R. Scott, and W. Huang, "Tood: Task-aligned one-stage object detection," *arXiv preprint arXiv:2108.07755*, 2021.

[16] Y. Wang, J. Guo, R. Wang, W. He, and C. Li, "Tienet: task-oriented image enhancement network for degraded object detection," *Signal, Image and Video Processing*, vol. 18, no. 1, pp. 1–8, 2024.

[17] C. Liu, H. Li, S. Wang, M. Zhu, D. Wang, X. Fan, and Z. Wang, "A dataset and benchmark of underwater object detection for robot picking," in *2021 IEEE international conference on multimedia & expo workshops (ICMEW)*, pp. 1–6, IEEE, 2021.

[18] W. Huang, A. Rezvani, H. Chen, Y. Ni, S. Yun, S. Jeong, and M. Imani, "A plug-in tiny ai module for intelligent and selective sensor data transmission," *arXiv preprint arXiv:2402.02043*, 2024.

[19] H. Zhang, J. Na, and B. Zhang, "Autonomous internet of things (iot) data reduction based on adaptive threshold," *Sensors*, vol. 23, no. 23, p. 9427, 2023.

[20] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

[21] L. Yang and A. Shami, "Iot data analytics in dynamic environments: From an automated machine learning perspective," *Engineering Applications of Artificial Intelligence*, vol. 116, p. 105366, 2022.

[22] A. Taha, M. Alrabeiah, and A. Alkhateeb, "Enabling large intelligent surfaces with compressive sensing and deep learning," *IEEE access*, vol. 9, pp. 44304–44321, 2021.

[23] Z. Ballard, C. Brown, A. M. Madni, and A. Ozcan, "Machine learning and computation-enabled intelligent sensor design," *Nature Machine Intelligence*, vol. 3, no. 7, pp. 556–565, 2021.

[24] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[25] A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina, and T. Krishna, "A systematic methodology for characterizing scalability of dnn accelerators using scale-sim," in *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 58–68, IEEE, 2020.

[26] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: Constructing Hardware in a Scala Embedded Language," in *Proceedings of the 49th Annual Design Automation Conference (DAC)*, (San Francisco, CA, USA), p. 1216–1225, 2012.

[27] J. Sawatzky, Y. Souri, C. Grund, and J. Gall, "What object should i use?-task driven object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7605–7614, 2019.

[28] P. Li, B. Tian, Y. Shi, X. Chen, H. Zhao, G. Zhou, and Y.-Q. Zhang, "Toist: Task oriented instance segmentation transformer with noun-pronoun distillation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17597–17611, 2022.

[29] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee, and D.-K. Jeong, "A^3: Accelerating Attention Mechanisms in Neural Networks with Approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, (San Diego, CA, USA), pp. 328–341, IEEE, Feb. 2020.

[30] H. Wang, Z. Zhang, and S. Han, "SpAtten: Efficient Sparse Attention Architecture with Cascade Token and Head Pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (Seoul, Korea (South)), pp. 97–110, IEEE, Feb. 2021.

[31] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A Co-Design Framework for Enabling Sparse Attention using Reconfigurable Architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, (Virtual Event Greece), pp. 977–991, ACM, Oct. 2021.

[32] H. You, Z. Sun, H. Shi, Z. Yu, Y. Zhao, Y. Zhang, C. Li, B. Li, and Y. Lin, "ViTCoD: Vision Transformer Acceleration via Dedicated Algorithm and Accelerator Co-Design," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, (Montreal, QC, Canada), pp. 273–286, IEEE, Feb. 2023.

[33] M. Han, L. Wang, L. Xiao, T. Cai, Z. Wang, X. Xu, and C. Zhang, "ReDas: A Lightweight Architecture for Supporting Fine-Grained Reshaping and Multiple Dataflows on Systolic Array," *IEEE Transactions on Computers*, vol. 73, pp. 1997–2011, Aug. 2024.

EVA: An Efficient and Versatile Generative Engine for Targeted Discovery of Novel Analog Circuits

Jian Gao[1], Weimin Fu[2], Xiaolong Guo[2], Weidong Cao[3], Xuan Zhang[1]
[1]Northeastern University; [2]Kansas State University; [3]The George Washington University

***Abstract*—Analog circuit design has traditionally depended on manual expertise, slowing the discovery of novel topologies essential for advanced technologies like AI, 5G/6G, and quantum computing. While AI-driven methods have accelerated hardware design workflows, most of them focus on topology synthesis, often reusing known structures to achieve specific goals. The challenge of discovering entirely new, high-performance topologies remains largely underexplored due to its abstract nature. In this work, we introduce EVA, an efficient and versatile generative engine for discovering novel analog circuit topologies. EVA employs a bottom-up generation framework, using a decoder-only transformer to sequentially predict device pin connections and create diverse circuits from scratch. Pretraining on unlabeled circuit topologies builds foundational knowledge about circuit connectivity, achieving baseline discovery efficiency by generating valid circuits and reducing performance-labeled samples needed in fine-tuning. For targeted discovery of high-performance designs, EVA leverages two fine-tuning strategies—proximal policy optimization (PPO) and direct preference optimization (DPO)—to further enhance discovery efficiency for relevant, high-performing topologies. Experimental results across various circuit types highlight EVA's strengths in validity, novelty, versatility, and both training sample and discovery efficiency.**

I. Introduction

Analog circuit topologies' design has historically relied heavily on manual effort and domain expertise. This approach not only results in a lengthy design cycle but also restricts the discovery of novel analog circuit topologies. Emerging technologies, such as AI, 5G/6G, and quantum computing, impose stringent performance demands, creating a need for innovative analog circuit topologies beyond the capabilities of well-established designs. To support the advancement of these technologies, a critical question arises: how can we efficiently explore and invent circuit topologies that might achieve unprecedented performance, surpassing what human expertise alone has accomplished? Recently, AI-driven methodologies for analog hardware design have shown promise in accelerating the design cycle across various stages of the design flow [1]–[10]. However, existing research [11]–[13] mainly addresses the analog circuit topology synthesis problem, focusing on meeting specific design objectives by selecting or reusing known topologies or sub-blocks from existing designs. In contrast, the discovery problem of novel analog circuit topologies remains underexplored due to its abstract and complex nature.

In this work, we tackle the novel circuit topology discovery problem due to its scientific significance and potential for breakthrough innovations. Existing approaches have achieved great progress in topology discovery for certain types of analog circuits. CktGNN [1], a pioneering effort, employs a variational autoencoder (VAE) with a two-level graph neural network (GNN) to learn and generate diverse operational amplifier (Op-Amp) topologies. Similarly, Karahan et al. [14] propose an inverse design approach for the rapid synthesis of complex RF-to-terahertz (THz) matching networks by generating various electromagnetic (EM) structures. However, these approaches are limited in versatility and discovery efficiency without the ability to discover diverse types of analog circuit topologies while targeting certain performance preferences.

To address these challenges, we introduce EVA, an efficient and versatile generative engine for discovering novel analog circuit topologies. To achieve great versatility, EVA proposed a bottom-up generation framework that uses a decoder-only transformer to generate diverse analog designs from scratch, starting from a VSS pin and sequentially predicting the next device pin connection. Instead of training with performance-labeled topology from scratch, EVA is first pre-trained on a diverse set of unlabeled circuit topologies to learn the circuit connectivity. In this way, the model itself has already achieved a basic level of discovery efficiency by generating valid topology and setting a great foundation for finetuning. For targeted discovering specific circuit types with high performance within limited attempts, EVA proposed two fine-tuning strategies based on proximal policy optimization (PPO) [15] and direct preference optimization (DPO) [16] with a limited number of labeled topology thanks to the foundation laid by the pretrained model. In this way, EVA is able to achieve great training sample efficiency and discovery efficiency without wasting evaluation time on irrelevant or low-performing circuits. Our key contributions are as follows:

- EVA is a **versatile generative engine** that can discover diverse, novel analog circuit topologies by sequentially predicting device pin connections, starting from VSS.
- EVA pretrains a decoder-only transformer on an unlabeled dataset of circuit topologies, enabling efficient generation of valid circuits and significantly **improving training sample efficiency** by reducing the reliance on performance-labeled samples during fine-tuning.
- EVA **enhances discovery efficiency** by fine-tuning with PPO and DPO on a small set of performance-labeled topologies, focusing on high-performance circuits and minimizing irrelevant evaluations.

This work is partially supported by NSF #2416375 and NSF #2526432.

- Experimental results across various analog circuit types demonstrate EVA's advantages in topology validity, novelty, versatility, sample efficiency, and discovery efficiency.

II. Related Work

A. ML methods for analog circuit topology synthesis

Analog circuit topology synthesis aims to develop circuits that meet design goals using established topologies or subblock topologies. Existing ML methods primarily address the issue of data scarcity when building models capable of automating analog circuit design. AnalogCoder [11] highlights that while large language models (LLMs) have been extensively trained on Python code, they lack exposure to the SPICE netlist, the industry-standard format for describing analog circuits. To bridge this gap, they employ domain-specific prompt engineering, enabling LLMs to generate PySpice code that can be seamlessly converted into SPICE netlists without additional training—thus maximizing training sample efficiency. Similarly, Artisan [12] translates SPICE netlists into natural language, allowing the development of domain-specific foundation models for analog circuit design, even with limited data. On the other hand, LaMAGIC [13], a fine-tuned masked language model (MLM), has been proposed to generate power converter circuit topologies by predicting the connections between predefined nodes. However, most current approaches to analog circuit topology design rely heavily on existing circuit topologies or subblock structures, which constrains their ability to discover novel topologies.

B. ML methods for novel analog circuit topology discovery

Novel analog circuit topology discovery aims to identify novel, non-existent circuit topologies that could potentially expand our understanding of analog circuits. Existing ML approaches have primarily focused on discovering designs at two distinct levels, each restricted to a specific type of analog circuit. CktGNN [1] addresses the schematic level by generating a variety of Op-Amp topologies. This method adopts a top-down graph generation approach but limits the design scope to sub-blocks without incorporating device-level details. While this constraint simplifies the generation process and ensures valid topologies (e.g., avoiding floating nodes), it also restricts the diversity of circuits that can be explored. Karahan et al. [14], on the other hand, focus on the physical level, employing an inverse-design approach using binary matrices to generate unintuitive EM structures for RF-to-THz matching networks. However, applying this method to schematic-level design is challenging, as the complexity of analog circuit topologies—due to the wide variety of device types and numbers—cannot be adequately captured by a binary matrix representation. Beyond versatility limitations, these methods are also constrained by inefficient discovery processes. Both approaches lack target discovery capabilities, relying heavily on trial and error, which leads to substantial evaluation overhead from real-world simulators due to frequent low-performance circuit assessments. To mitigate this, Karahan et al. proposed training a forward model to predict performance for specific EM structures, a promising direction but one that demands over 600k performance-labeled EM structures, making it extremely sample inefficient.

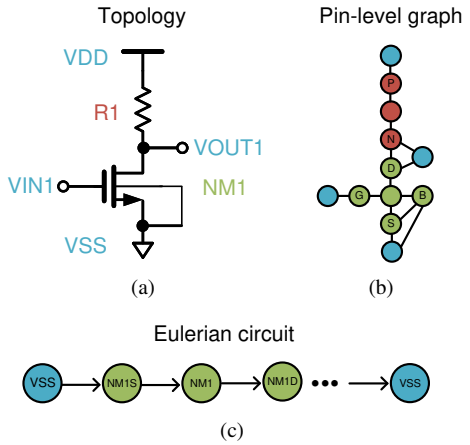


Fig. 1: An example of sequential graph presentation EVA used to represent the circuit topology [17]. (a) Original topology. (b) Pin-level graph representation. (c) Eulerian circuit.

III. EVA Framework

EVA in Figure 2 is an efficient and versatile generative engine designed to discover novel analog circuit topologies. It uses a device pin-level graph representation [17], sequentialized as an Eulerian circuit, to capture all possible connections in a compact format while retaining critical structural information. EVA leverages a decoder-only transformer architecture, pre-trained on diverse analog circuit topologies, to generate circuits from scratch by sequentially predicting device pin connections. Through fine-tuning strategies such as PPO and DPO, EVA efficiently generates high-performance circuit topologies with minimal labeled data, targeting specific circuit types and desired performance.

A. Expressive and efficient sequential graph representation

Existing approaches to schematic-level analog circuit topology discovery generally formulate the problem as a graph generation task, where each circuit topology is represented as a device-level graph with nodes corresponding to individual devices (e.g., NMOS transistors) [1]. EVA, in contrast, adopts a device pin-level graph representation as shown in Figure 1 where each node corresponds to an individual device pin (e.g., NM1_G, NM1_D, NM1_S, and NM1_B) [17]. Since analog circuit topologies are typically sparse—most devices connect only to their immediate neighbors—traditional adjacency matrices are inefficient, as they waste space representing non-existent edges [1], [13]. To address this, EVA sequentializes the graph as an Eulerian circuit [17], a path that traverses each directed edge exactly once before returning to the starting node (e.g., VSS). Unlike prior approaches that limit graph structures to directed acyclic graphs (DAGs) [1]—which cannot universally represent all analog topologies—the Eulerian circuit

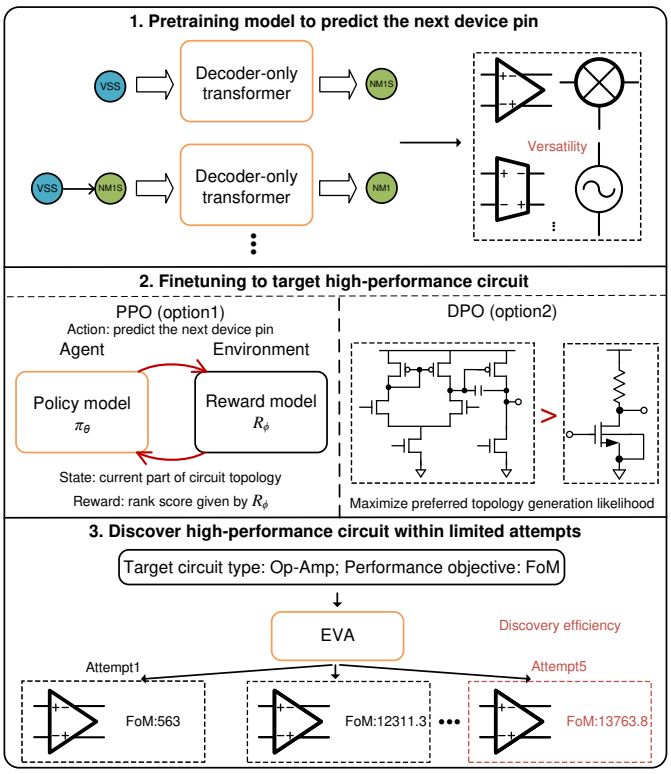


Fig. 2: Overview of EVA framework.

approach is more versatile, capable of representing any analog circuit that can be modeled as a finite connected undirected graph.

B. Pretraining model to generate diverse circuits

EVA employs a customized domain-specific tokenizer to encode and decode sequences for pretraining a decoder-only transformer to discover a wide range of analog circuit topologies. Each token in EVA's tokenizer corresponds either to a device pin (e.g., NM1_G, NM1_D, NM1_S, NM1_B) or to a circuit-level pin for the overall analog topology (e.g., VIN1, VOUT1, VDD, VSS). To ensure the model can generalize across circuit topologies with varying numbers and types of devices, EVA uses a data-driven approach to scan the entire dataset and determine device limits. A special token, "Truncate," is introduced to handle padding when sequences of varying lengths are processed.

With this domain-specific tokenizer, EVA pre-trains a decoder-only transformer to predict the next device pin within a circuit topology. Unlike typical language model pretraining, which crops sequences randomly from text, EVA ensures that each sequence corresponds to a single complete circuit topology. Given an corpus of tokens $\mathcal{U} = \{u_1, \ldots, u_n\}$ representing one circuit topology, EVA maximizes the standard language modeling objective [18] to train the model.

$$\mathcal{L}_{\text{pretrained}}(\mathcal{U}) = \sum_i \log P\left(u_i \mid u_{i-k}, \ldots, u_{i-1}; \Theta\right) \quad (1)$$

TABLE I: Rank score definitions for PPO finetuning

Reward	Definition
1.0	High performance relevant valid circuit
0.5	Low performance relevant valid circuit
-0.5	Irrelevant valid circuit
-1.0	Invalid circuit

Here, k represents the size of the context window, and the conditional probability P is modeled by a neural network with parameters Θ. During the generation phase, the model is initially provided with just one context token, "VSS," which serves as the starting node for all Eulerian circuits. From this single token, the model generates the rest of the sequence, completing it to represent an entire circuit topology.

C. Finetuning model to target high performance circuit

The pre-trained model in EVA has developed foundational knowledge of circuit connections across various circuit types, enabling it to achieve baseline discovery efficiency by generating valid circuits. To further enhance its ability to efficiently generate high-performance circuits within limited attempts, EVA fine-tunes this model to align with human preferences for both type and performance. Building on the foundation provided by pretraining, EVA can leverage a small set of performance-labeled topologies to guide generation towards high-performance designs. EVA introduces two fine-tuning strategies that balance training stability and overfitting.

1) Proximal policy optimization: EVA adopts PPO [15] as one of its reinforcement learning with human feedback (RLHF) strategies. PPO uses limited labeled samples to train a reward model as a labeler, enabling the pre-trained model to generate new samples, which are then labeled by the reward model for fine-tuning based on human preferences. Specifically, EVA utilizes a Plackett-Luce ranking model [19] for sequence reward, ranking multiple classes as shown in Table I. The reward model combines a rule-based checker with a multiclass classifier: it first checks if a generated circuit is valid (i.e., simulatable with default sizing) and then classifies circuit type and performance. To train this multiclass classifier, a limited set of circuit topologies is initially labeled for type and performance. For instance, if EVA aims to discover high-performance Op-Amps, only Op-Amps are relevant, while other types are irrelevant. The figure of merit (FoM) determines whether relevant circuits are high-performance, with Otsu's method [20] setting the FoM threshold. Once labeled, EVA adds a three-output linear layer on top of the final transformer layer, outputting a reward value after softmax.

With the trained reward model, PPO performs optimization as shown in Algorithm 1. The major components of PPO are defined as follows:

Agent: The agent is the policy model π_θ, which is the pre-trained model fine-tuned to align with human preferences. EVA adds a value head to it, which consists of a simple linear layer that takes hidden states as input and outputs a single scalar per token. The value function provides an estimate of the expected cumulative reward from any given state.

Algorithm 1 PPO RLHF Finetuning

Require: Policy model π_θ, Reference model $\pi_{\theta_{ref}}$, Reward model R_ϕ, Epochs N_{epochs}, PPO epochs N_{ppo}, Batch size D, Minibatch size B, Value loss coefficient vc

for epoch $= 1$ to N_{epochs} **do**
  Let π_θ generate a batch of sequences with size D and corresponding values and store them as x, y, V
  Compute PPO rewards $\mathbf{r}$ from Eq. (2)
  for ppo_epoch $= 1$ to N_{ppo} **do**
    Randomly sample minibatch $(x_b, y_b, V_b, \mathbf{r}_b)$
    Repredict $V_{b_{new}}$ for minibatch using π_θ
    Calculate $\mathcal{L}_{\text{policy}}$, $\mathcal{L}_{\text{value}}$ from Eq. (3) and Eq. (4)
    Calculate $\mathcal{L}_{\text{PPO}} = -\mathcal{L}_{\text{policy}} + vc \cdot \mathcal{L}_{\text{value}}$
    Backpropagate and update π_θ
  end for
end for

Environment: The environment is the reward model R_ϕ. This environment assigns rewards based on whether generated sequences meet criteria defined in Table I.

Action: The action is discrete by selecting a token y_t (i.e., device-pin) from a pre-defined tokenizer's look-up table based on the current context x_t (i.e., generated sequence).

State: The state is the current context x_t.

Reward: Starting from the initial state x_0 (i.e., "VSS"), the RL agent takes a discrete action by predicting the next token y_t given the current state x_t in each time step t, and the environment returns a reward r_t at that step. EVA defines the PPO reward function as follows:

$$r_t = R_\phi(x, y) - \beta \log \left(\frac{\pi_\theta(y_t \mid x_t)}{\pi_{\theta_{ref}}(y_t \mid x_t)} \right) \tag{2}$$

Here, $R_\phi(x, y)$, is the reward model's score that measures how well the entire sequence aligns with preference defined in Table I. The second term is a penalty that discourages π_θ from deviating too much from $\pi_{\theta_{ref}}$, with β controlling the strength of this penalty. The reference model $\pi_{\theta_{ref}}$ is a fixed version of the pre-trained language model that the policy model was initially based on.

In PPO, the entire sequence or circuit topology acts as a trajectory, with the objective of optimizing the policy model π_θ to generate sequences that maximize the expected accumulated reward via the advantage A and the policy loss $\mathcal{L}_{\text{policy}}$. Concurrently, PPO refines value prediction $V_{b_{\text{new}}}$ using the value loss $\mathcal{L}_{\text{value}}$. As shown in Algorithm 1, each epoch begins with PPO generating rollouts (sequences) using the policy model, and the reward model R_ϕ assigns rewards $\mathbf{r}$, where each r corresponds to a rollout. PPO then performs minibatch optimization. The policy loss $\mathcal{L}_{\text{policy}}$ is designed to maximize the expected advantage of the actions taken by the agent while preventing large, destabilizing updates to the policy.

$$\begin{aligned} \mathcal{L}_{\text{policy}} &= \mathbb{E}_t \left[\min \left(L \cdot A_t, \text{clip}(L, 1-\epsilon, 1+\epsilon) \cdot A_t \right) \right] \\ \text{where} \quad L &= \frac{\pi_\theta \left(y_{b_t} \mid x_{b_t} \right)}{\pi_{\theta_{\text{old}}} \left(y_{b_t} \mid x_{b_t} \right)} \end{aligned} \tag{3}$$

Here, advantages $A_t = \delta_t + \gamma\lambda \cdot A_{t+1}$ where $\delta_t = r_{b_t} + \gamma V_b \left(x_{b_{t+1}} \right) - V_b \left(x_{b_t} \right)$ measure how much better (or worse) an action at time t is compared to the expected outcome calculated from value function $V_b(x_{b_{t+1}})$. L represents the probability ratio between the current and previous policies during minibatch optimization. With the clip objective, the policy loss in PPO strikes a balance between exploiting the current advantage—encouraging the model to increase the probability of actions that yield higher expected rewards—and maintaining stable, incremental updates to the policy. In PPO, the value loss $\mathcal{L}_{\text{value}}$ is calculated to enhance the accuracy of the model's value function.

$$\mathcal{L}_{\text{value}} = \frac{1}{2} \cdot \left(V_{b_{new}} \left(x_{b_t} \right) - G_t \right)^2 \tag{4}$$

The value loss $\mathcal{L}_{\text{value}}$ is the squared difference between the newly predicted value $V_{b_{\text{new}}} \left(x_{b_t} \right)$ and the target return G_t. The $G_t = A_t + V_b \left(x_{b_t} \right)$ represents the total expected reward for the trajectory from t onward, including the estimated value of future states. By incorporating both immediate and future rewards, G_t is a robust target for updating the value function.

2) Directed preference optimization: PPO is effective in preventing overfitting by continually generating new training data [21]. However, its complexity and stability issues arise from its sensitivity to hyperparameters and dependence on a reward model. To address these challenges in a simpler and more stable way, EVA introduces a finetuning strategy based on DPO [16]. Unlike PPO, which requires a separate reward model to label newly generated data, DPO finetunes the pre-trained model using a static dataset with expert-labeled data. This approach enhances stability, as the training labels are accurate and unaffected by the reward model's classification reliability. Moreover, DPO's objective function is simplified compared to PPO, with only a single hyperparameter, β, which controls the extent of π_θ deviation from the reference model.

$$\begin{aligned} &\mathcal{L}_{\text{DPO}} \left(\pi_\theta; \pi_{\text{ref}} \right) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \\ &\left[\log \sigma \left(\beta \log \frac{\pi_\theta \left(y_w \mid x \right)}{\pi_{\text{ref}} \left(y_w \mid x \right)} - \beta \log \frac{\pi_\theta \left(y_l \mid x \right)}{\pi_{\text{ref}} \left(y_l \mid x \right)} \right) \right] \end{aligned} \tag{5}$$

DPO utilizes the Bradley-Terry model [22] as its preference framework, focusing on pairwise comparisons (win or lose) rather than the multi-rank model employed in PPO. Specifically, optimizing $\mathcal{L}_{\text{DPO}}$ guides the pre-trained model to identify analog circuit topologies that maximize the expression $\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$, thus encouraging it to generate "winning" topologies while discouraging "losing" ones. Given the complexity of human preferences for analog circuit topologies, EVA still relies on human experts to label circuit topologies according to a multi-rank model, as defined in Table I. For any four data points where each belongs to a unique class, EVA transforms these into six unique win-lose pairs for DPO training (e.g., High performance relevant valid circuit $>$ Low performance relevant valid circuit, High performance relevant valid circuit $>$ Irrelevant valid circuit, etc.).

IV. Experiments

A. Experiments Setup

Datasets: The EVA circuit dataset comprises 3470 unique and real-world topologies across 11 circuit types: Op-Amps, LDOs, Bandgap references, Comparators, PLLs, LNAs, PAs, Mixers, VCOs, Power converters, and Switched Capacitor Samplers from public resources [23]–[28]. Each circuit's performance was assessed through circuit simulation, and a corresponding label was assigned based on its performance metrics and label definition defined in Table I.

EVA training setup: EVA splits the topology dataset into training and validation sets with a 9:1 ratio, ensuring the validation topologies remain unseen during training. EVA uses a depth-first search (DFS) to identify the Eulerian circuit for each topology. To prevent overfitting, it permutes DFS traversal order, creating multiple unique Eulerian circuits per topology, expanding the original 3470 topologies to 234393 sequences. EVA's generation model is a decoder-only transformer with 6 layers, 6 attention heads, and 11.825 million parameters, with a vocabulary size of 1029 and maximum sequence length of 1024. The reward model architecture is detailed in Section III-C1.

Baseline: To compare with EVA, we select AnalogCoder [11], Artisan [12], and LaMAGIC [13] as the representative ML baselines for analog circuit topology synthesis and select CktGNN [1] as the representative ML baselines for novel analog circuit topology discovery at the schematic level. The differences between these methods and EVA are discussed in Section II. We follow original work to produce their results.

Evaluation tasks and metrics: We evaluate generative quality across the following metrics: (1) Validity: An unsized circuit is valid if it can be simulated in SPICE without errors (e.g., floating or shorting nodes). Each method generates 1000 topologies, and we report the percentages that are valid. (2) Novelty: Each method generates 1000 topologies, and we measure the percentage of them that are different from the topologies in the dataset. To quantify the topology difference, We converted them into graphs and computed the maximum mean discrepancy (MMD) [29] between these and real-world graphs converted from our circuit dataset. (3) Versatility: The number of distinct analog circuit types generated indicates versatility. (4) Training sample efficiency: Efficiency is evaluated by comparing the number of labeled topologies (i.e., with known performance) required for training. (5) Discovery efficiency: Given the time cost of performance evaluation, each method only generates 10 topologies, and we report the maximum FoM (e.g., gain, bandwidth, power for Op-Amps) after sizing with a genetic algorithm and SPICE evaluation.

B. Necessity of pretraining and finetuning

We first perform an ablation study to evaluate the necessity of both pretraining and fine-tuning. As shown in Figure 3, PPO results reveal that the pretrain-only model establishes a solid foundation in generating irrelevant valid circuits (score = -0.5) and low-performance relevant valid circuits (score = 0.5). However, it lacks the ability to specifically target high-performance relevant valid circuits (score = 1), necessitating further fine-tuning. Similar observations are evident in DPO results, where the pretrain-only model shows no strong preference for winning topologies, as indicated by 0 reward accuracy. This suggests the model predominantly generates losing topologies during generation, i.e., $\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} < \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$. In contrast, the finetune-only model lacks the foundational understanding of circuit connectivity, making it challenging to effectively train on a limited number of labeled samples to generate valid topologies. For PPO, EVA's vast design space (1029^{1024}), dominated by invalid topologies, leads to an extremely sparse reward signal (i.e., most rewards are suboptimal). As a result, guiding the finetune-only model to discover high-performance relevant valid circuits from scratch becomes infeasible using a simple reward model. On the other hand, DPO results indicate that even an untrained model can achieve reward accuracy exceeding 50%, highlighting a limitation in this metric. Specifically, the metric only measures the preference for generating winning topologies, $\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} > \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}$, without assessing the actual generation quality of winning topologies $\pi_\theta(y_w \mid x)$ or losing topologies $\pi_\theta(y_l \mid x)$. This limitation is further validated in Table II, where EVA (DPO only) fails to generate any valid topologies. In conclusion, EVA can effectively target high-performance relevant valid circuits only through a combination of pretraining and finetuning.

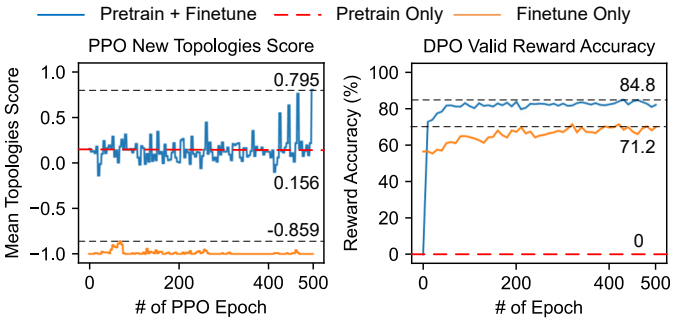


Fig. 3: PPO score and DPO validation reward accuracy comparison between Pretrain + Finetune, Pretrain only, and Finetune only while targeting Op-Amp design as an example.

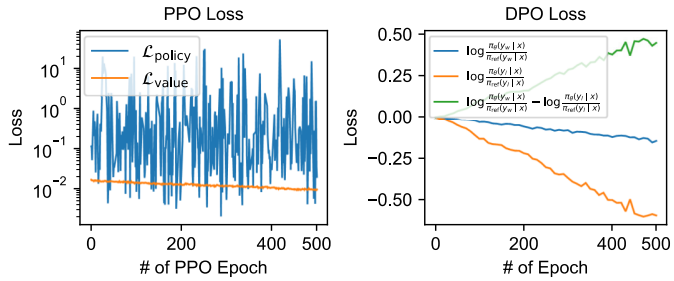


Fig. 4: EVA's PPO loss and DPO loss after pretraining while targeting Op-Amp design as an example.

TABLE II: Performance comparison between EVA and existing analog circuit topology generation work.

Evaluation metric	Validity (%) ↑	Novelty		Versatility ↑	# of labeled topology ↓		FoM@10 [12], [13] ↑	
		Diff circuit (%) ↑	MMD ↓		Op-Amp	Power converter	Op-Amp	Power converter
AnalogCoder [11]	66.1	0	0	7	11	N/A	232.1	N/A
Artisan [12]	82	0	0	1	14000	N/A	12769.5	N/A
CktGNN [1]	68.5	93	0.313	1	10000	N/A	311.3	N/A
LaMAGIC [13]	75	3	0.001	1	N/A	132000	N/A	2.2
EVA (Pretrain)	**84**	**99**	**0.0518**	**11**	**0**	**0**	**274.1**	**2.5**
EVA (PPO only)	**0**	**N/A**	**N/A**	**11[a]**	**850**	**362**	**N/A**	**N/A**
EVA (DPO only)	**0**	**N/A**	**N/A**	**11[a]**	**850**	**362**	**N/A**	**N/A**
EVA (Pretrain+PPO)	**94**	**99**	**0.0509**	**11[a]**	**850**	**362**	**13647.2**	**3.3**
EVA (Pretrain+DPO)	**83**	**99**	**0.0491**	**11[a]**	**850**	**362**	**13763.8**	**3.4**

[a] Targeting another circuit type requires redo fine-tuning, which takes less than an hour on an A100 GPU.

C. Comparison between PPO and DPO for EVA finetuning

We compare PPO and DPO for EVA fine-tuning in terms of training stability and generation quality. PPO, an online RL method, dynamically generates new topologies and samples rewards from the environment, but its training instability stems from the stochastic reward signal r_{b_t}, leading to noisy advantage estimates A_t and oscillations in $\mathcal{L}_{\text{policy}}$. In contrast, DPO, an offline RL method, optimizes the relative log-likelihood between winning and losing topologies using static preference data, avoiding noisy rewards. For generation, PPO demonstrates superior validity by optimizing for both preference and reward-based quality through continuous exploration of new topologies. This enables it to overcome degeneration issues and produce valid, diverse outputs that outperform the pretrained model. DPO, however, shifts the model's preference to high-performance topologies in the dataset without learning from new data, leading to degeneration in validity. This degeneration is characterized by a reduction in both winning and losing topology generation likelihoods in Figure 4, with the latter declining faster. At low learning rates, this degeneration is manageable, and our results focus on these settings, as shown in the Figure 4 and Table II. Higher learning rates, while not depicted, can cause the model to collapse, generating repetitive tokens. Despite these limitations, DPO excels in targeted discovery, producing higher-FoM topologies within 10 attempts and achieving lower MMD values, which indicate a closer resemblance to real-world circuits.

D. Comparison between EVA and prior arts

Finally, we compare EVA with prior methods:

Validity: EVA demonstrates higher validity than previous methods. Top-down approaches like CktGNN and LaMAGIC rely on predefined representations, which limit their generalization to other circuit types and restrict the range of topologies they can be trained on. In contrast, EVA's universal circuit representation and bottom-up generation framework enable pretraining on diverse analog circuits with more training data.

Novelty: EVA excels in discovering novel circuits compared to AnalogCoder and Artisan, which primarily reuse existing topologies or subblocks. LaMAGIC is confined to a small design space (fewer than 4 devices), limiting its ability to discover new topologies. While CktGNN and EVA both support larger circuits (around 20 and 60 devices, respectively), CktGNN is trained on synthetic datasets, which lack critical real-world features. EVA, trained on real-world circuits, improves MMD by over $6\times$ and generates around 99% novel circuits.

Versatility: EVA outperforms Artisan, CktGNN, and LaMAGIC, which can only design one type of circuit. While AnalogCoder supports seven circuit types, it is limited to a synthesis library of just 20 topologies. EVA, trained on a dataset with 11 circuit types and over 3000 topologies (minimum 30 per type), demonstrates unparalleled versatility.

Sample efficiency: Unlike prior work that typically uses performance-labeled topology to train a model from scratch, EVA first pretrains on an unlabeled dataset to build a strong foundation in circuit connectivity. This enables efficient finetuning with only a small labeled dataset, achieving up to $364\times$ greater training sample efficiency compared to other methods.

Discovery efficiency: CktGNN and EVA (Pretrain) lack targeted discovery, often producing low-performance circuits within 10 attempts. AnalogCoder's synthesis library only contains simple circuits, which also limits the FoM of the circuit it synthesized. After finetuning, EVA is able to align with human preference and target high-performance circuits within limited attempts. Combining the foundation knowledge it learned from all sorts of analog circuit types during pretraining, EVA is able to discover unseen topologies with superior FoM compared to dedicated synthesizers like Artisan and LaMAGIC, excelling in both Op-Amp and power converter.

V. Conclusion

In this work, we propose EVA, a versatile and efficient generative engine for discovering novel analog circuit topologies. EVA employs a bottom-up generation framework using a decoder-only transformer to sequentially predict device pin connections, starting from a VSS pin. By pretraining on an unlabeled dataset of circuit topologies, EVA establishes a strong foundation for topology generation, enabling finetuning with minimal performance-labeled data. Through finetuning strategies based on PPO or DPO, EVA achieves high discovery efficiency, targeting high-performance circuits while minimizing wasteful evaluations. EVA paves the way for AI-driven discovery in analog circuit, unlocking unprecedented potential for innovation in next-generation technologies.

REFERENCES

[1] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "Cktgnn: Circuit graph neural network for electronic design automation," *arXiv preprint arXiv:2308.16406*, 2023.

[2] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[3] W. Cao, M. Benosman, X. Zhang, and R. Ma, "Domain knowledge-infused deep learning for automated analog/radio-frequency circuit parameter optimization," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1015–1020.

[4] J. Gao, W. Cao, and X. Zhang, "Rose: Robust analog circuit parameter optimization with sampling-efficient reinforcement learning," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[5] W. Cao, J. Gao, T. Ma, R. Ma, M. Benosman, and X. Zhang, "Rose-opt: Robust and efficient analog circuit parameter optimization with knowledge-infused reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[6] W. Lyu, F. Yang, C. Yan, D. Zhou, and X. Zeng, "Batch bayesian optimization via multi-objective acquisition ensemble for automated analog circuit design," in *International conference on machine learning*. PMLR, 2018, pp. 3306–3314.

[7] B. Liu, Y. Wang, Z. Yu, L. Liu, M. Li, Z. Wang, J. Lu, and F. V. Fernández, "Analog circuit optimization system based on hybrid evolutionary algorithms," *Integration*, vol. 42, no. 2, pp. 137–148, 2009.

[8] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, and S. S. Sapatnekar, "Align: Open-source analog layout automation from the ground up," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[9] B. Xu, K. Zhu, M. Liu, Y. Lin, S. Li, X. Tang, N. Sun, and D. Z. Pan, "Magical: Toward fully automated analog ic layout leveraging human and machine intelligence," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–8.

[10] Z. Wu and I. Savidis, "Circuit-gnn: A graph neural network for transistor-level modeling of analog circuit hierarchies," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[11] Y. Lai, S. Lee, G. Chen, S. Poddar, M. Hu, D. Z. Pan, and P. Luo, "Analogcoder: Analog circuit design via training-free code generation," *arXiv preprint arXiv:2405.14918*, 2024.

[12] Z. Chen, J. Huang, Y. Liu, F. Yang, L. Shang, D. Zhou, and X. Zeng, "Artisan: Automated operational amplifier design via domain-specific large language model," in *2024 61th ACM/IEEE Design Automation Conference (DAC)*. ACM, 2024, pp. 1–6.

[13] C.-C. Chang, Y. Shan, S. Fan, J. Li, S. Zhang, N. Cao, Y. Chen, and X. Zhang, "Lamagic: Language-model-based topology generation for analog integrated circuits," *arXiv preprint arXiv:2407.18269*, 2024.

[14] E. A. Karahan, Z. Liu, and K. Sengupta, "Deep-learning-based inverse-designed millimeter-wave passives and power amplifiers," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 11, pp. 3074–3088, 2023.

[15] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[16] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[17] J. Gao, W. Cao, J. Yang, and X. Zhang, "Analoggenie: A generative engine for automatic discovery of analog circuit topologies," in *The Thirteenth International Conference on Learning Representations*, 2025. [Online]. Available: https://openreview.net/forum?id=jCPak79Kev

[18] A. Radford, "Improving language understanding by generative pre-training," 2018.

[19] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[20] N. Otsu *et al.*, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[21] S. Xu, W. Fu, J. Gao, W. Ye, W. Liu, Z. Mei, G. Wang, C. Yu, and Y. Wu, "Is dpo superior to ppo for llm alignment? a comprehensive study," *arXiv preprint arXiv:2404.10719*, 2024.

[22] R. A. Bradley and M. E. Terry, "Rank analysis of incomplete block designs: I. the method of paired comparisons," *Biometrika*, vol. 39, no. 3/4, pp. 324–345, 1952.

[23] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 1st ed. USA: McGraw-Hill, Inc., 2000.

[24] B. Razavi and R. Behzad, *RF microelectronics*. Prentice hall New York, 2012, vol. 2.

[25] D. A. Johns and K. Martin, *Analog integrated circuit design*. John Wiley & Sons, 2008.

[26] P. R. Gray, P. J. Hurst, S. H. Lewis, and R. G. Meyer, *Analysis and design of analog integrated circuits*. John Wiley & Sons, 2009.

[27] P. E. Allen and D. R. Holberg, *CMOS analog circuit design*. Elsevier, 2011.

[28] H. Camenzind, *Designing analog chips*. Virtualbookworm Publishing, 2005.

[29] X. Guo and L. Zhao, "A systematic survey on deep generative models for graph generation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 5, pp. 5370–5390, 2022.

Reinforcement Learning-Driven Window Selection for Enhanced Window-Based Rip-up and Reroute in Chip Detailed Routing

Yu-Chan Keng[1], Yu-Chun Pai[1], Wen-Hao Liu[2], Haoxing Ren[2], Danny Liu[2], Rongjian Liang[2],
Mark Ho[2], Anthony Agnesina[2], and Yih-Lang Li[1]
[1]Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu, Taiwan; [2]NVIDIA Corporation, USA
{kch772018, a0935796620}@gmail.com, {wenhliu, haoxingr, dannliu, rliang, chiatungh, aagnesina}@nvidia.com, ylli@cs.nctu.edu.tw

ABSTRACT

With increasingly complex design rules and pin density in advanced technology nodes, achieving a violation-free layout has become more challenging, also making rip-up and reroute (RUR) the most runtime-intensive component of detailed routing. We propose a novel reinforcement learning (RL)-based approach to enhance the window-based RUR process. Our method features a dynamic window generation strategy that adjusts window size and position based on the distribution of design rule violations (DRV), enabling efficient targeting of congested areas. By leveraging the predictive capabilities of RL, our approach aims to minimize DRVs and achieve high-quality routing results. Experimental results demonstrate that our method outperforms the state-of-the-art detailed routers, TritonRoute, achieving a DRV-free solution, averagely improving wirelength by 0.07%, via count by 2.42%, and consuming almost the same average runtime.

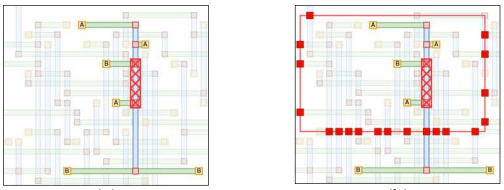
1. INTRODUCTION

In the physical design flow, the detailed routing process is divided into two main stages based on different objectives: initial routing and RUR. In the first stage, connectivity for each net is established by following the routing guides provided by global routing. This stage typically requires less time, and is less likely to achieve a DRV-free result. To address remaining DRVs, the RUR scheme becomes crucial as detailed routing requires considering increasingly complex design rules and pin density.

Previous routing research employs a straightforward yet effective rip-up scheme that rips up the complete or partial routing of nets in the subsequent rerouting operation. This approach is known as the path-based RUR scheme [2, 3]. While the path-based scheme can improve outcomes through the negotiation-based RUR strategy [4], it can incur huge runtime costs, especially when the routing space is congested.

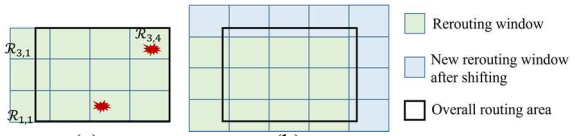
Another approach is to select a specific region or window containing DRVs and reroute partial or all of the wire segments within this window. This method is known as the window-based RUR scheme. A. B. Kahng et al. [1] introduced TritonRoute, an end-to-end detailed router that leverages the window-based approach to define routing regions. Fig. 1 presents a representative instance of DRV and window-based rerouting. The red rectangular region delineates the rerouting window, and the red pins denote the intersections between routed paths and the window's boundary. These intersection points and signal pins within the window serve as the connection targets for the rerouting operation. This method reduces routing problem size and can significantly reduce runtime of RUR process by rerouting non-overlapping windows in parallel using multithreading.

TritonRoute's window generation uses identical patterns, and



(a) (b)

Fig. 1 (a) Example of DRV; (b) Example of window generation.



(a) (b)

Fig. 2 (a) Partition an area into 3×4 rerouting windows; (b) Partition an area into 4×5 rerouting windows.

creates diverse rerouting regions by monotonic window shifting. Fig. 2(a) depicts a routing area partitioned into a 3×4 grid of rerouting windows, each encompassing 7×7 global cells (GCells). The rerouting procedure entails a systematic diagonal shift of each window towards the bottom-left corner until it exits the routing area. Upon the window shifting, a new window is formed to cover the newly exposed region, as illustrated in Fig. 2(b).

The monotonic window generation and shifting scheme depicts efficacy in the removal of DRVs. However, Fig. 2(a) shows the potential insufficiency in resolving DRVs. The DRVs situated within regions $\mathcal{R}_{3,4}$ and $\mathcal{R}_{1,3}$ cannot be rectified by the current partitioning scheme if they demand rerouting towards regions with locally lowest metal density, such as $\mathcal{R}_{2,4}$ and $\mathcal{R}_{1,2}$. If the lower metal-density areas within $\mathcal{R}_{3,4}$ and $\mathcal{R}_{1,3}$ are located at the top and right areas, respectively, rerouting paths will be forced towards these areas, making DRV removal more challenging.

Although AI techniques have been successfully applied to various physical design problems—such as placement optimization, routing guide selection for detailed analog design routing, DRV removal for cell layouts, and obstacle-avoided Steiner tree generation [5-9]—there has yet to be a solution capable of addressing the general detailed routing problem due to its high complexity and difficulty. Our objective is to design an RL agent to optimize the selection of rerouting windows, with the aim of enhancing the quality of the detailed routing process.

The main contribution of this paper is the proposed RL-based dynamic window generation strategy, allowing flexible window sizing and positioning based on the distribution patterns of DRVs. We leverage RL's extensive learning experiences and predictive capabilities to evolving layout conditions, thereby expediting DRV resolution and minimizing the iterative process required to achieve a DRV-free layout. Experimental results show that,

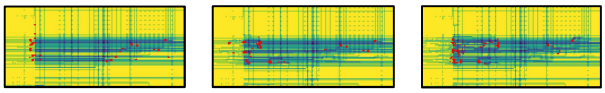


Fig. 3 The phenomenon of violation propagation (from left to right). The red markers represent the violations.

compared to TritonRoute, our approach resolves all DRCs, and improves wire length (WL) by 0.07% and via count by 2.42% with almost the same runtime.

The remainder of this paper is organized as follows: Section 2 introduces problem formulation and important previous works. Section 3 describes the design concepts. Section 4 outlines the RL framework design. Section 5 presents the experimental results, and Section 6 concludes the paper.

2. PRELIMINARY

2.1 Problem Formulation

The input to the problem includes a placed netlist, routing guides generated from global routing, and a set of design rule constraints. The primary goal is to successfully route all connections, while optimizing several key metrics such as DRVs, total wirelength, via count, and runtime.

2.2 Window-Based RUR Scheme

A. B. Kahng et al. [1] proposed TritonRoute, an open-source end-to-end detailed router that employs a window-based RUR scheme. In each iteration, the router segments the layout into discrete, non-intersecting windows, each spanning 7×7 GCells. These window routings are subsequently conducted individually.

For each rerouting window, TritonRoute constructs a local grid graph. Maze routing is then applied to reroute selected subnets based on the window's Rip-up Mode (RM). In RM 0, only subnets with DRVs are ripped up and rerouted to reduce runtime and focus on DRV elimination; in RM 1, all subnets are ripped up to rebuild their topology, aiming for the opportunity to achieve a lower DRV count. RM 1 is utilized in preceding iterations of the TritonRoute to reduce wirelength, via count, and mitigate DRVs.

2.3 Reinforcement Learning

RL [13] is a machine learning paradigm where an agent learns to make decisions through interactions with an environment. The agent receives state observations from the environment, takes actions based on its policy, and receives rewards that indicate the quality of these actions.

2.3.1 MaskablePPO

We employ Proximal Policy Optimization (PPO) [10], a SOTA RL algorithm known for its stability and reliability in continuous control tasks. PPO uses a clipped objective function to prevent excessive policy updates, making it particularly suitable for our window generation problem requiring consistent and stable learning. The algorithm updates its policy by optimizing:

$$L(\theta) = \mathbb{E}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)], \quad (1)$$

where $r_t(\theta)$ is the probability ratio between the new and old policies, A_t is the gain estimate, and ϵ is the clipping parameter.

To handle invalid window configurations in our specific problem domain, we adopt MaskablePPO [11], an extension of PPO that incorporates action masking. This modification allows us to explicitly exclude invalid actions, such as windows that would exceed layout boundaries or violate minimum/maximum size constraints during the policy sampling process. The action masking mechanism ensures that our agent only considers feasible window configurations.

2.3.2 RL Network Architecture

The network architecture used in our work is a general RL framework provided by Stable Baselines3 [12]. The framework includes three essential components: the feature extractor, the policy network, and the value network.

2.3.3 SE-ResNet

ResNet [14] addresses the gradient vanishing problem through shortcut connections, allowing for deeper neural network architectures. SE-ResNet enhances ResNet by integrating Squeeze-and-Excitation Networks (SE-Net) [15] into the original residual blocks. The "Squeeze" operation applies a global average pooling layer to condense spatial information into the channel dimension, capturing global insights across the feature map for each channel. Following this, the "Excitation" phase uses a fully connected network and a sigmoid function to assign importance to each channel, thus weighing them based on relevance.

A key advantage of SE-ResNet over standard ResNet is its adaptive channel weighting, which focuses on enhancing essential features while suppressing irrelevant ones. This selective emphasis on channel information, rather than purely spatial information, results in higher accuracy across a range of tasks, with only a modest increase in computation and parameters.

3. DESIGN CONCEPTS

3.1 DRV Propagation

The ideal RUR operation seeks to entirely eliminate all DRVs within a specified rerouting window. Nevertheless, in regions characterized by high congestion, the removal of existing DRVs can inadvertently result in the emergence of new DRVs within other areas that become congested after rerouting, termed *DRV propagation*. DRV propagation is a necessary process before all DRVs are eliminated. Fig. 3 shows the DRV propagation phenomenon. The figures, arranged from left to right, depict the routing iterations progressing from the early to the late stages.

Consider three regions: $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$, arranged sequentially from left to right. $\mathcal{R}_1$ is characterized by the presence of multiple DRVs, while $\mathcal{R}_2$ and $\mathcal{R}_3$ are characterized by high and low congestion, respectively. Rerouting paths through $\mathcal{R}_2$ is likely to induce the emergence of new DRVs. If $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$ are encompassed within a single rerouting window, the complete elimination of all DRVs is feasible. However, if only $\mathcal{R}_1$ and $\mathcal{R}_2$ are included within the same rerouting window, and $\mathcal{R}_3$ is excluded, rerouted paths are more likely towards low-congested regions, rather than $\mathcal{R}_2$, rendering DRV removal more intricate.

The dimension of the rerouting window constitutes another critical factor. A larger window possesses the potential to mitigate the aforementioned DRV propagation issue and eliminate all DRVs. Nevertheless, unnecessarily large windows can substantially increase computational runtime.

3.2 Layout-to-Tile Feature Transformation

3.2.1 Tile-Based Rerouting Window Generation

We apply a tile-based data structure, wherein each tile depicts the smallest unit within the layout and stores local information,

including the number of DRVs and metal density. Rerouting windows are constructed by merging adjacent tiles, leveraging the tile information to guide the window generation process.

3.2.1.1 Tiling the layout

Initially, the layout is partitioned into tiles, with each tile possessing a width equivalent to half the width of a GCell. This finer-grained partitioning, in contrast to TritonRoute's window shifting distance of one GCell, affords a greater number of potential positions for window generation, thereby facilitating the creation of more diverse routing regions.

Each tile encompasses multiple layers, including metal and cut layers. For each layer within a tile, the number of DRVs and metal density are calculated. This results in a list of metrics that collectively represent the layout information for each layer within the tile. A tile is classified as violated if it contains at least one DRV on any of its constituent layers.

3.2.1.2 Building the rerouting window

Window generation starts by selecting a violated tile with the regions around it as the kernel window, subsequently extending the boundary of the current window in four directions: left, bottom, right, and top. Each extension expands the entire boundary of the current window, preserving its rectangular shape. The dimension of the rerouting window is determined by the agent based on a multitude of factors, including the number and severity of DRVs within the window, the metal density of the region, the current iteration number, the overall quality of the routing solution, and the computation time.

3.2.1.3 Window Objectives

The generated windows can be broadly classified into two objectives: propagation and elimination.

***Propagation*:** In high-density regions, the removal of DRVs within a single window may prove challenging. A more effective approach involves enlarging the window to encompass regions of lower density or employing RM 1 to reconstruct the routing topology within the region. The reconstruction of the routing topology can also enhance the efficiency of DRV propagation, as the newly constructed topology will explore a more optimal solution to address the current DRVs.

***Elimination*:** For windows possessing ample routing resources, the utilization of RM 0 to directly rip up and reroute violated nets is feasible, leading to the convergence of DRVs and a reduction in computational runtime.

A strong correlation exists between early and later iterations within the rerouting process, especially when addressing high-density regions. In early iterations, rerouting is applied to divide and disperse DRVs towards low-density regions, thereby creating more opportunities for their elimination in subsequent iterations.

Theoretically, adherence to the aforementioned policies can result in enhanced rerouting outcomes. However, the RL agent must acquire the ability to discern when to adhere to these policies and when to deviate from them, as there may exist exceptions wherein alternative strategies yield superior outcomes.

3.2.2 Calculating Tile Density

Two key pieces of information are considered: DRV density and metal density.

3.2.2.1 DRV density

Rather than employing a simple DRV count, we calculate the DRV density within each tile. This normalization is crucial for providing meaningful input to the agent, as raw DRV counts can exhibit significant variation across different datasets. Besides, a DRV characterized by a larger DRV area typically implies a more challenging eliminate process. Furthermore, by eschewing the differentiation between various types of DRVs, we simplify the problem formulation. Thus, the DRV area and density constitute more suitable metrics in comparison to raw DRV counts. The DRV density can be formulated as:

$$den_{vio} = \sum_{i \in S_{DRV}} area_{vio}(i) \,/\, Tile\ Area, \tag{2}$$

where S_{DRV} is the set of DRVs and $area_{vio}(i)$ is the DRV area of the ith DRV in S_{DRV}. This calculation is suitable for both metal layers and cut layers.

3.2.2.2 Metal density

Metal density encompasses net wire segments, pins, and obstacles within the metal layers, as well as vias within the cut layers. TritonRoute adopted a grid-based routing model. To calculate the metal density of a specific layer, the metal ratio of each track on that layer is computed, and these metal ratios are subsequently summed. The metal ratio of a track represents the proportion of the track occupied by metal and can be calculated as follows:

$$rat_M(t) = \sum_{mo \in \mathcal{MO}(t)} ml(mo)/tlen, \tag{3}$$

where $rat_M(t)$ is the metal ratio of track t, $\mathcal{MO}(t)$ is the set of metal wire segments at track t, ml is the metal length of a metal wire segment, and $tlen$ is the track length. The metal density of a tile can then be formulated as:

$$den_M = \sum_{t \in \mathcal{T}} rat_M(t)\,, \tag{4}$$

where $\mathcal{T}$ is the track set in a tile.

To further refine our analysis, we subdivide metal wire segments into two categories: *fixed* and *non-fixed*. Fixed segments represent obstacle density and encompass objects such as pins and obstacles that remain unchanged during the RUR process. Non-fixed segments represent wire density and consist of routed wire segments that are subject to modification subsequent to the rerouting process.

In the cut layer, the available routing space can be determined by the total number of valid via candidates. A via candidate is a track intersection point between adjacent metal layers. The via density can be quantified as follows:

$$den_{via} = \#Vias \,/\, \#VVC, \tag{5}$$

where $\#VVC$ is the total number of valid via candidates.

3.3 Overview of the Detailed Routing Flow

In the preprocessing stage, we perform TritonRoute's initial procedures, including the parsing of input data, the preparation of routing data, the generation of pin access points, and the track assignment. After that, we construct the tile data structure and initialize the tile information. We then apply TritonRoute to obtain the first-iteration routing result that guarantees net connectivity (i.e., no open nets). Subsequent to the initial routing, we identify violated tiles and sort them in ascending order of density, from low-density regions to high-density regions. This density-based sorting strategy is designed to strategically release routing space around congested areas during the early iterations.

In each routing iteration, violated tiles are sequentially selected and utilized as kernel windows. For each kernel window, our agent generates a list of non-overlapping windows and selects a rip-up mode for each window. TritonRoute then completes the rerouting of each window. Any rerouting attempts that fail to meet TritonRoute's constraints are discarded rather than committed to the layout.

Subsequent to the completion of window processing in each iteration, the tile information is updated, which entails recalculating DRV densities and non-fixed metal densities. The termination conditions are then evaluated. The detailed routing stage concludes when either all DRVs are resolved or the total number of iterations attains the predetermined upper limit.

4. RL-BASED METHODOLOGY

4.1 RL Methodology Setting

4.1.1 Environmental State

We employ a CNN-based network architecture to process the layout information derived from our tile-based representation. Rather than processing the entire layout, we focus on a 64×64 tile region centered around the target violated tile as our environmental state. This localized approach is justified by two key factors: Firstly, DRV propagation typically affects only nearby regions, rendering local information sufficient for effective window generation decisions. Secondly, this fixed-size state representation enables our method to handle layouts of varying dimensions uniformly.

To provide a comprehensive representation of the layout, we adopt a 3D environmental state. This 3D state is constructed by considering the layout in three dimensions: width, height, and layer. Each layer, such as metal or cut, has its own density map.

Key components of the environmental state contain metal layer density map (3D), cut layer density map (3D), and auxiliary information map (2D). In metal layer density map, DRV density indicates the concentration of DRVs. Obstacle density represents the density of fixed objects like pins and obstacles. Wire density reflects the density of routed wires. In cut layer density map, DRV density indicates the concentration of DRVs. Obstacle density represents the density of fixed objects like vias in the power network. Via density reflects the density of routed vias. In auxiliary information map, rerouted count map tracks the number of rerouting attempts in a region. RM 0 failure count map records the number of failed RM 0 attempts in a region. RM 1 failure count map records the number of failed RM 1 attempts in a region. Window position map indicates the current position of the rerouting window on the layout.

By incorporating these components, the RL agent can make informed decisions about window generation and RUR strategies, leading to improved routing quality.

4.1.2 Action

To generate a rerouting window, the agent must sequentially decide on window boundary extensions (up, down, left, right) and select a rip-up mode (RM 0, RM 1). This action space enables the agent to configure the window and its associated routing strategy.

4.1.3 Reward

To optimize these actions effectively, we have designed separate reward functions for each of the two types of actions. For the window generation step, the reward function encourages optimal window configuration. Once the agent selects a rip-up mode, the window immediately proceeds with detailed routing, linking the detailed routing step directly to the rip-up mode selection. This structure allows each action type to be guided by its respective reward, promoting effective decision-making in both window sizing and routing strategy.

4.1.3.1 Window generation reward

A DRV is more likely to be eliminated if it is not located near the rerouting window's boundary. This is because in such cases, the DRV has four potential directions to allocate rerouting paths, increasing the chances of successful eliminate. To address this issue, we impose a constraint on the window's aspect ratio and designed a window size reward rw_{ws} for extension actions. We define window size reward as:

$$rw_{ws} = \begin{cases} 0.05, if\ 0.6 \le ar \le 1.7\ and\ (wid_W \le 15\ or\ hgt_W \le 15) \\ 0.02, if\ 0.6 \le ar \le 1.7\ and\ (wid_W > 15\ or\ hgt_W > 15), \\ 0, \qquad otherwise \end{cases} \quad (6)$$

where ar is the aspect ratio of the window, wid_W is the width of the window, and hgt_W is the height of the window. This reward encourages the agent to extend the window toward a nearly square shape, aiming to position DRVs closer to the center of the window. To prevent the agent from excessively favoring maximum window extensions, the reward diminishes when the window size exceeds 15. However, the design does not strictly enforce a square window to allow some flexibility in handling specific scenarios that may require different patterns.

4.1.3.2 Detailed routing reward

In this work, the main objective is to eliminate DRVs as much as possible. Therefore, the reward design considers the remaining DRVs after a window rerouting has been completed. We define the DRV reward as:

$$rw_{vio} = (vio_{ini} - vio_{rem})/\ vio_{rem}, \quad (7)$$

where vio_{ini} is the initial number of DRVs in the window before rerouting and vio_{rem} is the remaining number of DRVs in the window after rerouting.

Considering the two different rip-up modes, RM 0 takes less runtime than RM 1 due to the number of rerouted nets. To suppress the agent's preference for selecting RM 1, we first define the rerouted nets proportion (*rnp*):

$$rnp = \#Net_{\ reroute}/\ \#Net_{total}, \quad (8)$$

where $\#Net_{\ reroute}$ is the number of reroute nets in the window, and $\#Net_{total}$ is the total number of nets in the window. While RM 1 always reroutes all nets (rnp = 1), RM 0 reroutes a subset of nets. To account for potential runtime differences, especially in windows with fewer nets, we define the penalty for the rip-up mode as follows:

$$penalty_{RM} = rnp * (wd_{window}/\ \max_{\forall tile\ i}(wd_{tile}(i))), \quad (9)$$

where $wd_{tile}(i)$ is the wire density value of tile *i*. The ratio of the window's density to the maximum tile density serves as an indicator of the window's congestion level. In regions characterized by high congestion, RM 1, which involves the sequential rerouting of all nets, incurs a higher penalty due to increased computational complexity. Finally, the total reward of detailed routing is defined as:

$$rw_{dr} = \begin{cases} \alpha \cdot rw_{vio} + \beta \cdot penalty_{RM}, & if\ dr\ success \\ 0, & otherwise \end{cases}. \quad (10)$$

4.2 Flow Optimization

The SE-ResNet architecture integrates Squeeze-and-Excitation (SE) blocks, enabling the agent to assign adaptive weights across feature channels and make stage-aware decisions. In our method, the 3D state maps capture critical layout information, including DRV and metal density, which are intrinsically interrelated, while the 2D auxiliary state map provides essential routing iteration information. The SE blocks facilitate the integration of these diverse channels, allowing the agent to better capture cross-channel dependencies. This capability enhances the agent's ability to derive globally optimized solutions, effectively balancing DRV resolution and routing quality.

In early stages, the SE-ResNet agent tends to select larger windows and RM 1. This strategy efficiently resolves dense DRVs and mitigates congestion in critical area. As the routing process progresses into the mid-to-late stages, the SE-ResNet agent shifts its focus to smaller window sizes and RM 0, thus optimizing runtime and enhancing routing efficiency.

This dynamic adaptation demonstrates SE-ResNet's ability to balance short-term performance with long-term routing quality, providing a holistic approach to resolving DRVs while improving overall routing outcomes.

4.3 Training Workflow

Each layout is transformed into a training data instance by:

a. *Initial Routing*: Perform the preprocessing stage and the initial iteration of TritonRoute to find an initial routing solution.
b. *DRV Prioritization*: Identify and prioritize violated tiles based on the density-based sorting strategy outlined in Section 3.3.

Each training episode includes:

a. *Target Tile Selection*: Select a violated tile from the prioritized list as the target tile for the current episode.
b. *Environmental State Generation*: Provide the RL agent with an environmental state encompassing the layout information centered around the target tile.
c. *Window Generation*: Guide the agent to generate a window around the target tile through multiple decision-making steps.
d. *Detailed Routing*: Execute the detailed routing process within the generated window.
e. *Reward Calculation*: Compute a reward based on the quality of the routing solution within the window.
f. Episode Termination: Conclude the episode and transition to the next target tile or a new layout.

A new layout is introduced as training data once all DRVs in the current layout have been resolved or the total number of iterations attains the predetermined upper limit. By iteratively training the agent on a sequence of layouts, the agent learns to effectively generate windows and select appropriate RUR strategies to improve routing quality.

4.4 Network Architecture

Fig. 4 depicts the network architecture used in this work. The feature extractor uses a CNN-based network, SE-ResNet34, to process three individual environmental states. Due to the nature differences between metal layers and cut layers, we do not combine metal layer density and cut layer density as a single environmental state. Instead, we use different networks to process them. The 3D metal layer state (wire features) and 3D cut layer state (via features) are processed through two SE-ResNet networks with 3D convolutional layers. A mean function is then

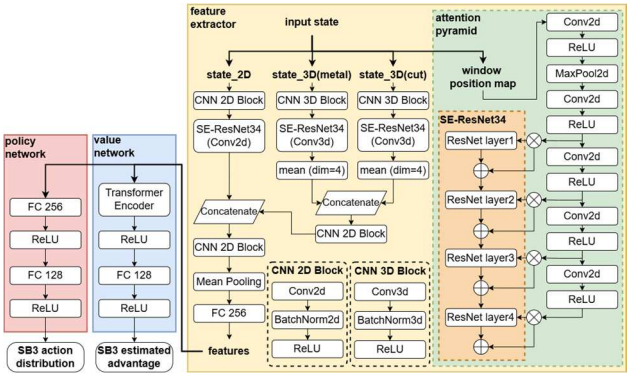


Fig. 4. The network architecture of the RL agent.

used to transform these two features into 2D representations, which are subsequently fused and passed through a convolutional layer to obtain integrated 3D features with 2D dimensions. On the other hand, the 2D states are processed using a SE-ResNet with 2D convolutional layers. The 2D features are concatenated with the integrated 3D features and passed through a final convolutional layer, followed by global pooling and a fully connected layer, to obtain the final feature representation.

A unique aspect of the feature extractor is the window position network. The window position map (one channel of the 2D state) is passed into an independent network consisting of four convolutional layers. The output of each layer becomes an attention feature with window position information. This attention feature is integrated into each SE-ResNet layer through element-wise multiplication and addition with the corresponding SE-ResNet output feature. The reason for adding window position information is that while the agent needs information about the surrounding region during window boundary extension, it only needs to consider regions inside the generated window when determining the rip-up mode.

In the policy and value network, we employ fully connected layers to produce the action distribution used to select an action. In the value network, a fully connected layer is alternated with a transformer encoder, enhancing the network's capacity to model complex value representations and strengthen the advantage estimation. After the outputs of the policy and value networks are generated, the Stable Baselines3 internal functions use the action distribution to select an action and use a final fully connected layer in the value network to estimate the advantages.

4.5 Detailed Routing Parallelization

In the training stage, we have to process the detailed routing of each window sequentially because of the reward calculation and returning in one episode. To achieve parallelization, we can sequentially generate multiple windows and reroute them concurrently. This approach works because we need to provide the next environmental state at each step during training, which is not necessary in the testing stage.

Another bottleneck for runtime is the agent inference time. For each step, we must pass through the entire network once, and the accumulated inference time can become considerable as the layout size increases. To address this, we utilize a batch prediction method. First, we generate multiple kernel windows on the layout. We then place each window state into a batch and pass them through the model simultaneously, which leverages GPU parallelization and generates all windows concurrently.

TABLE 1. COMPARISON BETWEEN OURS AND TRITONROUTE

	DRV		Runtime	Wirelength	Via
	Ours	TR	(Ours-TR)/TR	(Ours-TR)/TR	(Ours-TR)/TR
18_1	0	0	**-11.63%**	**-0.09%**	**-0.23%**
18_2	0	0	**-27.51%**	**-0.06%**	**-0.89%**
18_3	0	0	**-2.90%**	**-0.03%**	**-0.69%**
18_4	**0**	9	**-36.16%**	**-0.09%**	**-0.49%**
18_5	0	0	7.05%	**-0.13%**	**-3.68%**
18_6	0	0	3.26%	**-0.07%**	**-3.82%**
18_7	0	0	**-3.31%**	**-0.09%**	**-3.41%**
18_8	0	0	4.14%	**-0.03%**	**-3.42%**
18_9	0	0	9.38%	**-0.04%**	**-3.62%**
18_10	**0**	8	63.80%	**-0.08%**	**-3.90%**
Avg.			0.61%	**-0.07%**	**-2.42%**

5. EXPERIMENTAL RESULTS

The proposed window selection framework was implemented by modifying TritonRoute's open-source code [1] in C++, while the RL agent was developed using the Stable Baselines3 [12] RL framework in Python. All experiments were conducted on a Linux platform with a 2.1GHz Intel Xeon Gold 5218R CPU, 256 GB RAM, and an NVIDIA GeForce RTX 3090 GPU.

5.1 Training and Testing Datasets

Our experiments used the ISPD-2018 benchmark suite [16], which includes 10 test cases. To adapt the ISPD benchmarks for our testing dataset, we employed AutoDMP [17] to perform cell replacement and CUGR [18] to perform global routing on the benchmarks. The training dataset consists of 15 modified layouts derived from four ISPD benchmarks (ispd18_test2, test3, test4, and test5). Each replaced layout retains the same standard cell count and fixed macro positions, creating diverse placements that serve as identical layouts for training purposes. We preprocessed all training data using TritonRoute. The results show that 7 training data points achieved DRV-free routing with TritonRoute, while 8 data points exhibited DRVs. The testing dataset contains all 10 ISPD-2018 benchmark test cases. These benchmarks were processed similarly to the training dataset but were reserved exclusively for evaluating the framework's performance and generalizability on unseen layouts.

5.2 Detailed Routing Results

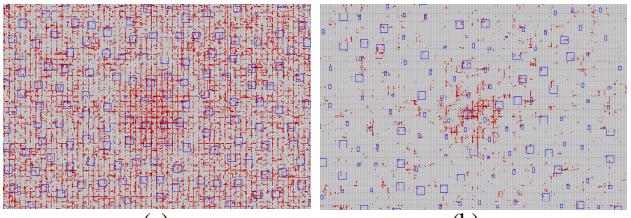
Table 1 presents a comparison between our method and TritonRoute for detailed routing on the ISPD-2018 benchmarks. Metrics such as DRV, wirelength, and via count were calculated using TritonRoute. Runtime was measured starting from the end of TritonRoute's initial iteration (iteration 0).

In the ISPD-2018 benchmarks, certain test cases (notably 18_3, 18_4, and 18_10) are considered challenging. TritonRoute is unable to achieve DRV-free results on 18_4 and 18_10. As we increase the rerouting iteration number of TritonRoute, the remaining DRV numbers of them remain unchanged. In contrast, our method successfully achieves DRV-free routing on all ISPD-2018 test cases, demonstrating that the proposed RL agent has more flexibility to handle complex DRVs in these benchmarks.

For runtime, our method outperformed TritonRoute in five test cases but was slower in others, with an average runtime increase of just 0.61%, effectively resulting in a near tie. Our runtime performance is measured as the total runtime of the routing flow, including the detailed routing time, the environment state update time, and the agent inference time required for generating actions.

TABLE 2. COMPARISON BETWEEN SE-RESNET AND RESNET

	Early Stage					Middle Stage			
	SE-ResNet		ResNet			SE-ResNet		ResNet	
Iter.	RM_0	RM_1	RM_0	RM_1	Iter.	RM_0	RM_1	RM_0	RM_1
3	8	389	378	20	12	157	188	341	37
4	34	362	387	19	13	175	161	344	33
5	28	367	361	40	14	163	156	343	29
6	64	315	366	34	15	199	122	346	36
7	67	307	335	31	16	208	100	336	40



(a) (b)

Fig. 5. Window selected by SE-ResNet agent and DRV tiles in the (a) early stage and (b) middle stage of ISPD 18_10.

Regarding quality metrics, our approach bettered both wirelength and via count by an average improvement of 0.07% and 2.42%, respectively. These results show that our method can produce cleaner routing results with better wirelength and via count at a little cost of runtime.

5.3 A Comparative Analysis of SE-ResNet

Table 2 highlights the distinct rip-up mode selection patterns for the 18_10 testcase across different iteration stages. ResNet-based agent shows a consistent preference, favoring a single rip-up mode throughout the iterations. In contrast, SE-ResNet-based agent shows a more dynamic approach, initially selecting RM 1 more frequently in the early stages, before gradually shifting to favor RM 0 as it progresses through the mid-to-late stages.

For the window size selection, ResNet-based agent tends to select smaller window sizes, while the SE-ResNet-based agent shows a more balanced distribution in window size choices. In Fig. 5, red and blue areas represent DRV tiles, and the generated windows, respectively. As iterations progress, the SE-ResNet-based agent increasingly selects smaller window sizes.

Based on the experimental results, it is evident that SE-ResNet leverages the SE-block to assign adaptive weights across channels, enabling the agent to learn that optimal window sizes and rip-up modes vary at different stages. This adaptation allows the agent to resolve DRVs while simultaneously improving routing quality. In the early stages, SE-ResNet's preference for larger windows and the selection of RM 1, which involves ripping up and rerouting all nets within the window, contributes to an increase in runtime. However, in the mid-to-late stages, the agent shifts to smaller windows and more frequent use of RM 0, effectively controlling runtime within manageable limits.

6. CONCLUSIONS

We proposed a novel framework for adaptive window generation and RM selection for detailed routing optimization. Our method, leveraging a SE-ResNet-based agent, dynamically generates and refines routing windows by merging tiles based on routing information. Experimental results on the ISPD-2018 benchmarks demonstrate that our method achieves superior routing quality and efficiency, and more effective capability of removing DRVs.

7. REFERENCES

[1] A. B. Kahng, et al., “TritonRoute: The Open-Source Detailed Router,” *IEEE TCAD*, vol. 40, no. 3, pp. 547-559, 2021.

[2] G. Chen, et al., “Dr. cu: Detailed routing by sparse grid graph and minimum-area-captured path search,” *IEEE TCAD*, vol. 39, no. 9, pp. 1902-1915, 2020.

[3] H. Li, et al., “Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction,” in *IEEE/ACM ICCAD*, pp. 1–7, 2019.

[4] W.-H. Liu, et al., “NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing,” *IEEE TCAD*, vol. 32, no. 5, pp. 709–722, 2013.

[5] A. Mirhoseini, et al., “Chip placement with deep reinforcement learning,” arXiv:2004.10746 [cs.LG], 2020.

[6] H. Chen, et al., “Reinforcement learning guided detailed routing for custom circuits,” in *ACM/IEEE ISPD*, pp. 26–34, 2023.

[7] P.-Y. Chen, et al., “A reinforcement learning agent for obstacle-avoiding rectilinear steiner tree construction,” in *ACM/IEEE ISPD*, pp. 107–115, 2022.

[8] J. Liu, et al., “Rest: Constructing rectilinear steiner minimum tree via reinforcement learning,” in *ACM/IEEE DAC*, pp. 1135–1140, 2021.

[9] H. Ren, et al., “Invited- nvcell: Standard cell layout in advanced technology nodes with reinforcement learning,” in *ACM/IEEE DAC*, pp. 1291–1294, 2021.

[10] J. Schulman, et al., “Proximal policy optimization algorithms,” arXiv:1707.06347 [cs.LG], 2017.

[11] S. Huang, et al., “A Closer Look at Invalid Action Masking in Policy Gradient Algorithms,” arXiv:2006.14171 [cs.LG], 2022.

[12] A. Raffin, et al., “Stable-Baselines3: Reliable Reinforcement Learning Implementations,” *Journal of Machine Learning Research*, 2021.

[13] R. Sutton, et al., “Reinforcement Learning: An Introduction,” MIT Press, 1998.

[14] K. He, et al., “Deep Residual Learning for Image Recognition,” arXiv:1512.03385 [cs.LG], 2015.

[15] J. Hu, et al., “Squeeze-and-Excitation Networks,” arXiv:1709.01507 [cs.LG], 2019.

[16] S. Mantik, et al., “ISPD 2018 initial detailed routing contest and benchmarks,” in *ISPD*, pp. 140–143, 2018.

[17] A. Anthony, et al., “AutoDMP: Automated DREAMPlace-based Macro Placement,” in *ISPD*, pp. 149–157, 2023.

[18] J. Liu, et al., "CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model," in *ACM/IEEE DAC*, pp. 1-6, 2020.

Real-Time Dynamic IR-drop Prediction for IR ECO

Yu-Che Lee[1], Yu-Hsuan Chen[1], Yu-Chen Cheng[1], Yong-Fong Chang[2], Jia-Wei Lin[2], Hsun-Wei Pao[2], Yung-Chih Chen[3], Yi-Ting Li[1], Wuqian Tang[1], Shih-Chieh Chang[1], and Chun-Yao Wang[1]

[1]National Tsing Hua University, Taiwan
[2]Mediatek Inc., Taiwan
[3]National Taiwan University of Science and Technology, Taiwan

***Abstract*—During the IR Engineering Change Order (ECO) stage, cell moving leads to uncertain IR-drop results, requiring designers to explore multiple ECO candidates in each iteration to find a solution that effectively mitigates IR-drop, resulting in a long evaluation time. Although machine learning (ML)-based predictors have been proposed to expedite IR-drop evaluation, partial simulations are still needed to update features after ECO, taking over an hour and delaying IR-drop results. In this work, we propose a real-time dynamic IR-drop estimation method based on an XGBoost model with a global view of a cell's surroundings. After ECO, our method provides dynamic IR-drop results in minutes without running any simulations and thus achieves real-time estimation. This allows designers to evaluate multiple ECO candidates concurrently in a single iteration. We conducted the experiments on five ECO candidates of an industrial design with $3nm$ technology. The results show that the proposed model can effectively predict the IR-drop variations of moved cells after ECO with over 93% of fixed cells detected and an average MAE of $8.75mV$ achieved. Furthermore, our method achieves an $88X$ speedup over Voltus (commercial tool) and a $64X$ speedup over traditional ML predictors when evaluating a single ECO candidate. The speedup is expected to increase as the number of ECO candidates increases.**

I. Introduction

According to Ohm's law, when current flows through the power delivery network (PDN) and reaches the standard cells, the supply voltage drops due to the resistance of power rails. The voltage consumed in this process is known as IR-drop. IR-drop can increase the cell delay and potentially lead to circuit performance degradation and functional errors. As a result, IR-drop analysis for identifying violations and IR Engineering Change Order (ECO) for fixing the violations are extremely crucial in the sign-off flow.

The IR sign-off flow mentioned in previous works [7], [12], [10] indicates that an ECO and an IR-drop analysis are performed iteratively to identify and fix violations. However, the flow is oversimplified for the industry. In real-world scenarios, designers conduct cell moving-based ECO by adjusting the physical positions of violation cells in the layout to mitigate IR-drop. Due to the uncertain IR-drop variations caused by cell moving, designers typically experience several trials and errors within an iteration to reach a desired result. Consequently, multiple ECO candidates may exist in a single iteration. When evaluating each ECO candidate, even minor cell movements require running computationally expensive simulations for the entire circuit to obtain the IR-drop report. It is therefore time-consuming for designers to evaluate all candidates and select the best one for the next iteration. Additionally, due to design cycle time constraints and computational resource limitations during sign-off, the number of ECO candidates that can be attempted per iteration is limited.

This work is supported in part by the National Science and Technology Council (Taiwan) under NSTC 112-2218-E-007-014, NSTC 112-2221-E-007-106-MY2, NSTC 112-2221-E-007-108, NSTC 113-2218-E-007-020, NSTC 113-2221-E-007-082-MY3, NSTC 113-2425-H-007-004, NSTC 113-2640-E-011-003, NSTC 114-2425-H-007-002, and National Tsing Hua University under NTHU 112A0316 and NTHU 113A0314.

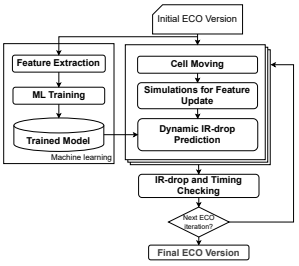


Fig. 1: Industrial flow + IR-drop prediction

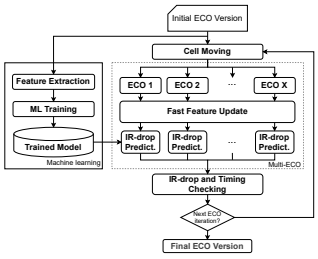


Fig. 2: Industrial flow + Real-time IR-drop prediction

Several machine learning (ML)-based IR-drop predictors have been proposed to accelerate IR-drop estimation [7], [12], [9], [3], [10], [8], [13], [15]. The ML-aided IR ECO flow is depicted in Fig. 1, where the IR-drop analysis process is replaced by IR-drop prediction. However, this flow still requires power analysis and partial rail analysis for resistance extraction to update the features of the modified circuit for prediction. This prevents designers from quickly acquiring IR-drop result caused by cell moving and becomes a primary bottleneck in parallelizing the evaluations of multiple ECO candidates due to the time- and resource-intensive nature of simulations.

In this paper, we propose a new scheme for real-time prediction of dynamic IR-drop after cell moving-based ECO, aiming to enhance the efficiency of the industrial IR sign-off flow. Firstly, we develop an XGBoost model [4] to predict the dynamic IR-drops of cells after ECO. Unlike previous works [7], [3], [9], [12] that let the model learn neighboring features in a local manner, we globally consider neighboring features in advance and provide representative ones for the model. Then, to achieve real-time IR-drop prediction, instead of rerunning simulations to update the features of the revised circuit, we use the physical information and raw features of cells before ECO to quickly estimate the changes in neighboring features. With real-time dynamic IR-drop prediction, we further propose an innovative industrial IR sign-off flow as shown in Fig. 2. In this flow, we can rapidly estimate the dynamic IR drop result after ECO. Since no simulations are required for updating features, multiple ECO candidates can be evaluated in parallel,

significantly reducing the time for designers to explore ECO candidates. Moreover, with the realization of parallelization, designers can explore more ECO candidates within an iteration and increase the probability of selecting a better ECO version for the next iteration.

The contributions of this paper are summarized as follows:

- An XGBoost-based dynamic IR-drop predictor. It utilizes newly proposed features designed to represent a cell's surrounding information globally and successfully improves the accuracy by 20.2% compared to the local approach.
- A feature update technique. It rapidly updates features after ECO without the need for resimulations, enabling real-time IR-drop estimation. It makes the predictor effectively capture the IR-drop variations of moved cells with more than 96% of fixed cells detected and an average MAE of $8.75mV$ achieved.
- An innovative ML-aided IR sign-off flow. It allows designers to evaluate multiple ECO candidates concurrently, enabling the exploration of more candidates within an iteration. It achieves $88X$ and $64X$ speedups over Voltus and traditional ML predictors for an ECO candidate.

The rest of this paper is organized as follows: Section II reviews the related works. Section III presents the proposed sign-off flow and details each step of it. Section IV shows the experimental results. Finally, Section V concludes our work.

II. Prior Works

Being aware of the time-consuming nature of IR-drop analysis, several ML-based methods for IR-drop prediction have been proposed [7], [12], [9], [3], [8], [13], [15], [6], [5], [11]. These methods can be classified based on the prediction target. [8], [13] focus on static IR-drop prediction of power nodes in the PDN; [15], [6], [5], [11] focuses on tile-based dynamic IR-drop prediction; [7], [12], [9], [3] focus on cell-based dynamic IR-drop prediction. We review the methods in [7], [12], [9], [3] in more detail, as we also focus on cell-based dynamic IR-drop prediction.

The method in [7], [12], [9] aims to accelerate the estimation of IR-drop in a modified circuit after cell moving-based ECO. In [7], numerous small regional XGBoost models are constructed to predict dynamic IR-drop results post-ECO. [12] introduces an Artificial Neural Network model for predicting dynamic IR-drop post-ECO. [9] proposes a dynamic IR-drop ECO optimization through cell movement, and uses a Random Forest model to predict dynamic IR-drop post-ECO to guide the cell movement process. Meanwhile, the work in [3] aims to speed up the vector-based IR-drop analysis by using an XGBoost model trained by certain vectors and predicting the other vectors.

Despite their innovations, [7], [9], [3] rely on specific neighboring features (e.g., $21x11$ partitions, rhombus-shaped 13 partitions, rhombus-shaped 13 vias) to capture the environmental effect on IR-drop. While effective, this approach may limit tree-based models' ability to learn relationships between neighboring features, potentially leading to confusion. Additionally, the neighboring features in [12] do not account for the simultaneous switching of the target cell and surrounding cells.

A shared drawback of [7], [12], [9] is their reliance on partial simulations to update the features of modified circuits before prediction. The time required for partial simulations is comparable to that for IR-drop simulations, making it time-intensive for the industry to evaluate multiple ECO candidates within an IR sign-off iteration.

Furthermore, [7], [12], [9] treat the revised circuit post-ECO as an independent entity, neglecting its relationship with the pre-ECO circuit. This oversight could result in the model missing crucial information.

III. Proposed IR Sign-off Flow

In this section, we first provide an overview of the proposed IR sign-off flow and present how the proposed methods facilitate IR sign-off. Next, we present the proposed features and the extraction method. Then, we present the proposed ML model for dynamic IR-drop prediction. Finally, we present an estimation method to update the features of the revised circuit after ECO for obtaining real-time IR-drop prediction.

A. Overview

The proposed IR sign-off flow is shown in Fig. 2. Given an initial circuit with its IR-drop report, we first extract features and IR-drop labels to train an XGBoost model as a dynamic IR-drop predictor. Meanwhile, designers start the process of cell moving-based ECO manually or automatically with an ECO tool for fixing the IR-drop violations. When using an ECO tool for automatic cell location adjustment, three key parameters influence its behavior: (1) the $design\ rule\ violation\ (DRV)$, which establishes the trade-off between timing- and IR-drop-aware ECO, (2) the $maximum\ distance$, which determines the furthest allowable displacement of violation cells, and (3) the $padding\ size$, which specifies the blockage size surrounding violation cells. Designers cannot expect to achieve the best cell movement result in a single attempt. Thus, they configure various parameter combinations to generate a set of ECO candidates for exploration, where the number of candidates, denoted as n, can vary widely. Then, we utilize the fast feature update technique to estimate the features of these candidates, enabling the trained model to predict the dynamic IR-drop results. This process, from feature update to IR-drop prediction, can be completed rapidly and eliminates the need for simulation, so that the parallel evaluation of all ECO candidates is allowed. Finally, designers select the most suitable ECO candidate based on the predicted IR-drop and timing results reported by *Tweaker* [14] to proceed to the next iteration.

B. Feature Extraction

Both the target cell and its surrounding environment can impact its dynamic IR-drop. We categorize the extracted features into cell-based features and tile-based features. Cell-based features represent the electrical characteristics of the cell, while tile-based features capture effects from the neighboring cells.

1) Cell-based features: The cell-based features listed below capture the raw features relevant to the dynamic IR-drop of the target cell itself.

- *Cell loc* (x, y): The lower-left coordination of the cell
- *Cell size* (w, h): Cell width and cell height
- C_{load}: Load capacitance which is the sum of the net capacitance and the output pin capacitance

- I_{peak}: Peak current which is the maximum current during the simulation
- P_{switch}: Switching power which is caused by the charging and discharging of C_{load}
- TR_{out}: Output toggle rate which is the average toggle count per cycle of the output pin
- $R_{eff}(2)$: Effective resistance which is the equivalent path resistance from the cell's VDD (VSS) pin to the voltage source within the die
- $R_{bump}(2)$: The resistance of the bump that the cell's VDD (VSS) pin connects with
- $L_{bump}(2)$: The inductor of the bump that the cell's VDD (VSS) pin connects with

Among the features, C_{load}, I_{peak}, P_{switch}, and TR_{out} can be extracted after power analysis, and R_{eff} can be extracted after rail analysis. R_{bump} and L_{bump} reflect the package effect on IR-drop. They can be extracted using a commercial package extraction tool and remain constant before and after ECO.

2) Tile-based features: Nearby cells have a significant impact on the dynamic IR-drop of the target cell, particularly those that switch simultaneously with the target cell. To effectively capture the neighboring information, we generate the tile-based features in three steps. We first construct the local tile-based features [7], [3], [9]. Then, we transform them to the global tile-based features. Finally, among them, we select the features that are highly correlated with IR-drop as our final tile-based features.

In the first step, we create a $tile\ map$ by dividing the design into tiles and assign each cell to the corresponding tile based on its *cell loc*. The $tile\ map$ records the physical information of cells in the layout. For a target cell, its neighboring tiles are the $5x5$ tiles surrounding the tile where the cell is located. We then calculate the tile value by summing up the features of cells in each neighboring tile. To consider the simultaneous switching effect on dynamic IR-drop, we sum only the features related to timing, including peak current (I_{peak}), switching power (P_{switch}), internal power $(P_{internal})$, output toggle rate (TR_{out}), and IR-drop (IR). Moreover, we include only cells whose timing window (tw) overlaps with the target cell's tw in the summation. The inclusion of IR-drop is based on our observation that the ECO tool often mitigates IR-drop by moving cells from regions of higher IR-drop to those of lower IR-drop, indicating a strong correlation between the cell's IR-drop and the regional IR-drop. Now, there are 125 local tile-based features (5 timing-related features $\times$ 25 neighboring tiles).

In the second step, for each timing-related feature, we extract the following three global tile features among the neighboring tile features.

- *Min. tile feature*: the minimum tile among the neighboring tiles
- *Max. tile feature*: the maximum tile among the neighboring tiles
- *Overall tile feature*: sum of the neighboring tiles

The previous works [7], [3], [9] directly use all the neighboring tiles as features, which could confuse the XGBoost model. Fig.3 demonstrates one example. Assume that there are three cells, each surrounded by 3x3 neighboring tiles. Two of them (left and middle) are IR-drop violations and the other (right) is a non-violation. In the neighboring tiles, the red one indicates a high-power tile and the green one indicates a low-power tile. When the XGBoost model grows the tree by splitting nodes for training, one node considers only one feature at a time. Suppose the model is using $tile_3$ to split a node. The model learns that: (1) a high-power tile leads to a violation; (2) a low-power tile leads to a violation; (3) a low-power tile leads to a non-violation. Obviously, there are conflicts between (1) and (2), and between (2) and (3), which confuses the model. This is mainly because the model learns the neighboring tiles separately, lacking a global view of them. Therefore, rather than learn all neighboring tiles in a local manner, we use *min.*, *max.* and *overall* tile features which have taken the relationship between neighboring tiles into account, providing the XGBoost model with the global view. In the above example, *max tile feature* and *overall tile feature* cause zero conflict. We currently have 15 (5 timing-related features x 3 tiles) tile-based features.

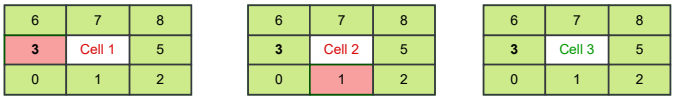


Fig. 3: Neighboring tiles of three cells

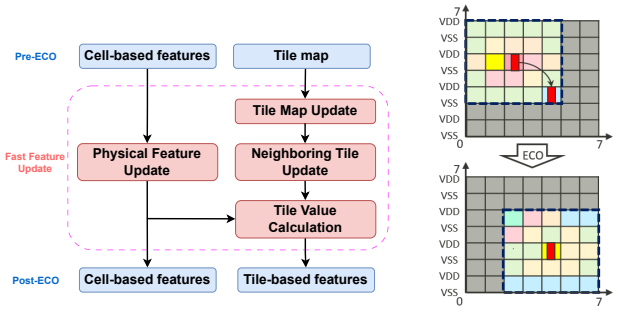
TABLE I: Selected tile-based features

CC rank	Tile-based features	Description
1	*Max. IR tile*	The maximum tile among all neighboring tiles of IR-drop
2	*Overall IR tile*	Sum of all neighboring tiles of IR-drop
3	*Min. IR tile*	The minimum tile among all neighboring tiles of IR-drop
4	*Overall P_{switch} tile*	Sum of all neighboring tiles of switching power
5	*Overall $P_{internal}$ tile*	Sum of all neighboring tiles of internal power
6	*Overall TR_{out} tile*	Sum of all neighboring tiles of output toggle rate
7	*Max. TR_{out} tile*	The maximum tile among all neighboring tiles of output toggle rate

In the final step, to mitigate the curse of dimensionality and overfitting during learning, we compute the correlation coefficients (CC) between each tile-based feature and the golden IR-drop, and select the top half of these features as our final set of tile-based features. The selected tile-based features are listed in TABLE I, and the results reveal that the regional IR-drop is highly correlated with the cell's IR-drop, which is overlooked by previous works. In the end, our final feature set has a total of 21 features (14 cell-based features + 7 tile-based features). Compared with previous works, we use fewer features to represent the surrounding information more globally.

C. XGBoost-Based IR-Drop Predictor

XGBoost [4] is a decision tree ensemble based on gradient boosting and it is a state-of-the-art machine learning model utilized for supervised learning. In this work, we use XGBoost as our regression model since (1) it is faster than other supervised learning such as Random Forest and Neural Network (NN) models; (2) it can achieve IR-drop prediction for each cell; (3) it is explainable, making it easier to optimize; (4) XGBoost-based IR-drop predictor is used in existing works, such as [7], [8], [13], and shows the competitive result of prediction.



(a) Flowchart

(b) Example of tile-based feature update

Fig. 4: Fast feature update

We have a training dataset $D = \{(x_i, y_i) : |D| = N, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}\}$, where N is the number of cells and m is the number of features in our final feature set ($m = 21$). $x_i = (x_i^{(1)}, x_i^{(2)}, ..., x_i^{(m)})^T$ denotes the i_{th} cell's feature set. y_i denotes the golden IR-drop of the i_{th} cell. Let $\hat{y_i}$ be the predicted IR-drop of the ensemble tree model, which is generated as follows:

$$\hat{y_i} = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), f_k \in F \tag{1}$$

f_k denotes the function of the k_{th} tree model. K is the number of tree models. F represents the set of functions of all trees. We need to find the best set of functions above by minimizing the objective function as follows:

$$L(\phi) = \sum_{i=1}^{N} l(y_i, \hat{y_i}) + \sum_{k=1}^{K} \Omega(f_k) \tag{2}$$

l is a loss function that indicates the difference of the predicted IR-drop $\hat{y_i}$ and the golden IR-drop y_i, and Ω is a measure of model complexity to avoid overfitting. We employ asymmetric Mean Square Error (MSE) as our loss function to deal with the imbalanced data set. When violation cells are underpredicted, we apply a penalty to MSE as shown in Eq. 3.

$$\alpha \left(1 + \frac{y_i - \hat{y_i}}{y_i - thrs}\right) \cdot \|\hat{y_i} - y_i\|^2 \tag{3}$$

α is a penalty factor and $thrs$ is the violation threshold. Within the fraction part, the numerator indicates the magnitude of underprediction, and the larger value results in a greater penalty. The denominator indicates how close the golden IR-drop is to the threshold. The closer value results in a greater penalty as it is more likely to be misclassified as non-violation.

D. Fast Feature Update

After designers perform cell moving-based ECO, the features corresponding to the moved cells change. It is time-consuming to re-conduct power analysis and partial rail analysis to update the new feature values for prediction. Thus, an alternative to obtain the new feature values without the need to run simulations is desirable.

TABLE II: Design information

Technology node	3nm
Cell count	1,033,849
Chip size	6984×4135 $(\mu m \times \mu m)$
Supply voltage	$0.8V$
IR violation threshold	$120mV$

We propose a simple yet powerful feature update method that utilizes the original information before cell moving to estimate the new feature values after cell moving. The flowchart is shown in Fig. 4(a), where the left and right parts are responsible for the cell-based and tile-based feature updates, respectively. In the cell-based feature update, since moved cells do not change their cell types, the electrical characteristics of the cell itself are affected slightly. Therefore, given the cell-based features of the initial circuit as input, we only update the physically related features, including cell location and cell size, which can be immediately obtained without simulation. The other cell-based features remain the same as those before ECO.

As for the tile-based feature update, moving cells mainly alters the relationships between cells and their surrounding cells, so the tile-based features, which represent the neighboring information, are our focus for update. There are three steps. Firstly, given the $tile\ map$ constructed in the feature extraction stage, we update it to account for layout changes resulting from moved cells. This is accomplished by removing each moved cell from its original tile before ECO and adding it to the new tile it moves to after ECO. Secondly, due to the update of $tile\ map$ or the movement of cells, the neighboring tiles for both moved and non-moved cells may change and need to be updated. Thus, we iterate over each cell, determine the corresponding tile where the cell is located, and redefine the 5x5 tiles surrounding the tile in the updated tile map as the new neighboring tiles. Finally, after updating the neighboring tiles, we calculate the tile value and extract the tile-based features listed in TABLE I, as described in Section III-B2. Note that the timing-related features, I_{peak}, P_{switch}, $P_{internal}$, TR_{out}, and IR of cells used to calculate the tile value, are taken from the input cell-based features, as these features are considered unchanged after ECO in our estimation method. In Fig. 4(b), we use a small tile-based design to demonstrate the tile-based feature update. Before ECO, the red cell was originally in $tile_{2,4}$, and the surrounding 5x5 tiles were its neighboring tiles. After ECO, the red cell moves to $tile_{4,2}$, and the colors of $tile_{2,4}$ and $tile_{4,2}$ change due to the tile map update. Then, the new neighboring tiles are updated to the 5x5 tiles surrounding $tile_{4,2}$.

The time complexity of the fast feature update is dominated by the tile value calculation, where each cell needs to interact with its neighboring cells to determine tw overlaps and summation. The number of neighboring cells is bounded by a constant because it is independent with the cell count n in the design. Therefore, the estimation method can update the features in linear time.

IV. Experimental Results

A. Experiment Setup

In the experiments, we used Cadence Voltus [1] to conduct power analysis and rail analysis to obtain the features and the

TABLE III: ECO candidates

ECO version	# Moved cells	# Moved cells in Δ IR intervals (mV)			# Fixed cells	Max IR-drop (mV)
		0-20	20-40	40-80		
ECO0	-	-	-	-	-	162.62
ECO1-1	184	52	76	56	168	142.64
ECO1-2	195	74	75	46	167	139.01
ECO1-3	192	56	86	50	166	142.40
ECO1-4	159	55	60	44	133	144.64
ECO1-5	179	75	72	32	152	155.67

TABLE IV: Training data preprocessing

Preprocessing step	# Cells	# Vio.	# Non-vio.	#Vio. : #Non-vio.
Initial	1,033,849	233	1,033,616	1:4436
Create regions	12,219	233	11,986	1:51
Oversampling	13,184	1,198	11,986	1:10

IR-drop report. We utilized Synopsys Tweaker [14] as the ECO tool for automated cell moving. Our method was implemented on a machine with an NVIDIA GeForce RTX 3090 GPU.

We evaluated the proposed methods on an industrial design with $3nm$ technology. The information about the design is shown in TABLE II. A cell is considered a violation cell if its IR-drop exceeds $120mV$, which corresponds to 15% of the supply voltage $0.8V$. To illustrate the scenario where our proposed methods can concurrently explore multiple ECO versions within a single iteration, we took the industrial design as the initial design, denoted as ECO0, and performed cell moving separately on ECO0 to generate five ECO candidates, ECO1-1 to ECO1-5. ECO1-1 to ECO1-3 were generated through manual cell moving by designers, and ECO1-4 and ECO1-5 were generated with Tweaker.

TABLE III presents the statistics of these ECO versions. Column 2 shows the number of moved cells in each ECO version. All the moved cells in ECO1-1 to ECO1-5 are originally violation cells in ECO0. Columns $3 \sim 5$ show the numbers of moved cells within different IR-drop variation intervals caused by ECO. Column 6 shows the number of fixed cells in each ECO version. The final column shows the maximum IR-drop value of a cell in each ECO version. For example, in ECO1-1, there are 184 moved cells after ECO. Among them, there are 52, 76, and 56 cells whose IR variations are within $0 \sim 20mV$, $20 \sim 40mV$, and $40 \sim 80mV$, respectively. Additionally, 168 out of the 184 violation cells are fixed. In ECO1-1, the maximum IR-drop value is $142.64mV$.

B. Real-Time IR-Drop Prediction for ECOs

We trained the dynamic IR-drop predictor with ECO0 and tested it on moved cells in the other ECO versions. Since the industrial design is nearing tape-out with only a few violation cells, the training data is highly imbalanced, which may lead to a biased model. Therefore, we preprocess the training data to address the issue. The changes in the cell numbers are outlined in TABLE IV. First, we frame a $6\mu m \times 6\mu m$ region around each violation cell, and only the cells covered in the regions are collected to be the training data. This significantly reduces the number of non-violation cells, making a more balanced dataset. Second, due to the limited number of violation cells, we utilize the Synthesized Minority Oversampling Technique (SMOTE) [2] to generate pseudo-violation cells.

TABLE V shows the prediction results. In the five ECO versions, the majority of moved cells are fixed. To check if the trained model can capture the trend within them, we define the confusion matrix as follows:

- TP: Fixed cells are predicted as fixed cells.
- FP: Non-fixed cells are predicted as fixed cells.
- TN: Non-fixed cells are predicted as non-fixed cells.
- FN: Fixed cells are predicted as non-fixed cells.

$Recall = TP/(TP + FN)$ indicates the proportion of fixed cells that were correctly detected; $Precision = TP/(TP + FP)$ indicates the proportion of cells correctly identified as fixed among all cells detected as fixed. We achieve an average recall of approximately 96% and an average precision of approximately 93%, demonstrating that our predictor can effectively capture the IR-drop variations of the moved cells through fast feature update. Then, we evaluate the prediction accuracy with mean absolute error (MAE), maximum absolute error (MaxE), and correlation coefficient (CC). The MAE is further classified into three classes to analyze the errors of all, fixed, and non-fixed cells within moved cells, respectively. The average MAE is $8.75mV$ for all moved cells, $7.41mV$ for non-fixed cells, and $8.96mV$ for fixed cells. The MAE for fixed cells is higher than that for non-fixed cells. This is mainly because fixed cells have larger IR-drop variations, compared with non-fixed cells after ECO, posing a greater challenge for prediction. Even so, the average MAE for fixed cells is less than $10mV$, which is within the acceptable error range for the industry.

For the evaluation time comparison, Voltus spends about 2 hours on power analysis and rail analysis, while our real-time method only spends 1.5 minutes on fast feature update and IR-drop inference.

Fig. 5 shows the scatter plots of the prediction results of five ECO candidates. x-axis is the golden IR-drop and y-axis is the predicted IR-drop. The red line is the perfect prediction line, and the two black lines next to it form the $\pm$ $10mV$ error range. The two green lines, which are the IR violation thresholds, divide the scatter plot into four quadrants. Before ECO, we can imagine that all moved cells (green dots) are in the upper right quadrant, and most of them should move to the lower left quadrant after ECO. As shown in the scatter plots, we successfully capture the IR-drop changes, predicting the majority of cells to the lower left quadrant.

Fig. 6 visualizes the relations among IR-drop before ECO (red), golden IR-drop after ECO (green), and predicted IR-drop after ECO of moved cells in ECO1-2. The x-axis represents the index of the moved cell, with a total of 195 cells, and the y-axis represents the IR-drop value.

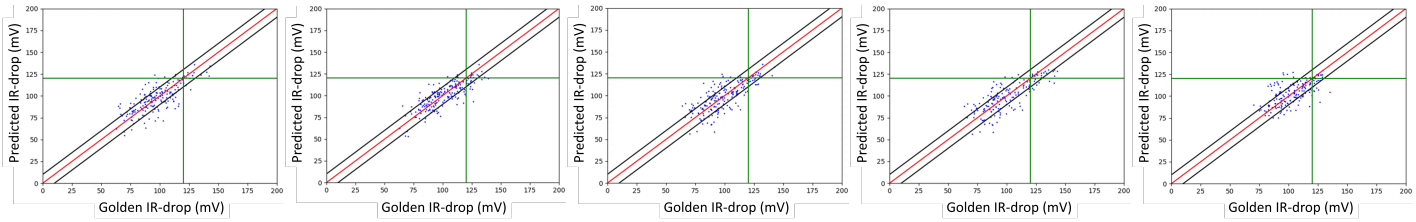
C. Local vs. Global Tile-based Features

In Section III-B2, we introduced the features to capture neighboring information. The local tile-based features are first constructed. Then we propose the global tile-based features, and then further extract the global tile-based features having higher CC with IR-drop.

TABLE VI takes the local approach adopted from the work [7] as a baseline and compares our two global approaches with the baseline using the metrics MAE and CC. For the local approach, the average MAE is $10.96mV$ exceeding the acceptable error range of $10mV$. The average CC is 64% and the CC in ECO1-5 is even lower than 50%. In contrast, for our global approach, the average MAE is $9.75mV$ and the average CC is 70.5%, where the improvements are 11% and

TABLE V: Prediction results of moved cells after ECO

ECO cand.	# Moved cell	# Fixed cell	TP	FP	TN	FN	**Recall**	**Precision**	All MAE (mV)	Non-fixed MAE (mV)	Fixed MAE (mV)	MaxE (mV)	CC	Voltus eval. time	Our eval. time
ECO1-1	184	168	162	3	13	6	0.964	0.982	8.84	7.20	9.00	31.08	0.760	127.10m	1.45m
ECO1-2	195	167	162	15	13	5	0.970	0.915	7.60	8.33	7.48	33.23	0.810	100.97m	1.45m
ECO1-3	192	166	161	11	15	5	0.970	0.936	9.08	5.62	9.62	32.87	0.757	129.72m	1.43m
ECO1-4	159	133	129	8	18	4	0.970	0.942	9.68	8.22	9.96	30.01	0.773	99.77m	1.47m
ECO1-5	179	152	142	14	13	10	0.934	0.910	8.56	7.69	8.72	33.39	0.641	127.23m	1.52m
Avg.	182	157	151	10	14	6	**0.962**	**0.937**	8.75	7.41	8.96	32.12	0.748	116.95m	1.47m



(a) ECO1-1 (b) ECO1-2 (c) ECO1-3 (d) ECO1-4 (e) ECO1-5

Fig. 5: Scatter plots of moved cells after ECO

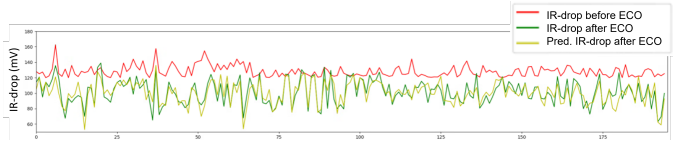


Fig. 6: Line chart of moved cells after ECO

TABLE VI: Tile-based features comparison

ECO cand.	Local [7]		Global		Global with CC ranking	
	MAE (mV)	CC	MAE (mV)	CC	MAE (mV)	CC
ECO1-1	11.01	0.630	9.85	0.720	8.84	0.760
ECO1-2	10.65	0.679	8.73	0.764	7.60	0.810
ECO1-3	10.75	0.674	9.83	0.723	9.08	0.757
ECO1-4	10.54	0.731	10.63	0.737	9.68	0.773
ECO1-5	11.86	0.484	9.72	0.580	8.56	0.641
Avg.	10.96	0.640	9.75	0.705	8.75	0.748
Impr.(%)	-	-	11.0	10.2	20.2	16.9
# Tile features	125		15		7	

10.2%, respectively, compared to the baseline. Furthermore, when we further select the top half of the features based on the CC ranking, the average MAE is $8.75mV$ and the average CC is 74.8%, showing the improvements of 20.2% and 16.9%, respectively, compared to the baseline. Additionally, the number of features in these three approaches decreases from 125 to 15, and finally to 7. In summary, by employing global tile-based features and removing features that are less relevant to IR-drop, we can not only reduce the feature dimension but also make significant improvements in accuracy for the XBGoost predictor.

D. Evaluation Time Comparison

TABLE VII compares the evaluation time of Voltus, traditional ML, and our real-time ML in ECO1-1 with 1.03 million cells. In Voltus, the evaluation time includes power analysis and rail analysis; In traditional ML, it includes (1) partial simulations: run power analysis and partial rail analysis, (2) feature extraction: transform raw data to the features, and (3) ML inference: IR-drop prediction; In real-time ML, it includes fast feature update and ML inference. Our real-time ML takes only 1.45 minutes for each ECO candidate, achieving 88X speedup over Voltus and 65X speedup over traditional ML.

TABLE VII: Evaluation time comparisons

Step	Voltus	Traditional ML	Real-time ML
Partial simulations	127.1 mins	87 mins	1.45 mins
Feature extraction		7.1 mins	
ML inference		0.014 secs	0.014 secs
Eval. time	127.1 mins	94.1 mins	1.45 mins

Additionally, considering the real-world scenario in industrial IR sign-off flow, designers must experience multiple ECO candidates within an iteration. Thus, the evaluation time for multiple ECO candidates is crucial. Due to the time and resource-intensive nature of simulations, both Voltus and traditional ML (Fig. 1) are unable to evaluate multiple ECO candidates in parallel. Therefore, if there are n ECO candidates, the evaluation time of Voltus would be $127.1 \times n$ minutes, and that of traditional ML would be $87 \times n + 7.1$ minutes. In contrast, since real-time ML (Fig. 2) does not require any simulations, ECO candidates can be evaluated in parallel, so that the evaluation time would ideally remain 1.45 minutes.

The result shows that using Voltus or traditional ML to the industrial IR sign-off flow is very time-consuming and designers have to sacrifice considering multiple ECO candidates to speed up the process. In contrast, our real-time ML can not only obtain the IR-drop results in a very short time but also allow designers to explore more ECO candidates within an iteration, increasing the probability of selecting a better ECO version for the next iteration.

V. Conclusion

We propose a real-time dynamic IR-drop estimation method based on an XGBoost model equipped with the global view of cells' neighboring tiles. After ECO, our method produces the dynamic IR-drop result without the need for any simulations. This enables designers to concurrently evaluate multiple ECO candidates within a single iteration to facilitate IR sign-off. The results show that we can capture the dynamic IR-drop variations in five ECO candidates in a very short time compared to Voltus and the traditional ML predictors, with $< 10mV$ MAE overhead.

REFERENCES

[1] Cadence. Voltus.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Over-sampling Technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002.

[3] Jia-Xian Chen, Shi-Tang Liu, Yu-Tsung Wu, Mu-Ting Wu, Chieo-Mo Li, Norman Chang, Ying-Shiun Li, and Wen-Tze Chuang. Vector-based Dynamic IR-drop Prediction Using Machine Learning. In *2022 27th Asia and South Pacific Design Automation Conference*, pages 202–207, 2022.

[4] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[5] Vidya A. Chhabria, Vipul Ahuja, Ashwath Prabhu, Nikhil Patil, Palkesh Jain, and Sachin S. Sapatnekar. Thermal and ir drop analysis using convolutional encoder-decoder networks. In *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 690–696, 2021.

[6] Vidya A. Chhabria, Yanqing Zhang, Haoxing Ren, Ben Keller, Brucek Khailany, and Sachin S. Sapatnekar. MAVIREC: ML-Aided Vectored IR-Drop Estimation and Classification. In *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1825–1828, 2021.

[7] Yen-Chun Fang, Heng-Yi Lin, Min-Yan Sui, Chien-Mo Li, and Eric Jia-Wei Fang. Machine-learning-based Dynamic IR Drop Prediction for ECO. In *2018 IEEE/ACM International Conference on Computer-Aided Design*, page 1–7. IEEE Press, 2018.

[8] Chia-Tung Ho and Andrew B. Kahng. IncPIRD: Fast Learning-Based Prediction of Incremental IR Drop. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[9] Xuan-Xue Huang, Hsien-Chia Chen, Sheng-Wei Wang, Iris Hui-Ru Jiang, Yih-Chih Chou, and Cheng-Hong Tsai. Dynamic IR-Drop ECO Optimization by Cell Movement with Current Waveform Staggering and Machine Learning Guidance. In *2020 IEEE/ACM International Conference On Computer Aided Design*, pages 1–9, 2020.

[10] Santanu Kundu, Manoranjan Prasad, Sashank Nishad, Sandeep Nachireddy, and Harikrishnan K. MLIR: Machine Learning based IR Drop Prediction on ECO Revised Design for Faster Convergence. In *2022 35th International Conference on VLSI Design and 2022 21st International Conference on Embedded Systems*, 2022.

[11] Yonghwi Kwon, Giyoon Jung, Daijoon Hyun, and Youngsoo Shin. Dynamic ir drop prediction using image-to-image translation neural network. In *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2021.

[12] Shih-Yao Lin, Yen-Chun Fang, Yu-Ching Li, Yu-Cheng Liu, Tsung-Shan Yang, Shang-Chien Lin, Chien-Mo Li, and Eric Jia-Wei Fang. IR drop prediction of ECO-revised circuits using machine learning. In *2018 IEEE 36th VLSI Test Symposium (VTS)*, pages 1–6, 2018.

[13] Chi-Hsien Pao, An-Yu Su, and Yu-Min Lee. XGBIR: An XGBoost-based IR Drop Predictor for Power Delivery Network. In *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 1307–1310, 2020.

[14] Synopsys. Tweaker.

[15] Zhiyao Xie, Haoxing Ren, Brucek Khailany, Ye Sheng, Santosh Santosh, Jiang Hu, and Yiran Chen. PowerNet: Transferable Dynamic IR Drop Estimation via Maximum Convolutional Neural Network. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 13–18, 2020.

A Systematic Approach for Multi-objective Double-side Clock Tree Synthesis

Xun Jiang[1], Haoran Lu[1], Yuxuan Zhao[2], Jiarui Wang[1,3], Zizheng Guo[1], Heng Wu[1], Bei Yu[2], Sung Kyu Lim[4], Runsheng Wang[1,5,6], Ru Huang[1,5,6] and Yibo Lin[1,5,6*]
[1]School of IC, Peking University [2]Department of CSE, The Chinese University of Hong Kong
[3]School of CS, Peking University [4]School of ECE, Georgia Institute of Technology
[5]Institute of EDA, Peking University, Wuxi [6]Beijing Advanced Innovation Center for Integrated Circuits

***Abstract*—As the scaling of semiconductor devices nears its limits, utilizing the back-side space of silicon has emerged as a new trend for future integrated circuits. With intense interest, several works have hacked existing backend tools to explore the potential of synthesizing double-side clock trees via nano Through-Silicon-Vias (nTSVs). However, these works lack a systematic perspective on design resource allocation and multi-objective optimization. We propose a systematic approach to design clock trees with double-side metal layers, including hierarchical clock routing, concurrent buffers and nTSVs insertion, and skew refinement. Compared with the state-of-the-art (SOTA) methods, the widely-used open-source tool, our algorithm outperforms them in latency, skew, wirelength, and the number of buffers and nTSVs.**

I. Introduction

Back-side interconnection [1], [2] has emerged to continue the scaling of semiconductor technologies. With increasingly congested designs and tight timing budgets on the front-side (FS), both the academia [2]–[9] and industry [10], [11] have started to consider utilizing back-side (BS) resources for routing wires, including signal, power, and clock net. Based on the evaluation in [2], the latency of the clock tree is decreased from 50ps to 20ps with back-side metal layers. However, lacking a systematic double-side clock tree synthesis (CTS) algorithm is impeding further exploration of the potential of back-side resources, considering the new complex design space unfolding before researchers.

As shown in the bottom of Fig. 1, the double-side clock net is jointly implemented with connecting the back-side and front-side metal layers via additional nTSVs. Although timing benefits have been reported, the overhead of nTSVs, buffers, and clock wirelength cannot be neglected, as the primary objectives of CTS has included latency, skew, and resource consumption. For instance, many methods, e.g., clock routing [12]–[15], buffer insertion [16]–[19], useful skew [20]–[22], Flip-Flop (FF) clustering [23]–[25] and 3D clock tree [26]–[28] have all once referred to as timing and power optimization techniques involving clock trees. However, considering the challenges of more complex design resource allocation and multi-objective optimization, the exploration of double-side CTS is still insufficient to keep pace with advanced technology.

Existing works [2], [6], [7], [29] investigating double-side CTS all follow the incremental flow shown in the left of Fig. 1. To elaborate, Veloso et al. [2] flip the high-level part of a clock tree to the back-side to reduce clock latency to the maximum extent. Bethur et al. [7] propose leveraging the fanout of driven sinks as the criteria to decide whether the net should be flipped to back-side. Bethur et al. [6] utilize the Graph Neural Network to select the subset of FFs with poor timing performance and flip the connected nets to the back-side. Vanna-iampikul et al. [29] incorporate the back-side design methodologies of Power Delivery Network (PDN) based on the clock synthesis method by Bethur et al. [6]. Although the

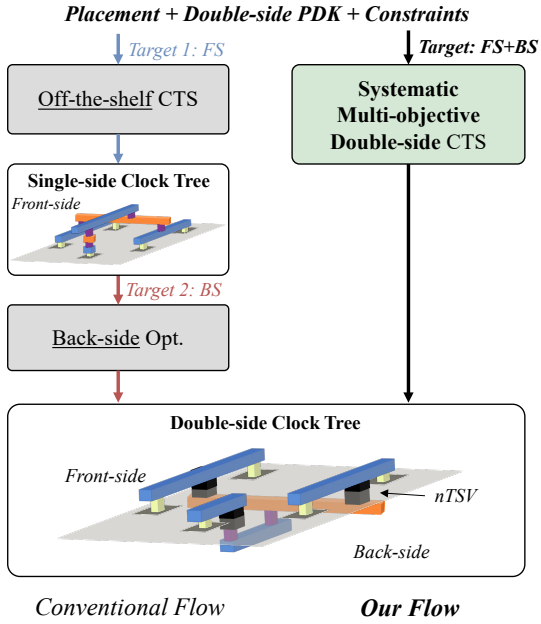


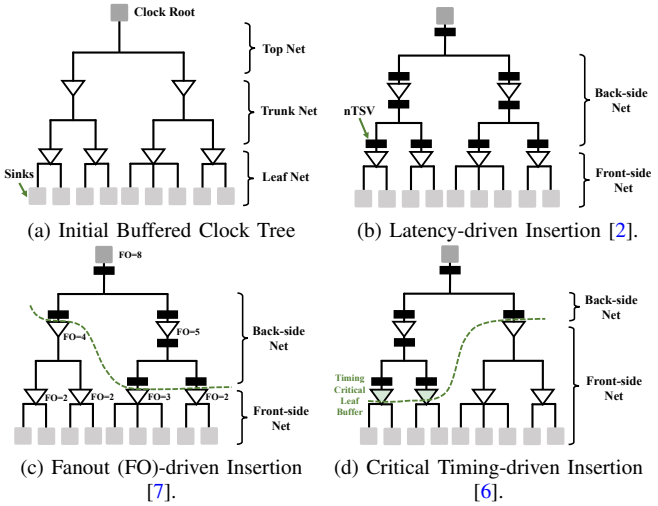
Fig. 1 Clock tree synthesis with double-side metal layers.

benefits of back-side metal layers to CTS are shown explicitly, the incremental flow cannot fully uncover their potential. For instance, the pre-generated single-side clock tree by the off-the-shelf CTS tool only considers the timing model based on the front-side technology parameters to guide the buffer insertion process, whose buffering solutions will deviate from the best one. The follow-up back-side optimization method has to obey these results to assign nets to back side by inserting nTSV, which can severely harm the eventual solution quality due to limited solution space. Thus, a unified design space of buffer and nTSV insertion with efficient solution exploration is urgent for double-side CTS, which is also the core of our work.

In this work, we aim at unleashing the potential of the back-side technology by handling the challenges from double-side CTS systematically. The major contributions of this work are as follows:

- We propose a systematic double-side clock tree synthesis framework aiming at pushing the boundary of the cutting-edge back-side technology exploration by multi-objective optimization.
- We propose an efficient hierarchical clock routing to reduce clock wirelength and preserve balanced structure.
- We propose a concurrent buffer and nTSV insertion based on multi-objective dynamic programming (DP) and an efficient resource-aware end-point buffer insertion as post-processing method for latency, skew, and resource usage optimization.
- We propose a novel methodology of design space exploration

*Corresponding author: Yibo Lin (yibolin@pku.edu.cn)



(a) Initial Buffered Clock Tree (b) Latency-driven Insertion [2].

(c) Fanout (FO)-driven Insertion [7]. (d) Critical Timing-driven Insertion [6].

Fig. 2 Comparison between the buffered clock tree and the double-side clock trees by different methods to assign top and trunk nets to the back side.

(DSE) of double-side CTS by our framework.

Experimental results demonstrate our superiority over recent works [2], [6], [7], [29]. Take the method [2] with extreme optimization on latency as an example, we can optimize the clock latency by 2.223×, skew by 2.464×, number of buffers by 1.010×, clock wirelength by 1.249×, and number of nTSVs by 1.441×, respectively, with 6.922× speed-up.

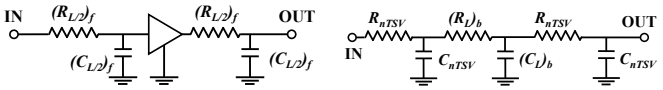
The rest of this paper is organized as follows. In Section II, we demonstrate the preliminaries of our work. In Section III, we explain the details of our algorithms. In Section IV, we present the results of our work. In Section V, we conclude our work and discuss further works.

II. Preliminaries

A. Double-side Clock Tree Structure

The comparison of structure between initial buffered clock tree and double-side clock trees are drawn in Fig. 2. In Fig. 2(a), the initial buffered clock tree assign all leaf nets and trunk nets are on the front side. The leaf net includes clock sink pins, while the trunk net encompasses all other nets excluding leaf nets. Top nets, defined by designers as highest-level trunk nets, can be distinguished from trunk nets for clarity.

We demonstrate three recent post-CTS methods to implement double-side clock tree in Fig. 2(b), Fig. 2(c), and Fig. 2(d), respectively. The post-CTS method from [2] assigns the trunk-level nets to the back-side by inserting nTSVs in Fig. 2(b). Owing to the input and output pins of the buffer on the front side, multiple nTSVs are incorporated to maintain connectivity between the front and back sides. The delay from source to sink pins can be reduced by the lower unit resistance and capacitance of the back-side metal layers, as most paths traverse these common trunk-level nets. It is noteworthy that extra nTSVs also increase the resource usage which needs to be carefully considered for the design of the entire chip. Therefore, [7] utilizes the fanout of driven sinks as the criteria to decide whether the nets should be flipped and [6] leverages the timing criticality of leaf driving buffers to decide the back-side nets assignment. In a word, these methods try to trade-off the timing benefits and the nTSV utilization but limited to the methodology of separated buffers and nTSVs insertion.



(a) Inserting Buffer (b) Inserting nTSV

Fig. 3 Delay modeling for buffer and nTSV insertion.

B. Delay Model for Buffers and nTSVs

In the clock tree synthesis with double-side metal layers, buffers and nTSVs are jointly utilized to minimize the latency of the clock tree. We follow previous work [2], [6], [7] to use the classic *L-type* Elmore delay [30] to compute the delay of wires. We set the front-side and back-side metal layer unit capacitance to c_f and c_b and set the front-side and back-side metal layer unit resistance to r_f and r_b. Furthermore, we take a wire segment with the length L and set the output driven capacitance of wire segment to C_d.

As buffer insertion on the front side in Fig. 3(a), we note C_b as the input capacitance of buffer and D_{buf} as the buffer delay. The delay of inserting a buffer at the middle of the wire is denoted as D_{bufOn}, which is computed as follows.

$$\begin{aligned} D_{bufOn} =& r_f \frac{L}{2}(c_f \frac{L}{2} + C_b) + D_{buf} + r_f \frac{L}{2}(c_f \frac{L}{2} + C_d) \\ =& \frac{r_f c_f}{2} L^2 + \frac{r_f (C_b + C_d)}{2} L + D_{buf}. \end{aligned} \tag{1}$$

We use C_{nTSV} and R_{nTSV} to represent the capacitance and resistance of one nTSV. The delay of inserting two nTSVs at the endpoints of the wire segment in Fig. 3(b) is denoted as D_{nTSVOn}, which is computed as follows.

$$\begin{aligned} D_{nTSVOn} =& R_{nTSV}(C_{nTSV} + C_d) + r_b L(c_b L + C_{nTSV} + C_d) \\ &+ R_{nTSV}(2C_{nTSV} + c_b L + C_d) \\ =& (r_b c_b) L^2 + (r_b C_{nTSV} + r_b C_d + R_{nTSV} c_b) L \\ &+ R_{nTSV}(3C_{nTSV} + 2C_d). \end{aligned} \tag{2}$$

In the double-side scenarios, $r_b c_b \ll r_f c_f$ reduce the delay of back-side metal layers. Meanwhile, the buffer can shield the output capacitance to reduce delay and meet maximum driven-capacitance constraint, whereas nTSV cannot. Therefore, the collaborative optimization of buffers and nTSVs insertion is crucial in the double-side CTS.

C. Multi-objective Optimization

Multi-objective optimization refers to solving problems with multiple conflicting objectives by exploring a set of optimal trade-off solutions. The Pareto frontier [31] is the core concept to represent these solutions, where no objective can be improved without degrading another. For the multi-objective optimization, it is essential to leverage Pareto frontier to comprehensively evaluate the performance of algorithms without being influenced by some parameter preferences. The diversity of solutions across the objective space is also important to avoid the algorithm getting stuck in local optimality.

D. Problem Formulation

Problem 1 (Double-side clock tree synthesis)**.** *Given the clock net, double-side metal layers, and candidate cells, e.g., nTSV and buffer, construct a double-side clock tree to optimize multiple objectives, e.g., latency, skew, wirelength, buffer count, and nTSV count, under connectivity and electricity constraints. The decision variables encompass the positions of inserted nTSVs and buffers, as well as their mutual topology relationships within the clock tree.*

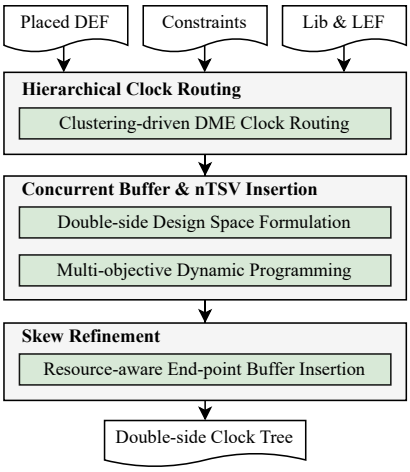


Fig. 4 Overview of our algorithm framework.

In this work, the key challenges are formulating unified double-side CTS design space and developing efficient multi-objective algorithm to explore Pareto-optimal solutions within this design space.

III. Algorithms

A. Overview

The overall flow of our algorithm is shown in Fig. 4. It takes placement results (Placed DEF), PDK, and capacitance and connectivity constraints as input, and output a legal clock tree with buffers and nTSVs. The algorithm mainly consists of three steps: hierarchical clock routing, concurrent buffer and nTSV insertion, and skew refinement (SR). Based on this algorithm, we also support design space exploration of the double-side CTS solutions. We explain each step in details in the following sections.

B. Hierarchical Clock Routing

In the modern CTS, the goal of clock routing is to firstly provide an initial clock tree topology that approximates the optimization of latency, skew, and wirelength. However, many follow-up timing-optimization stages, e.g., buffer insertion and sizing, make latency and skew more resilient to changes in topology, while the wirelength is still largely determined by the clock routing topology and impacts power significantly. We propose a hierarchical clock routing focuses on optimizing wirelength by combining clustering and deferred-merge-embedding (DME) clock routing.

In our hierarchical clock routing, the clustering is performed at two sequential steps, i.e., high-level clustering and low-level clustering, as shown in Fig. 5(a) and Fig. 5(b), respectively. High-level clustering groups the sinks into several large clusters with size H_c by minimizing the total intra-cluster wirelength approximately. Low-level clustering then divides each large cluster into smaller ones with size L_c. The purpose of the dual-level clustering is to obtain a hierarchy according to the spatial proximity of sinks. We also record the centroids of both high- and low-level clustering solutions for later steps. K-means algorithm is adopted as the backbone for both clustering steps. In the experiments, we set H_c to 3,000 and L_c to 30 empirically.

DME is widely used in many clock routing algorithms [13], [14]. It helps to minimize skew and wirelength efficiently. A typical DME is based on matching, as shown in Fig. 5(c). However, such an approach is reported to have poor wirelength when dealing with imbalanced distribution of sinks. Therefore, with the clustering

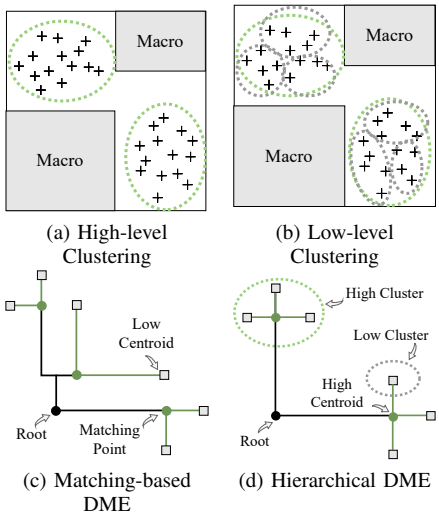


Fig. 5 Dual-level clustering and DME-based clock routing. Symbol "+" refers to sink.

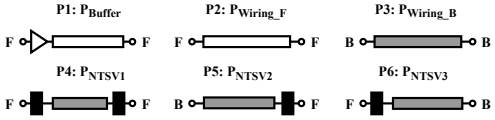


Fig. 6 Patterns of wire segments labeled as P1 $\sim$ P6. **F** refers to the front side. **B** refers to the back side. The **right end** is close to sinks and the **left end** is close to clock root.

results, we perform DME clock routing with the low-level clustering centroids as leafs and the corresponding high-level clustering centroids as root, denoted as hierarchical DME as shown in Fig. 5(d).

C. Concurrent Buffer and nTSV Insertion

In this section, we describe the details of double-side design space formulation and multi-objective DP.

1) Double-side Design Space Formulation: The double-side design space is formulated by the discrete edge patterns and connectivity constraints. Different from the traditional buffer insertion on the single side, the edge patterns in our algorithm should adapt to double sides and the characteristics of buffers and nTSVs. We list six candidate edge patterns, denoted as pattern set P, in the generated clock tree from hierarchical clock routing, as shown in Fig. 6. For instance, since the two pins of nTSVs are situated in different sides, resulting in side types of two endpoints of edge having distinct types as $\mathbf{P_{NTSV2}}$ and $\mathbf{P_{NTSV3}}$. However, the side types of two endpoints of $\mathbf{P_{NTSV1}}$ are still **F** due to two nTSVs flipping side twice. Meanwhile, since the two pins of buffers are located in the front side, the side types of two endpoints of edge have to be **F** as $\mathbf{P_{Buffer}}$. During buffers and nTSVs insertion process, the pattern decisions of adjacent edges cannot violate the **connectivity constraint**, that the shared vertex of any two edges in the clock tree must have the same side type.

2) Multi-objective Dynamic Programming: The multi-objective dynamic programming consists of four steps: build heterogeneous DP graph, bottom-up generation, multi-objective selection, and top-down decision, as shown in Fig. 7. With these steps, we can concurrently insert buffers and nTSVs into the clock tree efficiently.

Step 1 (Build Heterogeneous DP Tree): To build the heterogeneous DP graph, we firstly represent each edge of the clock tree by a node in the DP graph. Two adjacent edges in the clock tree

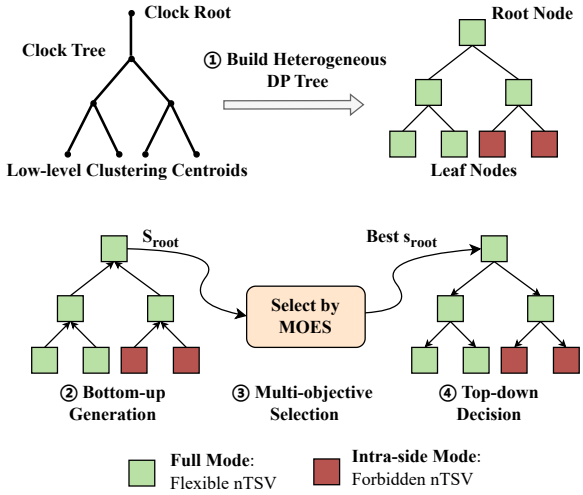


Fig. 7 The process of building DP formulation form clock tree and the bottom-up and top-down with DP execution. Each edge corresponds to two sets marked by different colors.

at different levels are connected by edge in DP graph as shown in Fig. 7. Due to the structure of clock tree, the DP graph is formed as a tree rooted by the node corresponding to the edge of clock root. Then, the pattern selection, i.e., buffer and nTSV insertion, can be conducted on the DP tree. Notice that the clock tree and DP tree are both binary tree.

A novel idea of ours is to further configure the nodes in DP graph with two types of nTSVs inserting mode: full mode (flexible nTSV with $P_1 \sim P_6$ allowed to select) and intra-side mode (forbidden nTSV with only $P_1 \sim P_3$ allowed to select), which could be easily implemented by restricting the allowed patterns on the edges in our framework. Thus, by the inserting mode configurations, we could obtain a heterogeneous DP tree. To support the DP algorithm to explore the solutions in the DP tree, we utilize S to represent the candidate solutions of each node and s to represent the selected final solution of each node.

Step 2 (Bottom-up Generation): In the process of bottom-up generation, we firstly set the undirected edges in DP tree by the topology of clock tree to directed edges, which all point to the root node finally. During the generation process, each node should undergo two operations: merging from two predecessor nodes and inserting in itself. We generated candidate solutions S for all nodes starting from the leaf nodes. For leaf nodes without predecessor nodes, the edge end-point close to sinks are forced to front-side, which restricts the initial insertion of leaf nodes to $\{\mathrm{P}_1, \mathrm{P}_2, \mathrm{P}_4, \mathrm{P}_5\}$ without merging. From the candidate leaf node solutions, we can generate the candidate solutions of successor nodes by merging each solution from one predecessor node with every solution from the other one. Meanwhile, these dependencies are recorded in merged solutions for fast traversal in Step 4. Notice that the merging operation is allowed which two predecessor solutions obeying the connectivity constraint. This rule can ensure we generate a legal double-side clock tree just by one turn of DP without any additional legalization steps, which improves the efficiency of our algorithm.

After the merging operations, each node should conduct the inserting operations to assign patterns based on the merged solutions. The patterns must be selected from the set P, while the selection could be restricted by the specific inserting mode. For instance, if one node is configured to the intra-side mode, the patterns

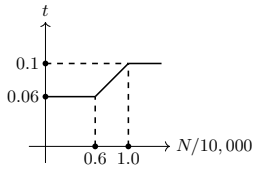


Fig. 8 Adaptive scale factor function $t \sim N/10,000$.

$\{\mathrm{P}_4, \mathrm{P}_5, \mathrm{P}_6\}$ involving nTSVs are forbidden to inserted in that node. By the delay model introduced in Section II-B and [16], we could calculate the effective capacitance and path delay for solutions after inserting patterns. By the iterative execution, we finally obtain the candidate solutions at the root node, denoted as S_{root} at Step 2 in Fig. 7.

Step 3 (Multi-objective Selection): With the generated candidate solutions S_{root}, we try to select the final solution $s_{root} \in S_{root}$ considering multi-objective optimization. With the efficient computation structure of DP, we could easily record the latency, buffer count, and nTSV count for each node during the bottom-up generation.

Thus, we propose a multi-objective enhancement score to approach the multi-objective optimization. With the additional nTSV as resource for insertion, the candidate solutions at the root node have many more combinations of buffers, nTSVs, and different latencies. The distribution of candidate solutions is much more diverse than that in the single-side buffer insertion scenario, as observed in IV-C in the experiment. Therefore, we utilize the multi-objective enhancement score (MOES) to decide the final solution s_{root} as follows.

$$\mathrm{MOES} = \alpha l_{root} + \beta b_{root} + \gamma n_{root}, \tag{3}$$

where l_{root}, b_{root}, and n_{root} are the values of latency, buffer count, and nTSV count for the candidate solutions at the root edge. α, β, and γ are manual parameters to weight each objective.

Step 4 (Top-down Decision): After the decision of s_{root}, we invert the direction of edges in the DP tree for top-down decision. By the recorded dependencies in the merged solutions at Step 2, we can quickly retrace the final solutions for all nodes.

Pruning technique: We extend the *inferior solution* concept in [16], that the effective capacitance and maximum path delay of one solution both worse than those of another solution means this solution will always be viewed as a bad candidate, to the double-side scenarios by pruning candidate solutions at front-side and back-side, respectively. This method ensures our DP algorithm is optimal in terms of latency. Meanwhile, to satisfy the constraints of the max-driven capacitance, we prune the solution with effective capacitance exceeding the maximum threshold.

D. Skew Refinement

As the DP in previous section mainly optimizes clock latency, we further introduce a resource-aware skew refinement technique to mitigate skew degradation by inserting buffers at end-points. This step is triggered when the skew is over $p\%$ of the maximum latency. In the experiments, we set p to 23. N refers to the number of sinks. t refers to an adaptive factor w.r.t. N, as shown in Fig. 8.

1) Set refined end-points number n as $\min(N \times t, m)$. m is the maximum number of refined end-points and set to 33 in the experiments.
2) Refine n end-points in descending order of delay by inserting one buffer at the low-level clustering centroids.

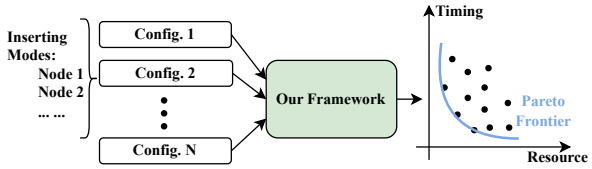


Fig. 9 Design space exploration for double-side CTS.

By the observation that the skew and wire delay within lower clusters have a negligible impact on the overall path, this method is efficient and effective in mitigating skew.

E. Design Space Exploration of Double-side CTS

Based on the concurrent buffer and nTSV insertion approach, we further propose a general double-side design space exploration methodology that demonstrates superiority in multi-objective optimization of double-side CTS. The main idea is to control the inserting modes of nodes in DP tree, as shown in Fig. 7. By setting up various configurations, users could explore more solutions in the objective space, as shown in Fig. 9.

To make the DSE process easy to control, we allow users to control the inserting modes of nodes in DP tree by setting a fanout threshold. Nodes with fanout lower than the threshold will be configured as full mode, while those with fanout larger than the threshold will only allow intra-side mode. Furthermore, more sophisticated methods to control the inserting modes could be incorporated into our framework other than the simple heuristics. The concept of decoupling the DP execution flow and the controlling of nodes inserting allows users to avoid dealing with cumbersome details, e.g., timing calculation, but still have large multi-objective optimization space, which will promote more optimization techniques in double-side CTS problem.

IV. Experimental Results

We perform the experiments on the Linux platform with a 20-core 2.40GHz Intel(R) Xeon(R) Silver 4210R CPU and 320GB memory. We take the ASAP7 PDK [32] to perform our experiments and adopt the unit resistance and capacitance of back-side metal layers and nTSV from [1]. These parameters are listed in TABLE I. We take designs from OpenROAD [33] and use its backend flow to generate benchmarks. The statistics of the benchmarks are listed in TABLE II. Our framework is implemented using C++.

A. Technology Settings

We follow OpenROAD's convention to take the unit resistance and capacitance of M3 for the evaluation of delays in the front-side. For the back-side wires, we compute delays according to the actual usage of layers (`BM1`∼`BM3`). We use the Elmore delay [30], the slew model [34], and the nonlinear delay model (NLDM) [32] for delay computation. In our work, we follow the default flow in OpenROAD where one kind of buffer is used. This is a reasonable setting, because buffer sizing will be further optimized for skew minimization in the follow-up clock tree optimization after clock tree synthesis in a real design flow [33], [35], [36]. We take `BUFx4_ASAP7_75t_R` with a shape of $0.378nm \times 0.27nm$ as the buffer and `nTSV` with a shape of $0.27nm \times 0.27nm$, which is aligned to the other standard cells in the layout. The resistance and capacitance of one `nTSV` are $0.020k\Omega$ and $0.004fF$.

TABLE I Layer resistances and capacitances [1], [32].

Layer	Unit Res. ($k\Omega/\mu m$)	Unit Cap. ($fF/\mu m$)
M1	0.138890	0.11368
M2	0.024222	0.13426
M3	0.024222	0.12918
M4	0.016778	0.11396
M5	0.014677	0.13323
M6	0.010371	0.11575
M7	0.009672	0.13293
M8	0.007431	0.11822
M9	0.006874	0.13497
BM1∼BM3	0.000384	0.116264

TABLE II The statistics of benchmarks [33].

ID	Design	Statistics		
		#Cells	#FFs	Util.
C1	jpeg	54973	4380	0.50
C2	swerv_wrapper	148407	14338	0.40
C3	ethmac	56851	10018	0.40
C4	riscv32i	11579	1056	0.50
C5	aes	29306	2072	0.50

B. Comparison with SOTA Methods

We utilize the open-source tool OpenROAD [33] to evaluate the effectiveness of back-side metal layers and the benefits of our algorithm. We generate the `post-place` and `post-cts` DEF files [37] from OpenROAD and use consistent evaluation methods and parameters for all designs. α, β, and γ are set to 1, 10, and 1. The fanout of [7] is set to 100 and the number of timing critical paths in [6] is set to 0.5. The inserting modes of all edges of our algorithm in TABLE III are set to full mode.

We also implement the method from [2] as the baseline, which extremely optimizes latency. We take the `post-cts` DEF generated by the CTS tools in OpenROAD without resizing operations following CTS. According to [2], we flipped the nets above the low clustering centroids to the back-side by inserting nTSV as illustrated in Fig. 2(b) to minimize latency, denoted as `OpenROAD Buffered Clock Tree` + [2] (`OpenROAD` + [2] for short) in TABLE III.

TABLE III summarizes the comparison between our framework and recent studies. Our algorithm outperforms the method [2] in the clock latency by 2.223×, skew by 2.464×, number of buffers by 1.010×, clock wirelength by 1.249×, and number of nTSVs by 1.441×. The significant reduction in latency and #nTSVs comes from our hierarchical clock routing and concurrent buffer and nTSV insertion. We also compare with recent studies [2], [6], [7] using the buffered clock tree generated by our framework, to demonstrate the effectiveness of the concurrent buffer and nTSV insertion. We can see that our algorithm could achieve better quality in almost all objectives. In addition, our algorithm is efficient. We achieve 6.992× speedup over `OpenROAD` + [2].

C. Effectiveness of MOES

In Fig. 10, we validate the effectiveness of MOES (Section III) in concurrent buffer and nTSV insertion compared with the solo buffered clock tree. The points show the best results achieved by using MOES and minimal latency deviate far away in the double-side scenario (see two triangles), while they are much closer in the single-side scenario (see two squares). The reason is that the double-side scenario enlarges the design space, where more combinations of buffers and nTSVs will be preserved at the end of DP and should be carefully considered. Therefore, an objective function considering holistic factors, like MOES, is necessary to improve the solution quality for double-side CTS.

TABLE III Comparison with recent studies on clock tree synthesis with back-side metal layers.

Design	OpenROAD Buffered Clock Tree†				OpenROAD Buffered Clock Tree + [2]						Ours					
	Latency (*ps*)	Skew (*ps*)	Buffers (#)	nTSVs (#)	Latency (*ps*)	Skew (*ps*)	Buffers (#)	Clk WL ($\times 10^6$)	nTSVs (#)	RT (*s*)	Latency (*ps*)	Skew (*ps*)	Buffers (#)	Clk WL ($\times 10^6$)	nTSVs (#)	RT‡ (*s*)
C1	246.154	37.189	**167**	0	172.027	40.171	**167**	4.768	189	4.351	**77.694**	**29.74**	172	**3.664**	**130**	**0.285**
C2	449.208	334.353	576	0	311.214	269.791	576	14.702	674	7.095	**123.355**	**58.632**	**571**	**11.394**	**499**	**1.693**
C3	214.629	25.953	375	0	159.982	20.929	375	6.019	487	4.470	**90.229**	**20.675**	375	**5.326**	**256**	**1.149**
C4	141.382	25.351	40	0	133.958	24.441	40	0.801	46	3.221	**54.296**	20.231	**33**	**0.623**	**24**	**0.038**
C5	213.993	75.261	79	0	192.928	78.307	79	1.723	89	3.436	**90.829**	**46.735**	**74**	**1.418**	73	**0.096**
Ratio	2.900	2.830	1.010	-	2.223	2.464	1.010	1.249	1.441	6.922	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

Design	Our Buffered Clock Tree*				Our Buffered Clock Tree + [2]				Our Buffered Clock Tree + [7]				Our Buffered Clock Tree + [6]			
	Latency (*ps*)	Skew (*ps*)	Buffers (#)	nTSVs (#)	Latency (*ps*)	Skew (*ps*)	Buffers (#)	nTSVs (#)	Latency (*ps*)	Skew (*ps*)	Buffers (#)	nTSVs (#)	Latency (*ps*)	Skew (*ps*)	Buffers (#)	nTSVs (#)
C1	144.603	41.839	189	0	130.420	54.757	189	243	129.802	54.300	189	167	130.420	55.068	189	200
C2	273.704	70.390	588	0	244.554	117.383	588	739	244.505	117.334	588	551	244.554	117.383	588	576
C3	130.559	29.076	**366**	0	110.956	30.677	**366**	418	113.196	32.917	**366**	385	110.956	30.677	**366**	382
C4	71.089	**17.289**	42	0	65.768	28.930	42	50	65.768	28.930	42	35	65.768	28.930	42	47
C5	127.997	60.538	85	0	110.083	64.472	85	109	110.083	63.693	85	72	110.083	63.693	85	**66**
Ratio	1.714	1.245	1.037	-	1.516	1.683	1.037	1.588	1.520	1.688	1.037	1.232	1.516	1.680	1.037	1.294

† Use OpenROAD to generate buffered clock tree on front-side only. The Clk WL metric is the same as OpenROAD + [2] since the same clock topology is used.
* Use our framework to generate buffered clock tree on front-side only by three steps: conducting hierarchical clock routing, buffer insertion, and skew refinement. The Clk WL metric is the same as Ours since the same clock topology is used.
‡ The runtime for other methods are omitted due to space limit, as their runtime is either similar to Ours or to OpenROAD + [2], according to which algorithm used to generate the buffered clock tree.

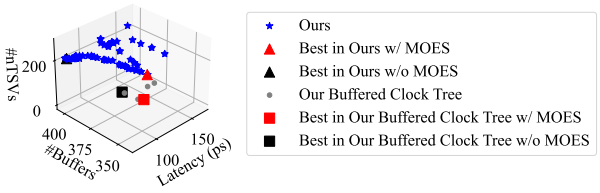


Fig. 10 The effectiveness of MOES with C3 (ethmac) under Ours and Our Buffered Clock Tree.

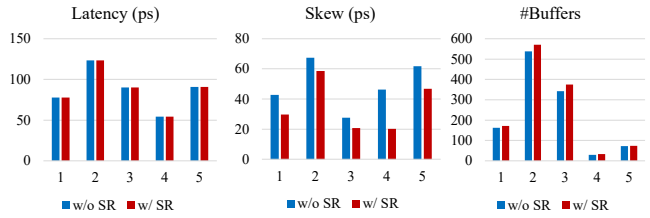


Fig. 11 Effectiveness of skew refinement.

D. Effectiveness of Skew Refinement

In Fig. 11, we demonstrate the effectiveness of skew refinement by the designs in TABLE II. In the skew refinement, we utilize the method in Section III-D to pick up the paths that need skew refinement. From Fig. 11, the skew can be reduced significantly, while the increases in latency and #buffers are very ignorable. This indicates that the method can effectively reduce skew as a post-processing step, which can also provide a better initial solution for the follow-up clock optimization stages.

E. Comparison on Design Space Exploration

To further verify the superiority of our systematic double-side framework, we perform design space exploration on different methods. We set the fanout ranging from 20 to 1000 with step 10 in our DSE flow as introduced in Section III-E. Meanwhile, we set the fanout of [7] also ranging from 20 to 1000 with step 10 and the percentage of critical paths of [6] ranging from 0.2 to 0.9 with step 0.05. To make the comparison fair, the buffered clock tree is all generated by our algorithm.

In Fig. 12, we compare our DSE flow with recent methods [2], [6], [7] in exploration of multi-objective solutions. We can see that the solution space of these methods are restricted to the buffered clock trees, and cannot effectively explore better latency or skew

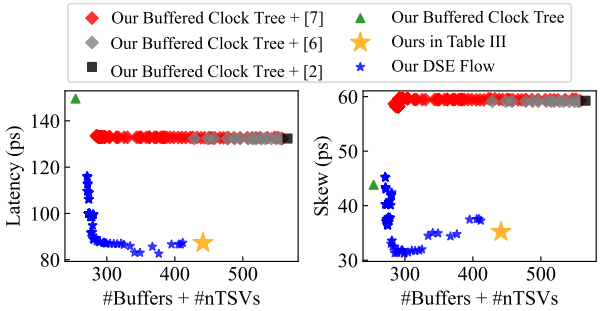


Fig. 12 The comparison of latency and skew of different flows.

even given more nTSVs. A possible reason is the target of buffered clock tree is to minimize clock latency by inserting buffers to shorten wirelength, while the ability of nTSV insertion will be weakened with the wirelength decreasing. Instead, our DSE flow controls the inserting modes in concurrent buffer and nTSV insertion allows much larger solution space. By simply sweeping the fanout threshold, we can find the Pareto frontier trading off latency, skew, buffers, and nTSVs. As our algorithm is very efficient, users can search for suitable solutions with our DSE flow at low cost.

V. Conclusion

In this work, we propose a systematic framework for clock tree synthesis with double-side metal layers and nTSVs. We propose hierarchical clock routing, concurrent buffer and nTSV insertion, and skew refinement to optimize clock tree in multiple objectives. Compared with the recent method [2] with extreme optimization on latency, we can optimize the clock latency by 2.223×, skew by 2.464×, number of buffers by 1.010×, clock wirelength by 1.249×, and number of nTSVs by 1.441×, respectively, with 6.922× speed-up. Meanwhile, our framework offers a DSE flow for multi-objective exploration to further boost the technology and design progressing. In the future, we will investigate placement and routing effects on double-side CTS and develop methodologies for full-flow optimization.

Acknowledgements

The project is supported in part by Grant QYJS-2023-2303-B, the Natural Science Foundation of Beijing, China (Grant No. Z230002), the 111 project (B18001), and Research Grants Council of Hong Kong SAR (No. RFS2425-4S02 and No. CUHK14211824).

References

[1] R. Chen, G. Sisto, A. Jourdain, G. Hiblot, M. Stucchi, N. Kakarla, B. Chehab, S. M. Salahuddin, F. Schleicher, A. Veloso *et al.*, "Design and optimization of sram macro and logic using backside interconnects at 2nm node," in *2021 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2021, pp. 22–4.

[2] A. Veloso, B. Vermeersch, R. Chen, P. Matagne, M. G. Bardon, G. Eneman, K. Serbulova, O. Zografos, S. Chen, G. Sisto *et al.*, "Backside power delivery: Game changer and key enabler of advanced logic scaling and new stco opportunities," in *2023 International Electron Devices Meeting (IEDM)*. IEEE, 2023, pp. 1–4.

[3] D. Prasad, S. T. Nibhanupudi, S. Das, O. Zografos, B. Chehab, S. Sarkar, R. Baert, A. Robinson, A. Gupta, A. Spessot *et al.*, "Buried power rails and back-side power grids: Arm® cpu power delivery network design beyond 5nm," in *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2019, pp. 19–1.

[4] T.-C. Lin, F.-Y. Hsu, W.-K. Mak, and T.-C. Wang, "An effective netlist planning approach for double-sided signal routing," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 288–293.

[5] F.-Y. Hsu, T.-C. Lin, W.-K. Mak, and T.-C. Wang, "A bounding box-based net partitioning method for double-sided routing," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 397–402.

[6] N. E. Bethur, P. Vanna-Iampikul, O. Zografos, L. Zhu, G. Sisto, D. Milojevic, A. García-Ortiz, G. Hellings, J. Ryckaert, F. Catthoor *et al.*, "Gnn-assisted back-side clock routing methodology for advance technologies," *Proceedings of the 61st Design Automation Conference*, 2024.

[7] N. E. Bethur, "A methodology for back-side clock delivery network design compatible with commercial eda flows," Master's thesis, Georgia Institute of Technology, 2023.

[8] M. M. S. Aly, T. F. Wu, A. Bartolo, Y. H. Malviya, W. Hwang, G. Hills, I. Markov, M. Wootters, M. M. Shulaker, H.-S. P. Wong *et al.*, "The n3xt approach to energy-efficient abundant-data computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 19–48, 2018.

[9] H. Lu, Y. Ge, X. Jiang, J. Sun, W. Peng, R. Guo, M. Li, Y. Lin, R. Wang, H. Wu *et al.*, "First experimental demonstration of self-aligned flip fet (ffet): A breakthrough stacked transistor technology with 2.5 t design, dual-side active and interconnects," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[10] M. Shamanna, E. Abuayob, G. Aenuganti, C. Alvares, J. Antony, A. Bahudhanam, A. Chandran, P. Chew, A. Chatterjee, B. Chauhan *et al.*, "E-core implementation in intel 4 with powervia (backside power) technology," in *2023 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2023, pp. 1–2.

[11] "Tsmc a16 technology," https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_A16.

[12] W. Wang, V. F. Pavlidis, and Y. Cheng, "Zero-skew clock network synthesis for monolithic 3d ics with minimum wirelength," in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 399–404.

[13] K. D. Boese and A. B. Kahng, "Zero-skew clock routing trees with minimum wirelength," in *[1992] Proceedings. Fifth Annual IEEE International ASIC Conference and Exhibit*. IEEE, 1992, pp. 17–21.

[14] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *Proceedings of the 30th international Design Automation Conference*, 1993, pp. 612–616.

[15] W. Li, Z. Huang, B. Yu, W. Zhu, and X. Li, "Toward controllable hierarchical clock tree synthesis with skew-latency-load tree," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[16] L. P. Van Ginneken, "Buffer placement in distributed rc-tree networks for minimal elmore delay," in *1990 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 1990, pp. 865–868.

[17] K. Han, A. B. Kahng, and J. Li, "Optimal generalized h-tree topology and buffering for high-performance and low-power clock distribution," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 2, pp. 478–491, 2018.

[18] M. R. Guthaus, G. Wilke, and R. Reis, "Non-uniform clock mesh optimization with linear programming buffer insertion," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 74–79.

[19] Y.-Y. Chen, C. Dong, and D. Chen, "Clock tree synthesis under aggressive buffer insertion," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 86–89.

[20] J. G. Xi and W. W.-M. Dai, "Useful-skew clock routing with gate sizing for low power design," in *Proceedings of the 33rd annual Design Automation Conference*, 1996, pp. 383–388.

[21] W. Shen, Y. Cai, W. Chen, Y. Lu, Q. Zhou, and J. Hu, "Useful clock skew optimization under a multi-corner multi-mode design framework," in *2010 11th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2010, pp. 62–68.

[22] N. Uysal, W.-H. Liu, and R. Ewetz, "Latency constraint guided buffer sizing and layer assignment for clock trees with useful skew," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, 2019, pp. 761–766.

[23] C. Deng, Y.-C. Cai, and Q. Zhou, "Register clustering methodology for low power clock tree synthesis," *Journal of Computer Science and Technology*, vol. 30, no. 2, pp. 391–403, 2015.

[24] A. D. Mehta, Y.-P. Chen, N. Menezes, D. Wong, and L. Pilegg, "Clustering and load balancing for buffered clock tree synthesis," in *Proceedings International Conference on Computer Design VLSI in Computers and Processors*. IEEE, 1997, pp. 217–223.

[25] S. Bang, K. Han, A. B. Kahng, and V. Srinivas, "Clock clustering and io optimization for 3d integration," in *2015 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*. IEEE, 2015, pp. 1–8.

[26] T.-Y. Kim and T. Kim, "Clock tree synthesis for tsv-based 3d ic designs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 16, no. 4, pp. 1–21, 2011.

[27] J.-S. Yang, J. Pak, X. Zhao, S. K. Lim, and D. Z. Pan, "Robust clock tree synthesis with timing yield optimization for 3d-ics," in *16th Asia and South Pacific Design Automation Conference (ASP-DAC 2011)*. IEEE, 2011, pp. 621–626.

[28] S. Pentapati, J. Lee, Y. S. Yu, S. K. Lim *et al.*, "Tier partitioning and flip-flop relocation methods for clock trees in monolithic 3d ics," in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.

[29] P. Vanna-iampikul, H. Yang, J. Kwak, J. X. Hu, A. Rahman, N. E. Bethur, C. Hao, S. Yu, and S. K. Lim, "Back-side design methodology for power delivery network and clock routing," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2024, pp. 1–2.

[30] J. Hu, G. Schaeffer, and V. Garg, "Tau 2015 contest on incremental timing analysis," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 882–889.

[31] T. Marler and J. S. Arora, "Multi objective optimization: concepts and methods for engineering," 2009.

[32] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[33] "Openroad," https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts.

[34] C. Sitik, W. Liu, B. Taskin, and E. Salman, "Design methodology for voltage-scaled clock distribution networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 10, pp. 3080–3093, 2016.

[35] "Cadence Innovus Implementation System," http://www.cadence.com.

[36] R. Ewetz, "A clock tree optimization framework with predictable timing quality," in *Proc. DAC*, 2017, pp. 1–6.

[37] J. Chen, I. H.-R. Jiang, J. Jung, A. B. Kahng, V. N. Kravets, Y.-L. Li, S.-T. Lin, and M. Woo, "Datc rdf-2019: Towards a complete academic reference design flow," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–6.

ArbiterQ: Improving QNN Convergency and Accuracy by Applying Personalized Model on Heterogeneous Quantum Devices

Tianyao Chu[†]
Zhejiang University
Hangzhou, China
tianyao_chu@zju.edu.cn

Siwei Tan[†]
Zhejiang University
Hangzhou, China
siweitan@zju.edu.cn

Liqiang Lu[*]
Zhejiang University
Hangzhou, China
liqianglu@zju.edu.cn

Jingwen Leng
Shanghai Jiao Tong University
Shanghai, China
leng-jw@sjtu.edu.cn

Fangxin Liu
Shanghai Jiao Tong University
Shanghai, China
liufangxin@sjtu.edu.cn

Congliang Lang
Zhejiang University
Hangzhou, China
langcongliang@zju.edu.cn

Yifan Guo
Zhejiang University
Hangzhou, China
guo2001yifan@zju.edu.cn

Jianwei Yin[*]
Zhejiang University
Hangzhou, China
zjuyjw@cs.zju.edu.cn

***Abstract*—In the current NISQ era, the performance of QNN models is strictly hindered by the limited qubit number and inevitable noise. A natural idea to improve the robustness of QNN is to involve multiple quantum devices. Nevertheless, due to the heterogeneity and instability of quantum devices (e.g., noise, frequent online/offline), training and inference on distributed quantum devices may even destroy the accuracy.**

In this paper, we propose ArbiterQ, a comprehensive QNN framework designed for efficient and high-accuracy training and inference on heterogeneous QPUs. The main innovation of ArbiterQ is it applies personalized models for each QPU via two uniform QNN representations: *model vector* and *behavioral vector*. The model vector specifies the logical-level parameters in the QNN model, while the behavioral vector captures the hardware-level features when implementing the QNN circuit. In this manner, by sharing the gradient among QPUs with similar behavioral vectors, we can effectively leverage parallelism while considering heterogeneity. We also propose shot-oriented inference scheduling, which is a much more fine-grained scheduling that can improve accuracy and balance the workload. The experiments show that ArbiterQ accelerates the training process by 4.03× with 7.87% loss reduction, compared with the previous distributed QNN framework EQC [1].

I. Introduction

Quantum neural network (QNN) has emerged as a popular quantum application that demonstrates practical utility in various tasks, including face recognition [2] and breast cancer prediction [3]. To be more specific, QNN is a new prototype of the neural network to handle machine learning tasks, where the classical feature is encoded into quantum states and subsequently evolved by quantum gates. A QNN model comprises multiple quantum circuit blocks that involve quantum superposition and entanglement to learn the features [4]. Since quantum computing exhibits the exponential expressiveness of quantum states and the huge potential parallelism [5], QNN is a promising model for the next generation of machine learning.

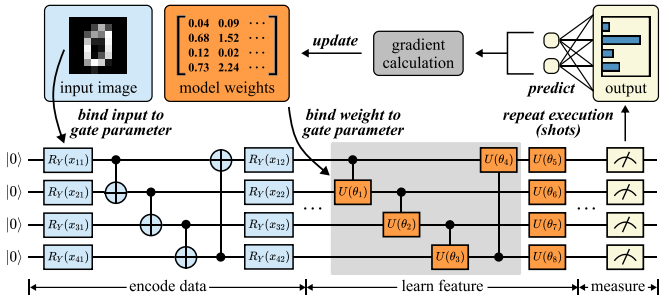
However, the scale of the current quantum processing unit (QPU) hinders the performance of QNN models, which arises from the limited number of qubits, rendering it impossible to deploy a large-scale QNN. For example, the inference of a 4-learning-layer QNN model takes around 0.26 seconds = ($50\mu s$ for circuit + $2\mu s$ for readout + $200\mu s$ for delay) × 1000 executions on the IBM *ibm_osaka* QPU. When training a single instance with parameter shift [6], it takes around 10 seconds = 0.26s × 40 parameters. A natural idea is to employ multiple quantum devices to facilitate the training and inference of the QNN model. However, the traditional methodology of parallel computing is hard to apply to quantum scenarios due to the huge theoretical gap, imperfect quantum devices, and vulnerable quantum state. Here, we summarize the main challenges to implementing QNN on heterogeneous devices:

- **Training challenge.** Even for the same QNN model, its circuit can be different after compilation due to the constraint of basic gates and topology [7]–[10]. On the other hand, quantum devices inevitably encounter various noises, leading to parameter shifting in diverse directions [11]. Therefore, the distinct behavior of the QNN circuit makes it difficult to construct a globally optimal QNN model on a heterogeneous quantum system.
- **Inference challenge.** Conventional distributed deep learning usually employs batch-level parallelism to speed up the training. However, this is unsuitable for quantum computing due to the its fundamentally different operational paradigm. Heterogeneous quantum hardware tends to have diverse preferences on tasks, leading to significant reductions in accuracy during inference compared to training.

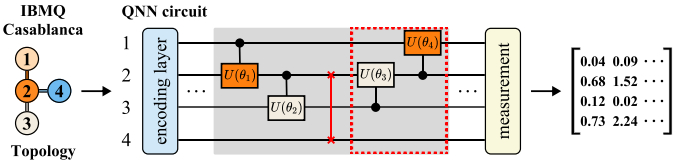
Prior works based on traditional parallelization are grounded in the preciseness of computation, which becomes ineffective when gathering the results from noisy quantum devices. For example, EQC [1] proposes ensemble learning for distributed training by abstracting the noise intensity as a voting weight to eliminate the high-degree noise. However, EQC applies unified model weights for distributed training, lacking the consideration of heterogeneity. Such unified weights can be suboptimal in each individual QPU, thereby resulting in the reduction of accuracy and massive epochs for convergence.

In this paper, we present ArbiterQ, a comprehensive QNN

[†]These authors contribute equally to this work. [*]Corresponding authors.



(a) Structure and operation of QNN.



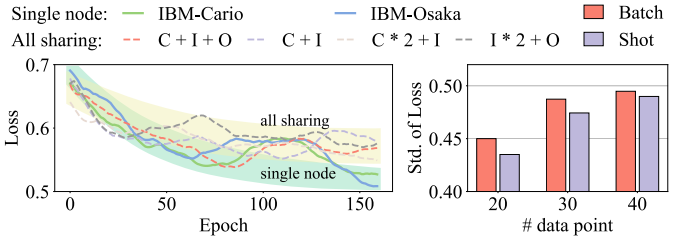
(b) QNN deployed on QPU.

Fig. 1: QNN model and its implementation on QPUs.

framework that improves the performance for training and inference on heterogeneous quantum devices. We first decompose the QNN circuit into *contextual vector* and *topological vector*. The contextual vector records the structural information of each layer in the model, where the element is defined as the cumulative error of a series of basic gates. While the topological vector is derived from the SWAP gates that originate from the topology constraint. These two vectors are further concatenated into a behavioral vector, which expressively describes the circuit behavior after implementing a specific QNN model.

To train on heterogeneous system, we employ a personalized QNN model in each QPU and design the similarity-aware gradient sharing scheme. Clearly, we only unify the network architecture and allow each device to house local model weights. By calculating the distance between the behavioral vectors of different devices, we propose to partially share the gradient among QPUs with similarity. This training approach enables efficient collaboration among variational quantum devices, substantially improving the accuracy and convergence speed. *To enable large-scale inference,* we propose to assign the tasks in the granularity of shot level since the inference results of QNN models rely on multiple shots to obtain the probability. Given the personalized model may exhibit different preferences for tasks, we divide the QPUs into multiple tori and adaptively allocate tasks to each torus. To be specific, we compensate for the noise in each torus, achieved by identifying the cycle period in the behavioral vectors using a discrete Fourier transform. In conclusion, this work makes the following contributions:

- We propose behavioral vectorization for modeling the QNN implementation on heterogeneous devices. By representing the contextual and topological features uniformly, ArbiterQ enables the distributed training with high accuracy and convergence speed.
- We design shot-oriented inference that distributes the tasks to a series of QPU tori. By compensating the noise in each torus, ArbiterQ shrinks the accuracy loss and balances the workload.
- We implement ArbiterQ with both quantum simulator and real quantum devices. Experiments show that ArbiterQ outperforms EQC [1] with 4.03× convergence speed and 7.87% accuracy loss. It also reduces the inference loss by 24.71% compared with batch-based scheduling.



(a) Poor convergence under all-sharing training. (b) Unfair batch-based inference.

Fig. 2: Motivational example of straightforward distributed QNN training (a 2-layer QNN model [12] trained with Wine dataset [13]).

II. Background

A. Quantum Neural Networks

Quantum neural network (QNN) is a new prototype of the neural network to handle machine learning tasks with classical data. QNN extracts the feature of data and outputs normalized results. Based on the rich data encoding structure in quantum states, QNN is a promising way to show the quantum advantage in practical tasks, such as face recognition [2] and breast cancer prediction [3], etc.

Figure 1 (a) depicts a typical QNN model that consists of three functional layers, namely the encoding layer, the learning layer, and the measurement layer. For the image dataset, the pixels are encoded into the parameters of a set of uniformly rotated gates, such as R_Y here. The learning layer comprises several repeated quantum circuit blocks that serve as the backbone of the QNN model, where the gate parameters can be regarded as trainable weights. The measurement layer outputs the probability distribution of quantum states by repeatedly executing and measuring the circuit multiple times. In this paper, we name every single measurement as one "shot".

B. Quantum Processing Unit

Quantum device is the fundamental hardware that employs quantum gates for conducting unitary operators on qubits. In this paper, we focus on the implementation of superconducting quantum devices and name quantum processing units (QPU). The real-world QPUs usually involve different physical configurations, including the number of qubits, decoherence time, topology, and fidelity. These features lead to diverse performance when deploying the same QNN model on heterogeneous QPUs, as shown in Figure 1 (b).

First, quantum devices inevitably suffer from various noises, leading to the errors of quantum gates. These errors can be different at spatial and temporal level due to environmental interaction [14], hardware defect [15], and diverse crosstalk [16]. Second, the topological constraint requires inserting additional SWAP gates to enable double-qubit gates. Therefore, the

executed circuit can be different under heterogeneous QPUs. For example, Figure 1 (b) shows an inserted SWAP gate and relocated following gates in the executable quantum circuits for IBMQ Casablanca QPU. In a word, the circuit design of the QNN model exhibits diversity on heterogeneous quantum devices, resulting in different outputs and different optimal model weights.

C. Motivational Examples

As aforementioned, training a QNN model on multiple QPUs does not necessarily accelerate the training process and improve the convergence, which may even hurt the final accuracy due to the variation of heterogeneous devices. To illustrate this, Figure 2 provides a motivational example that implements a QNN model using three superconducting quantum chips: IBM Cario (C), IBM Osaka (O), and IBM Ithaca (I). In Figure 2 (a), we apply an all-sharing training approach that equally combines the gradients from distributed QPUs and keeps the weights the same across all devices.

We can find that, as the number of epochs increases, the loss curve of these two methods gradually diverges, with the loss of distributed training being much higher than the single-node training. Such discrepancy arises from the heterogeneity of QPUs, leading to sub-optimal gate parameters in QNN circuits. This motivates us to quantify the gate error on the QNN model and orchestrate a gradient-sharing scheme to counteract the noise effect.

Even though we have trained the QNN model with customized weights for each heterogeneous QPU, it is still challenging to enable efficient and stable inference on QPUs with spatial and temporal heterogeneity. Figure 2 (b) shows the standard deviation of loss value using batch-based inference on heterogeneous QPUs. Considering that QNN relies on multiple shots to get the results, we propose a much more fine-grained scheduling called shot-based inference. The shot-based inference has less differential accuracy, which comes from that batch-based scheduling infers each data point with a single QPU only, while shot-based scheduling adaptively divides shots and assigns tasks to smooth the QPU difference.

III. Similarity-aware Personalized Training

The key characteristic of the QNN model is that its optimal weight parameters are closely related to the device properties, such as noise and topology. Therefore, the inherent heterogeneity of quantum devices makes it difficult to leverage the parallelism of distributed computing. To tackle this, we apply a personalized QNN model for each individual QPU. As shown in Figure 3, the QPU keeps a unified model structure while having different weight parameters in the quantum gates. In this manner, the model can better fit heterogeneous hardware properties, leading to higher accuracy and robustness. We collect the weight parameters into a *model vector*. To quantify the heterogeneity of various QPUs and guide the co-optimization of every personalized model, we propose to vectorize the behaviors of implementing the QNN model on a specific QPU, namely *behavioral vector*, with the goal of capturing both

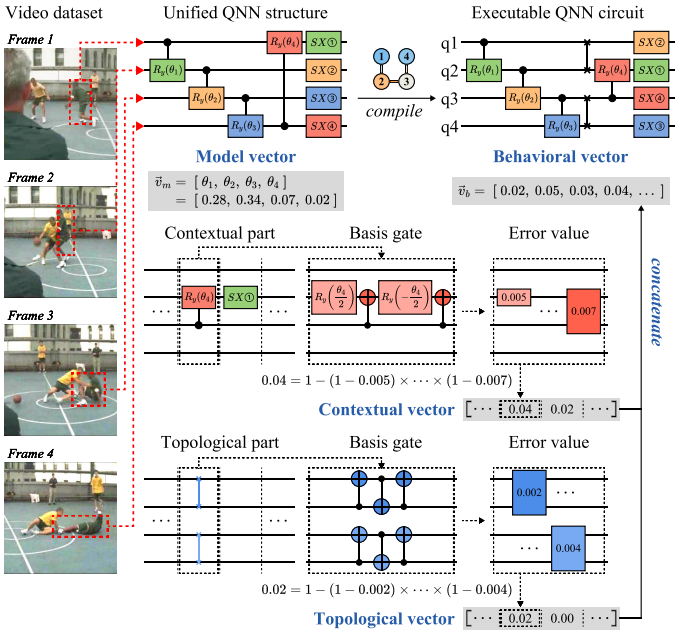


Fig. 3: Vectoring personalized QNN model and its compiled circuit.

contextual information and topological constraints. Then we introduce a similarity-aware gradient-sharing scheme, where the QPUs that show similarity in the behavioral vector are updated together.

A. Vectorizing the QNN Behavior on QPUs

We start by vectorizing the executable circuit after compiling the QNN circuit under the constraints of a specific QPU. During vectorization, elements are arranged by the execution sequence in the QNN circuit, from left-top to right-bottom. Considering that several parallel gates may exchange after compilation, we unify the order following the sequence of the original QNN circuit. For example, we label the four parallel SX gates in the original QNN model from top to bottom. Due to the SWAP gate between qubit 1 and qubit 2, the gate SX① is routed to qubit 2. As a result, we put this gate in front of SX② to ensure the consistency of vectorization.

We decouple the executable circuit into a contextual part and a topological part, as shown in Figure 3. In the contextual vector, each element is derived from the original gate in the QNN model, which aims to record the structural information of each layer in the model. As mentioned before, the topological constraints necessitate SWAP gates to establish entanglement between two unconnected qubits. Thus, the topological part refers to these extra SWAP gates, also with each gate corresponding to one vector element.

The optimal weights mainly depend on the infidelity of quantum gates. Therefore, we formulate each element value as the cumulative error of a series of basic gates: $\vec{v}_c(i) = 1 - \prod_j (1 - e_{ij})$, where $\vec{v}_c(i)$ is the i-th element in the contextual vector, and e_{ij} represents the error of the j-th basis gate decomposed from the i-th gate. We apply a simple formula [17] to calculate the executional error of quantum gate $e = 1 - \mathrm{e}^{-\frac{t}{\tau} f}$, which involves the execution time (t),

depolarization time (τ, for single-qubit gate) or decoherence time (τ, for double-qubit gates), and fidelity (f). Figure 3 gives an example of calculating the element value for $R_y(\theta_4)$ gate, which is decomposed into four basic gates.

For the topological part, its vector length is aligned with the contextual vector. Specifically, the SWAP gate is generated from the two-qubit gate that comprises two nonadjacent qubits, which means the error from SWAP gates will lead to parameter deviation in its corresponding two-qubit gate. Taking Figure 3 as an example, two SWAP gates are inserted for implementing $R_y(\theta_4)$ gate, contributing to the fourth element in the topological vector. Similarly, we can calculate the element value for these two SWAP gates that imply the connectivity of QPU topology. Finally, we concatenate the contextual vector and topological vector, which fully extracts the behavior of deploying the QNN model on QPUs.

B. Similarity-aware Gradient Sharing

The training of personalized QNN models is conducted in the space formed by all behavioral vectors. Our key idea is to group the QPUs that exhibit similar behavioral vectors, which helps to leverage the parallelism of distributed devices and minimize the overhead of heterogeneity. To identify such similarity, we formulate the distance between QPU i and QPU j as follows,

$$dist(i,j) = \frac{\|\vec{v_b^i} - \vec{v_b^j}\|_2}{length(\vec{v_b})} \tag{1}$$

where $length(\vec{v_b})$ denotes the length of the behavioral vector. Then, we can obtain the grouping scheme by setting a distance threshold. Furthermore, we quantify the degree of similarity between every two QPU nodes to ensure that the devices with the highest similarity collaborate best. The similarity is defined as $sim(i,j) = e^{-\kappa \cdot dist(i,j)}$, where κ is a hyperparameter that determines the scope of similarity.

The QPUs that exhibit similarity tend to prefer the same optimal weights. Therefore, we only allow gradient sharing within the group. Concretely, in each iteration, the QPU node executes its local QNN model and calculates the loss by measurement. Given the loss function, we apply parameter shifting [18] to get the local gradient of weights, which will be sent to the QPU that shares the similarity. Accordingly, each QPU also buffers gradients from neighbor nodes and uses them to update the local QNN model. More importantly, the shared gradient is multiplied by the similarity degree to fully utilize the parallelism of near-homogeneous devices. This gradient sharing speeds up the training of each node.

IV. Shot-oriented Inference on QPU Torus

Paralleling large-scale inference tasks is necessitated by the demand for low latency and high throughput. As analyzed in Section II-C, shot-based parallelism, i.e. inferring each task with multiple devices, provides a much more fine-grained opportunity to schedule the inference tasks. With this insight, we propose the shot-oriented distributed inference that flexibly allocates the tasks at the shot level, achieving high accuracy and throughput of the overall system.

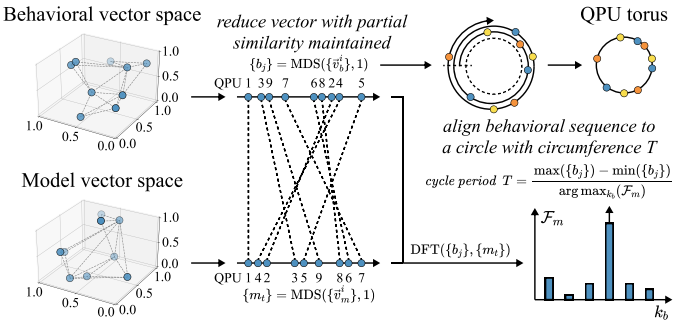


Fig. 4: Mapping QPUs onto torus using DFT.

A. Noise Compensation via QPU Torus

To enable large-scale inference with high accuracy, our goal is to minimize the heterogeneity (inference bias) among personalized QNN models. Clearly, the QPUs with adjacent behavioral vectors may introduce inference bias due to similar noise patterns, resulting in accuracy loss. Therefore, a natural idea to alleviate this is the compensation of various noises, which requires scattering the QPUs with minimum bias. However, the behavioral vector is high-dimensional, giving rise to huge time complexity to arrange QPUs with low behavioral similarity.

Therefore, we first use MDS algorithm [19] to reduce the dimension of the behavioral vector space and the model vector space, as shown in Figure 4. Using MDS algorithm, the nodes in these two spaces are arranged in two one-dimensional sequences meanwhile keeping the same relative distance as before. In other words, two adjacent nodes in the behavioral vector space are also close to each other after MDS. However, the MDS algorithm removes several connection information, which may cause two distant elements in the sequence to share similarities. To handle this, we apply the model sequence to classify such "distant similarity", which involves a non-uniform discrete Fourier transform (DFT) to obtain the initial torus to string the QPU nodes together. Mathematically, considering the model sequence as a signal sequence varying with the behavioral nodes, the sequence of model nodes is transformed as follows,

$$\mathcal{F}_m[k_b] = \sum_j m_t \cdot e^{-i\frac{2\pi}{\max(\{b_j\}) - \min(\{b_j\})} k_b b_j} \tag{2}$$

where $\{b_j\}$ and $\{m_t\}$ is the behavioral sequence and model sequence, respectively. The element m_t in the model sequence points to the same QPU of b_j. For example, in Figure 4, m_2 and b_8 point to the same QPU.

Next, we select the maximum frequency to calculate the cycle period in the behavioral sequence,

$$T = \frac{\max(\{b_j\}) - \min(\{b_j\})}{\arg\max_{k_b}(\mathcal{F}_m)} \tag{3}$$

where k_b is the frequency index of each model weight component after DFT $\mathcal{F}_m$. The period T implies the maximum distance among the nodes in the behavioral vector space. Thus, we construct the initial torus by aligning the behavioral sequence to a circle with a circumference of T, as shown

TABLE I: Training results on different tasks and QNN models.

Benchmark		Convergence epoch					Convergence loss				
		Single-node	All-sharing	EQC [1]	ArbiterQ	Speedup	Single-node	All-sharing	EQC [1]	ArbiterQ	Reduction
Iris [20]	**Model-CRz**	24	38	28	3	9.33×	0.0881	0.0905	0.0737	0.0566	23.23%
	Model-CRx	29	33	20	3	6.67×	0.1136	0.1057	0.0741	0.0608	17.90%
Wine [13]	**Model-CRz**	47	43	37	30	1.23×	0.4717	0.4686	0.4691	0.4550	3.02%
	Model-CRx	54	40	37	9	4.11×	0.4711	0.4739	0.4710	0.4550	3.40%
MNIST [21]	**Model-CRz**	185	198	144	87	1.66×	0.3213	0.3534	0.2858	0.2824	1.19%
	Model-CRx	159	67	98	28	3.50×	0.2910	0.2734	0.2736	0.2626	4.03%
HMDB51 [22]	**Model-CRz**	168	158	178	82	2.17×	0.3116	0.3218	0.3177	0.3102	2.35%
Average		-	-	-	-	4.03×	-	-	-	-	7.87%

The convergence loss is calculated by averaging the loss of the test dataset across all QPUs, without any inference scheduling strategy.

TABLE II: Benchmarks

Dataset	# sample	# feature	# qubits	# QNN weights
Iris [20]	100	4	2	8
Wine [13]	114	13	4	16
MNIST [21]	100	64	6	24
HMDB51 [22]	100	108	10	200

TABLE III: Configuration of simulators

QPU	①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩
1q-gate infidelity ($\times 10^{-4}$)	2.36	3.06	1.45	5.07	3.41	2.29	4.27	1.72	3.66	2.42
2q-gate infidelity ($\times 10^{-3}$)	7.58	8.67	4.81	4.33	3.69	2.93	4.62	3.66	2.90	9.75
T_1(us)	193	137	349	134	114	103	171	232	260	166
T_2(us)	21.4	67.1	84.7	89.2	96.5	25.7	83.2	47.9	58.4	38.6

in Figure 4. Finally, we divide it into multiple smaller sub-tori by equidistant partition and sort these tori according to their average accuracy. Since MDS algorithm maintains the similarity in the sequence of QPU nodes, the QPUs involved in each sub-torus exhibit low similarity, being able to compensate for the noise with each.

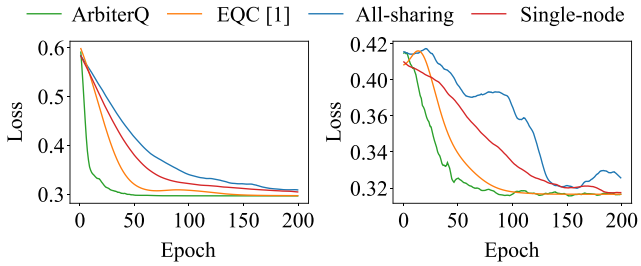
B. Shot-oriented Scheduling

With the available QPUs divided into multiple tori, we start assigning tasks via a warm-up stage to preliminarily sketch the difficulty of target tasks, i.e., the loss of each task. Then, we adopt a greedy-based scheduling that allocates "hard" tasks to the QPU torus with higher accuracy. In each torus, we assign the shots of each task to all QPUs proportionally according to the throughput. Besides, the inference tasks are also proportionally allocated according to the throughput of each QPU torus, resulting in fully balanced workload scheduling.

V. Evaluation

A. Experimental Setup

Datasets. Table II shows the benchmarks used in this paper. We choose two typical pattern recognition datasets, including Iris [20] and Wine [13]. We also conduct the evaluation on an image classification dataset MNIST [21] and a video classification dataset HMDB51 [22] that consists of 6,766 realistic video clips from 51 action categories, such as jump, kiss, and laugh. Similar to the prior methodology, we focus on the binary classification task by selecting two types of objects



(a) MNIST [21]. (b) HMDB51 [22].

Fig. 5: Convergence across different benchmarks.

from these datasets. The proportion of the training dataset and inference dataset is 80% and 20%, respectively.

QNN models. We adopt the angle encoding method [23] to transform classical input to the input quantum state. We apply two types of backbones used in [12], resulting in two QNN models, namely *Model-CRz* and *Model-CRx* (circuit 5 and 14 in Figure 2 of [12]). The hyperparameter κ defined in the gradient sharing is set to 20000.

Platforms. We evaluate ArbiterQ on the real-world quantum platform of ORIGIN QUANTUM [24] accessible via `PyQPanda` (3.8.3.4) [25]. It provides a 72-qubit superconducting quantum chip with a 6×12 grid qubit topology, supporting U3 and CZ gates. The average 1-qubit gate fidelity and 2-qubit gate fidelity are 99.72% and 95.86% respectively. To control the hardware specifications for a fair comparison, we also evaluate ArbiterQ on 10 simulators of quantum computers with 2 to 10 qubits, which are configured as Table III.

Comparison. ArbiterQ is compared with three QNN implementations, including 1) *EQC* [1]: which applies ensembled quantum computing that sets different voting weights according to the noise configuration; 2) *all-sharing*: which updates the gradient by calculating the average value across all QPUs; 3) *single-node*: which trains the QNN model on a single quantum device.

B. Training Performance

Table I compares the training performance associated with the convergence epoch and loss on different benchmarks.

Model convergence. Compared to EQC, all-sharing and single-node, the ArbiterQ accelerates the convergence with an average of 4.03×, 5.19×, 5.04× speedup, respectively.

TABLE IV: Configuration and results of shot-oriented inference

QPU			① ② ③ ④ ⑤ ⑥	① ② ③ ④ ⑤ ⑥ ⑦ ⑧	① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩
Iris [20]	ArbiterQ	Cycle T	0.0119	0.0177	0.0060
		Torus	{1, 3, 5} {2, 4, 6}	{4, 3, 8, 5} {2, 1, 6, 7}	{8, 6, 2, 1} {9, 7, 4} {10, 5, 3}
		Loss	0.0385	0.0354	0.0386
	EQC [1]	Loss	0.0745	0.0650	0.0741
Wine [13]	ArbiterQ	Cycle T	0.0247	0.0267	0.0177
		Torus	{3, 5, 6} {2, 4, 1}	{5, 1, 7, 3} {8, 2, 4, 6}	{2, 7, 4, 3} {1, 6, 10} {9, 8, 5}
		Loss	0.4985	0.4819	0.4626
	EQC [1]	Loss	0.5068	0.4971	0.4710

By sharing the gradients with similar behavioral vectors, parallelism is utilized to learn the image features with fewer iterations. Furthermore, the convergence curve depicted in Figure 5 demonstrates that our approach is more stable compared to other methods. Because our similarity-aware training prevents communication among the QPUs with distinct structural features, eliminating the negative weight sharing. Interestingly, the all-sharing distributed solution shows the worst convergence, indicating that the heterogeneity of quantum devices may even overwhelm the advantages of parallel computing.

Accuracy improvement. ArbiterQ shows the best accuracy loss compared to the other methods. This stems from that we employ the personalized QNN model and locally update weights that fit the device properties. ArbiterQ exhibits the best improvement on the Iris dataset with 17.9% - 23.2% loss reduction. This is because the number of weights for the Iris dataset is quite small, making it easier for ArbiterQ to search the global optimal weights. We observe that the accuracy of EQC is slightly lower than single-node, which comes from that the central model in EQC is limited by the sub-optimal model weights.

C. Inference Performance

Table IV summarizes the accuracy loss on the test dataset using Iris [20] and Wine [13] benchmark. In this table, we provide the detailed configuration of shot-oriented scheduling, including the cycle period after discrete Fourier Transformation and the torus after equidistant partition. Overall, ArbiterQ achieves a 24.71% reduction of loss on these two benchmarks, where we assume that EQC adopts batch-based inference. This suggests that our designed scheduling strategies can significantly enhance the inference accuracy by adaptively assigning the tasks. Moreover, ArbiterQ has lower accuracy loss with a larger number of QPUs as it involves more tori with diverse preferences, which helps to avoid local optimal scheduling.

D. Evaluation on Real-world QPU

We design a 2-qubit QNN model with U3 and CZ basic gates of *origin_wukong* QPU [24]. In particular, we select 4 groups of qubits on this chip and form them as a distributed system. Due to the limited access to the quantum cloud, we only assign partial training data on real QPU. Overall, the results in Figure 6 on the real QPU match the trend in Table I, demonstrating the validness of our design. The final loss of ArbiterQ, EQC, all-sharing, single-node is 0.1045, 0.1092, 0.1397, and 0.1383, respectively. In terms of the convergence speed, ArbiterQ is 1.57×, 1.62×, and 1.64× faster than EQC, all-sharing, and single-node, respectively.

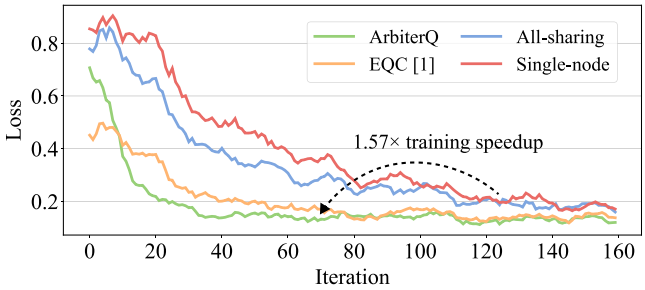


Fig. 6: Model convergence on the real-world QPU.

VI. Related Work

Quantum neural network. To enlarge the scale of the problem that the QNN can solve, [26] introduced a quantum-classical hybrid method to image classification. Additionally, [27] researches the effect of different quantum-classical workload splitting techniques during the distributed training QNN model, including circuit parallelization and data parallelization. There are also works that study the multi-device job scheduling for variational quantum algorithms including QNN [28].

Quantum noise mitigation. Researchers have proposed various methods to mitigate noise in QNN. [29] applies the classical error mitigation method to real-world QNN execution. The first effort to enable training on heterogeneous QPUs is EQC [1], which adopts a weighted parameter updating according to the estimated errors of the devices. There are also general optimization methods that can optimize circuit noises, such as QuCT [30] proposed to vectorize the circuit gate by gate for fidelity optimization and unitary decomposition.

VII. Conclusion

We present ArbiterQ, the first framework for full-process deployment of the QNN model on heterogeneous QPUs. Personalized QNN models are tailored to each QPU and the gradient-sharing strategy is facilitated to tackle the architectural heterogeneity of quantum devices. The shot-oriented inference scheduling enhances the accuracy and workload balance effectively. Overall, ArbiterQ represents a significant advancement toward the practical usage of QNNs on real-world quantum computers.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No.2023YFF0905200), the Zhejiang Provincial Natural Science Foundation of China under Grant (No.LR25F020002), and the National Natural Science Foundation of China (No.623B1001).

REFERENCES

[1] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, "Eqc: Ensembled quantum computing for variational quantum algorithms," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 59–71.

[2] Y. Xu, X. Zhang, and H. Gai, "Quantum neural networks for face recognition classifier," *Procedia Engineering*, vol. 15, pp. 1319–1323, 2011.

[3] P. Li and H. Xiao, "Model and algorithm of quantum-inspired neural network with sequence input based on controlled rotation gates," *Applied Intelligence*, vol. 40, p. 107–126, 2014.

[4] A. Ranjan, T. Patel, D. Silver, H. Gandhi, and D. Tiwari, "Proximl: Building machine learning classifiers for photonic quantum computing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 834–849.

[5] Y. Wu, J. Yao, P. Zhang, and H. Zhai, "Expressivity of quantum neural networks," *Phys. Rev. Res.*, vol. 3, p. L032049, Aug 2021.

[6] D. Wierichs, J. Izaac, C. Wang, and C. Y.-Y. Lin, "General parameter-shift rules for quantum gradients," *Quantum*, vol. 6, p. 677, 2022.

[7] A. Molavi, A. Xu, M. Diges, L. Pick, S. Tannu, and A. Albarghouthi, "Qubit mapping and routing via maxsat," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1078–1091.

[8] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014.

[9] A. Xu, A. Molavi, L. Pick, S. Tannu, and A. Albarghouthi, "Synthesizing quantum-circuit optimizers," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 835–859, 2023.

[10] T. Tomesh, P. Gokhale, V. Omole, G. S. Ravi, K. N. Smith, J. Viszlai, X.-C. Wu, N. Hardavellas, M. R. Martonosi, and F. T. Chong, "Supermarq: A scalable quantum benchmark suite," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 587–603.

[11] T. Li, L. Lu, Z. Zhao, Z. Tan, S. Tan, and J. Yin, "Qust: Optimizing quantum neural network against spatial and temporal noise biases," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 4, pp. 1434–1447, 2025.

[12] S. Sim, P. D. Johnson, and A. Aspuru-Guzik, "Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms," *Advanced Quantum Technologies*, vol. 2, no. 12, p. 1900070, 2019.

[13] S. Aeberhard, D. Coomans, and O. Y. de Vel, "Comparative analysis of statistical pattern recognition methods in high dimensional settings," *Pattern Recognit.*, vol. 27, pp. 1065–1077, 1994.

[14] D. Suter and G. A. Álvarez, "Colloquium: Protecting quantum information against environmental noise," *Rev. Mod. Phys.*, vol. 88, p. 041001, Oct 2016.

[15] B. Gardas, J. Dziarmaga, W. H. Zurek, and M. Zwolak, "Defects in quantum computers," *Sci. Rep.*, vol. 8, no. 4539, 2018.

[16] P. Zhao, K. Linghu, Z. Li, P. Xu, R. Wang, G. Xue, Y. Jin, and H. Yu, "Quantum crosstalk analysis for simultaneous gate operations on superconducting qubits," *PRX Quantum*, vol. 3, p. 020301, Apr 2022.

[17] Y. R. Sanders, J. J. Wallman, and B. C. Sanders, "Bounding quantum gate error rate based on reported average fidelity," *New Journal of Physics*, vol. 18, no. 1, p. 012002, dec 2015.

[18] H. Wang, Z. Li, J. Gu, Y. Ding, D. Z. Pan, and S. Han, "Qoc: Quantum on-chip training with parameter shift and gradient pruning," in *ACM/IEEE Design Automation Conference (DAC)*, ser. DAC '22. San Francisco, CA, USA: Association for Computing Machinery, 2022, p. 655–660.

[19] N. Saeed, H. Nam, M. I. U. Haq, and D. B. Muhammad Saqib, "A survey on multidimensional scaling," *ACM Comput. Surv.*, vol. 51, no. 3, may 2018.

[20] A. Unwin and K. Kleinman, "The iris data set: In search of the source of virginica," *Significance*, vol. 18, 2021.

[21] Y. Li, R.-G. Zhou, R. Xu, J. Luo, and W. Hu, "A quantum deep convolutional neural network for image recognition," *Quantum Science and Technology*, vol. 5, no. 4, p. 044003, jul 2020.

[22] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, "Hmdb: A large video database for human motion recognition," in *2011 International Conference on Computer Vision*. Barcelona, Spain: Institute of Electrical and Electronics Engineers, 2011, pp. 2556–2563.

[23] M. Weigold, J. Barzen, F. Leymann, and M. Salm, "Data encoding patterns for quantum computing," in *Proceedings of the 27th Conference on Pattern Languages of Programs*, ser. PLoP '20. USA: The Hillside Group, 2022.

[24] Q. Quantum, "Qrigin wukong cloud service," https://console.originqc.com.cn/zh/computerServices/services, 2024.

[25] ——, "Pyqpanda document," https://pyqpanda-tutorial-en.readthedocs.io/en/latest/, 2023.

[26] M. Alam, S. Kundu, R. O. Topaloglu, and S. Ghosh, "Quantum-classical hybrid machine learning for image classification (iccad special session paper)," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. Grenoble, France: Institute of Electrical and Electronics Engineers, 2021, pp. 1–7.

[27] T. Khare, R. Majumdar, R. Sangle, A. Ray, P. Seshadri, and Y. Simmhan, "Parallelizing quantum-classical workloads: Profiling the impact of splitting techniques," in *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, vol. 1. Los Alamitos, CA, USA: IEEE Computer Society, sep 2023, pp. 990–1000.

[28] M. Wang, P. Das, and P. J. Nair, "Qoncord: A multi-device job scheduling framework for variational quantum algorithms," in *Proceedings of the 57th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '57, 2024.

[29] H. Wang, J. Gu, Y. Ding, Z. Li, F. T. Chong, D. Z. Pan, and S. Han, "Quantumnat: Quantum noise-aware training with noise injection, quantization and normalization," in *ACM/IEEE Design Automation Conference (DAC)*, ser. DAC '22. San Francisco, CA, USA: Association for Computing Machinery, 2022, p. 1–6.

[30] S. Tan, C. Lang, L. Xiang, S. Wang, X. Jia, Z. Tan, T. Li, J. Yin, Y. Shang, A. Python *et al.*, "Quct: A framework for analyzing quantum circuit by extracting contextual and topological features," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 494–508.

Lookup Table-based Multiplication-free All-digital DNN Accelerator Featuring Self-Synchronous Pipeline Accumulation

Hiroto Tagata
Graduate School of Informatics
Kyoto University
Kyoto, Japan
htagata@easter.kuee.kyoto-u.ac.jp

Takashi Sato
Graduate School of Informatics
Kyoto University
Kyoto, Japan
takashi@i.kyoto-u.ac.jp

Hiromitsu Awano
Graduate School of Informatics
Kyoto University
Kyoto, Japan
awano@i.kyoto-u.ac.jp

***Abstract*—Deep neural networks (DNNs) have been widely applied in our society, yet reducing power consumption due to large-scale matrix computations remains a critical challenge. MADDNESS is a known approach to improving energy efficiency by substituting matrix multiplication with table lookup operations. Previous research has employed large analog computing circuits to convert inputs into LUT addresses, which presents challenges to area efficiency and computational accuracy. This paper proposes a novel MADDNESS-based all-digital accelerator featuring a self-synchronous pipeline accumulator, resulting in a compact, energy-efficient, and PVT-invariant computation. Post-layout simulation using a commercial 22nm process showed that 2.5× higher energy efficiency (174 TOPS/W) and 5× higher area efficiency (2.01 TOPS/mm^2) can be achieved compared to the conventional accelerator.**

***Index Terms*—Approximate Matrix Multiplication, Computing-in-Memory, Convolutional Neural Network, SRAM, Binary Decision Tree.**

I. Introduction

Deep neural networks (DNNs) are now widely applied in various fields, such as pattern recognition and natural language processing [1]–[3]. As their performance advances, DNNs continue to expand in parameter size[4], resulting in increased inference costs, mainly due to the high computational demands of multiply-and-accumulate (MAC) operations. In particular, the memory access needed to retrieve large amounts of weight parameters during inference is a significant factor in increasing energy consumption and latency [5].

Computing-in-Memory (CIM) is a leading approach to reducing memory access costs by integrating computation circuits directly into memory, thereby enhancing inference efficiency [6]. CIM is categorized into two types: analog and digital. Analog CIM employs voltage, current, and delay for computation, enabling high-speed, energy-efficient MAC operations without relying on advanced process nodes [7]. However, its inference accuracy is susceptible to process, voltage, and temperature (PVT) variations, and the increasing demand for multi-bit input/output support has introduced substantial costs associated with DA/AD conversion [8], [9]. In contrast, digital CIM offers high precision and robustness against PVT variations, with the potential for enhanced power, performance, and area (PPA) through scaling [10], [11]. However, it relies on large multipliers and adders, constraining energy efficiency. In particular, multipliers consume 6 to 31 times more energy and occupy 8 to 25 times more area than adders [12], [13].

One solution to reduce the high computational cost of multipliers is approximate matrix multiplication (AMM) using lookup tables (LUT) [14]–[18]. Especially, MADDNESS (Multiply-ADDitioN-lESS)[16] employs a product quantization (PQ) technique[19], whereby the input activation is approximated by prototypes that have been learned prior to the inference. Since the multiplication of the prototypes and known synaptic weight can be computed offline, the precomputed results can be stored in LUT. This approach enables matrix multiplication through low-cost table lookup operations, significantly reducing computational energy. Although PQ introduces some accuracy loss, previous studies report that 16 or more prototypes can achieve accuracy comparable to conventional DNNs [16]–[18].

Nevertheless, the implementation of MADDNESS in hardware presents significant challenges. The computational costs associated with MADDNESS are dominated by the PQ and data readout cost from LUT. However, GPU lacks optimization for these operations, limiting both throughput and energy efficiency [20]. While some MADDNESS accelerators have been proposed, challenges remain: for example, [21] uses analog computation in PQ, making it vulnerable to accuracy degradation from PVT variations. Furthermore, the synthesizable accelerator proposed in [22] has high LUT read energy attributed to the redundant layout, reducing energy efficiency.

This paper proposes a MADDNESS-based all-digital accelerator featuring a self-synchronous pipeline accumulator. The proposed circuit employs a data-driven asynchronous design, eliminating the need for a global clock and providing robust PVT variation tolerance. Furthermore, it incorporates a dataflow-based decision tree classifier for the PQ circuit and an area-efficient memory using 10-transistor static random access memory (10T-SRAM), achieving substantial reductions in energy consumption compared to the previous design.

The contributions of this paper are as follows:

- This study introduces the first compact, energy-efficient, and PVT-invariant MADDNESS accelerator with a self-synchronous pipeline architecture.
- Post-layout simulation using Synopsys HSPICE®on a commercial 22nm bulk-CMOS process demonstrated 2.5× enhancement in energy efficiency (174 TOPS/W) and 5× improvement in area efficiency (2.01 TOPS/mm^2) in comparison to the conventional accelerator [21].

The remainder of this paper is organized as follows: Section II introduces the MADDNESS algorithm and reviews prior LUT-based accelerators. Section III details the proposed all-digital MADDNESS accelerator, which incorporates self-synchronous pipeline accumulation. Section IV presents a performance evaluation based on post-layout simulations in a 22nm process and provides comparisons with conventional accelerators. Finally, Section V summarizes this paper.

II. Background

MADDNESS[16] is a technique to reduce neural network inference costs by using approximate matrix multiplication (AMM) based on product quantization (PQ). It improves the inference accuracy after quantization by incorporating codebook generation into the training process of the neural network. This section provides an overview of the fundamental algorithms underlying PQ and MADDNESS, followed by a review of prior research on MADDNESS-based DNN accelerators.

A. Product Quantization

PQ is a dimensionality reduction technique that divides a high-dimensional vector space into smaller subspaces and quantizes each subspace independently [19]. To provide a clear illustration of PQ, consider the following example with a d-dimensional vector, $\mathbf{x} \in \mathbb{R}^d$. Initially, $\mathbf{x}$ is partitioned into M subvectors, resulting in:

$$\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_M], \quad \mathbf{x}_i \in \mathbb{R}^{d/M} \tag{1}$$

For each subvector $\mathbf{x}_i$, a separate codebook with K prototypes $\mathbf{C}_i = [\mathbf{c}_{i,1}, \mathbf{c}_{i,2}, \ldots, \mathbf{c}_{i,K}]$ is prepared in advance. Subsequently, each subvector $\mathbf{x}_i$ is mapped to the corresponding prototype by using an encoding function, $enc_i(\mathbf{x}_i)$, yielding the quantized representation of $\mathbf{x}$ as follows:

$$\mathbf{q}(\mathbf{x}) = [enc_1(\mathbf{x}_1), enc_2(\mathbf{x}_2), \ldots, enc_M(\mathbf{x}_M)] \tag{2}$$

The encoding function enc should be carefully designed as it influences the accuracy and efficiency of PQ. Specifically, it is crucial to design enc that has a low computational cost and does not degrade the inference accuracy of neural networks.

B. MADDNESS Algorithm

MADDNESS consists of two primary operations: encoding and decoding. Encoding corresponds to PQ, where the input is divided into subvectors, and the nearest prototype vector is identified for each subvector. In circuit implementations, encoding is equivalent to generating a one-hot LUT address. Decoding involves retrieving and accumulating the precomputed multiplication results of the subvectors and synaptic weights from memory based on the generated address.

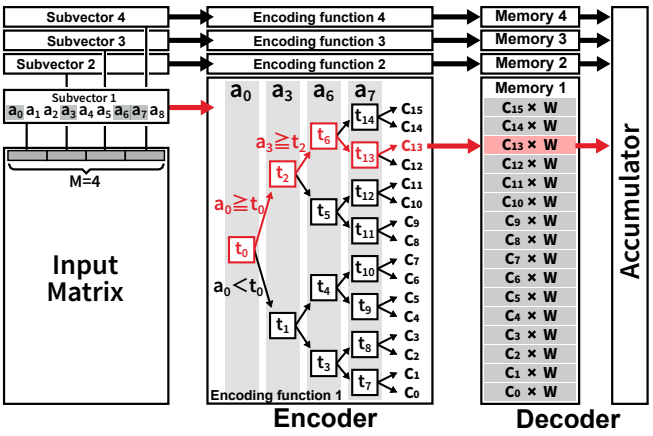


Fig. 1: Overview of the MADDNESS algorithm. The input vector is quantized into 16 prototypes by the decision tree classifier, and the corresponding multiplication results are loaded from memory to the accumulator.

Recent studies have proposed several MADDNESS-inspired algorithms. The main difference is the encoding function. The original MADDNESS proposed a balanced binary decision tree (BDT) based encoding function for fast and light inference. PECAN[17] employs Manhattan distance for encoding, while LUT-NN[18] utilizes Euclidean distance, to improve inference accuracy. Stella Nera[22] uses a matrix representation of the BDT to enable MADDNESS to be trained with backpropagation. In the remainder of this paper, the circuit implementations of these algorithms are collectively referred to as MADDNESS-based accelerators.

C. MADDNESS-based DNN Accelerator

The principal computational operation in MADDNESS accelerators is the calculation of an encoding function enc. Consequently, there has been a recent focus on the development of circuits capable of executing enc at high speeds with minimal energy consumption.

[21] proposed a method to efficiently compute enc as the similarity between the prototype and the input vector using time-domain analog circuit. The operational principle is as follows: firstly, the 6-bit precision input $\mathbf{x}$ and the 6-bit precision prototype $\mathbf{C}_i$ are converted into 60-bit thermometer codes. Subsequently, the circuit digital-to-time converter (DTC) is employed to calculate enc, whereby the Manhattan distance is converted into signal propagation delay. The DTC is constituted by a series of delay chains, with the k-th chain being responsible for computing similarity between input $\mathbf{x}$ and k-th prototype $\mathbf{C_k}$. The delay chain is constituted by delay cells, comprising variable delay elements and flip-flops connected in series. Each flip-flop pre-stores the prototype converted into thermometer codes. Therefore, by identifying the delay chain that provides the fastest signal propagation delay, it is possible to compute enc in an efficient manner. While this circuit demonstrates high energy efficiency through analog computing, the expansion of the codebook into thermometer codes necessitates a considerable allocation of area and memory capacity, with each n-bit codebook requiring 2^n-bit cells. Fur-

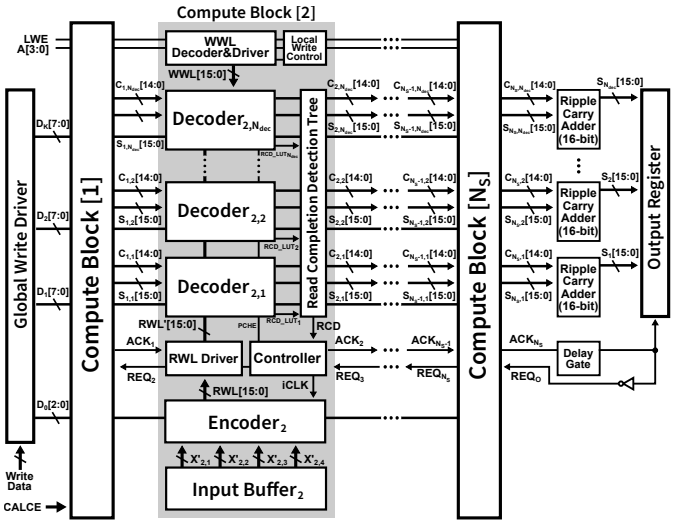


Fig. 2: Overall circuit of the proposed LUT-based DNN accelerator.

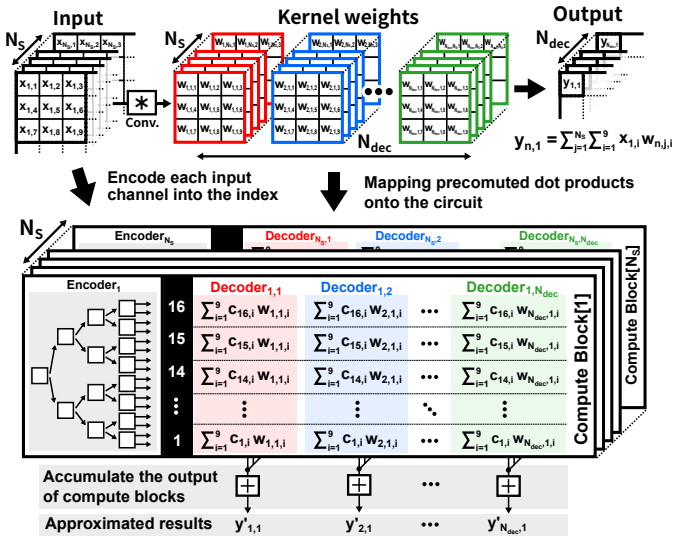


Fig. 3: CNN mapping onto the proposed circuit.

thermore, given the susceptibility of analog computing to PVT variations, the necessity arises for an additional circuit for post-fabrication calibration, with the objective of compensating for the computational errors that are caused by process variations.

To make the accelerator robust to PVT variation, [22] proposed a fully synthesizable all-digital accelerator featuring a decision tree-based classifier for PQ. Fig. 1 illustrates an example of the decision tree-based enc computation. A subvector (9-dimensional in the figure) extracted from the activation vector is classified into the corresponding prototype using a BDT. In this process, the elements of the subvector are compared with thresholds to determine the branch to be taken. In the example shown, the four elements, a_0, a_3, a_6, and a_7, are sequentially compared with the corresponding thresholds from left to right. Since there are four elements being compared in this example, the subvector is possible to be classified into $2^4 = 16$ prototypes. Based on the classification results, a lookup table is referenced to read out the precomputed inner product of the prototype and synaptic weights. This process is repeated for all four subvectors, and the results are accumulated to perform matrix multiplication. It has been reported that scaling the process node from 14nm to 3nm could achieve $4\times$ enhancement in energy efficiency and $32\times$ enhancement in area efficiency. However, the standard cell memory-based LUT consumes significantly higher energy and area than the standard SRAM-based LUT. Furthermore, the decision tree classifier requires additional memory costs for storing and reading threshold data, resulting in low energy efficiency.

This paper aims to improve energy efficiency by reducing the power consumption of both the PQ circuit and the LUT. The proposed all-digital macro incorporates a digital-CIM-based decision tree classifier for PQ and LUT based on area-efficient 10T-SRAM with an independent read port, achieving substantial reductions in energy consumption. Furthermore, since the proposed macro reliably operates in various PVT conditions by a global clock-free pipeline architecture, it can easily be implemented in advanced process nodes.

III. Proposed LUT-based DNN Accelerator

A. Overall Architecture

Fig. 2 illustrates the overall macro of the proposed accelerator. The proposed macro consists of multiple serially connected compute blocks, a global write driver to update the contents of SRAM cells in the compute blocks, a 16-bit ripple carry adder (RCA), and an output register. Each compute block comprises an encoder that maps input vectors to the corresponding prototypes and multiple decoders that read precomputed inner product from SRAM based on the output of the encoder and perform accumulation, a self-synchronous pipeline controller for inter-block operations, a read wordline (RWL) driver, a read completion detection (RCD) tree for handshake control, an input buffer, a local write driver, a write wordline (WWL) decoder, and a WWL driver. The encoder includes a 4-level decision tree classifier to perform PQ on the input, while the decoder contains a 10T-SRAM array to store precomputed results, a 16-bit carry-save adder (CSA) for LUT output accumulation, and registers to hold the CSA results.

The proposed circuit has two adjustable parameters, N_{S} and N_{dec}. N_{S} is the number of pipeline stages, and N_{dec} is the number of decoders per compute block. In convolutional neural network[23] inference, N_{S} corresponds to the number of input channels that can be processed concurrently, while N_{dec} corresponds to the number of weight kernels that can be processed simultaneously, as shown in Fig. 3. Increasing either parameter can reduce the overhead of peripheral circuits per operation. Notably, increasing N_{dec} reduces the encoder overhead to $1/N_{\mathrm{dec}}$, thereby enhancing area and energy efficiency. However, increasing N_{dec} also raises the WL wiring resistance within the LUT and the depth of the RCD tree, leading to increased latency. Additionally, since an internal critical path of each block limits block latency, worst-case delays may be increased for larger circuits. The optimal value for N_{dec} is evaluated in Section IV.

The operation of the compute block is as follows. First, prior to the inference, the precomputed dot products of the

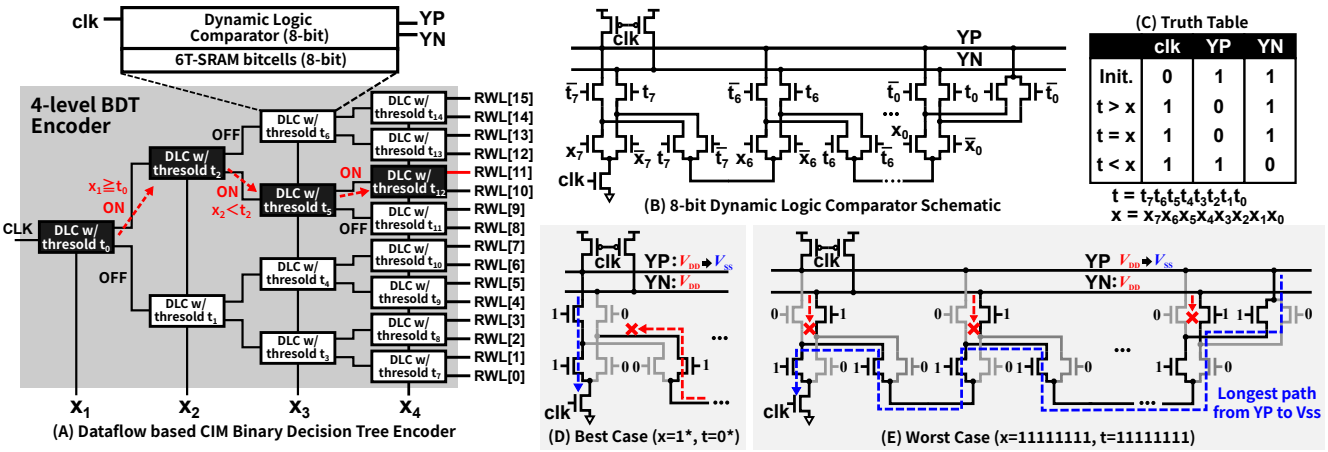


Fig. 4: Proposed decision tree-based encoder. (A) Overall schematic of the proposed encoder circuit. (B), (C) Schematic and truth table of the proposed dynamic logic comparator. (D), (E) Best and worst case operation of the DLC.

prototypes and synaptic weights are loaded in the corresponding SRAM cells embedded in decoders. In this paper, we employed an 8-bit integer precision. During the inference, the self-synchronous controller sets the block to an operation mode, which then activates the encoder to convert the subvector into the corresponding prototype. Similarly to [22], the proposed circuit employed a BDT for the encoding function, *enc*. Based on the mapped prototype, the decoder retrieves the precomputed dot product from the SRAM array. The retrieved data is accumulated with the output of the preceding block by the CSA, stored in the output latch of each decoder, and transmitted to the CSA of the subsequent block. Once all the lookup operations have been completed, the final summation is computed using the RCAs, and the result is stored in the output register.

Notably, all operations in the proposed circuit are performed asynchronously. In a typical clock-synchronized pipeline, the longest critical path among all stages determines the latency [24]. However, the asynchronous pipeline architecture allows each block to process the next input immediately, without waiting for the clock's rising edge, thereby minimizing latency [25]. Furthermore, this feature dynamically determines the operating speed, ensuring stable operation across various PVT conditions. A four-phase handshake protocol[26] is used for synchronization between compute blocks. Additionally, as will be detailed in Sec. III-B, the BDT inference engine also employs dynamic circuits, allowing only the necessary comparators to be activated automatically, which significantly reduces power consumption.

B. Decision Tree-based Encoder

As mentioned above, the proposed circuit uses a BDT in the encoder to classify subvectors into prototypes. Fig. 4(A) illustrates a block diagram of the encoder part of the proposed circuit. As shown, the encoder consists of dynamic logic comparators (DLCs) that compare the threshold values $\mathbf{t}$ with the elements of the subvector $\mathbf{x}$. To classify the subvectors into one of 16 prototypes, the proposed encoder connects 15 DLCs in a tournament structure. Conventional clock-synchronous-based circuit proposed in [22] require a global clock and many internal registers, resulting in high-power consumption. In contrast, the proposed encoder circuit achieves low-power computation through self-synchronous operation, eliminating both global clock and internal registers by employing dual-rail dynamic logic in the comparators. The red dotted line in Fig. 4(A) shows an example of how a subvector is classified into a prototype. First, all DLCs are precharged. Note here that each DLC stores a pre-learned threshold. Then, the root DLC is activated to compare the threshold t_0 with an element of the subvector $\mathbf{x_1}$. Depending on the result of the comparison, the next level DLC is selected to activate. By activating only the necessary DLCs, the circuit minimizes the discharge of precharged charges, thereby improving energy efficiency.

Fig. 4(B) and (C) show the schematic and the truth table of the DLC composed of a dual-rail dynamic logic. The DLC consists of eight 1-bit dynamic comparators connected in series. First, during the precharge phase (clk=0), both output nodes, YP and YN, are precharged to VDD. Additionally, the values of the two operands are set up in this phase. Then, in the evaluation phase (clk=1), the PMOS turns off, and depending on the input and threshold values, YP or YN is pulled down to VSS through the NMOS footer, as shown in the truth table. Note here that each 1-bit comparator discharges either YP or YN based on the comparison result if the comparison can be determined by the digit it is responsible for. If the comparison cannot be determined by its digit alone, the connection to the 1-bit comparator of the lower digit is activated, allowing YP/YN to be discharged based on the comparison of the lower digit. This method not only reduces the number of transistors required compared to comparators using naive subtractors, but also speeds up the comparison operation by immediately completing the calculation for those that can be determined by the higher digits alone, as shown in Figs. 4(D) and (E).

C. Two-port 10T-SRAM-based Decoder with CSA

After mapping the subvector to the corresponding prototype with the encoder, decoders retrieve the precomputed dot product

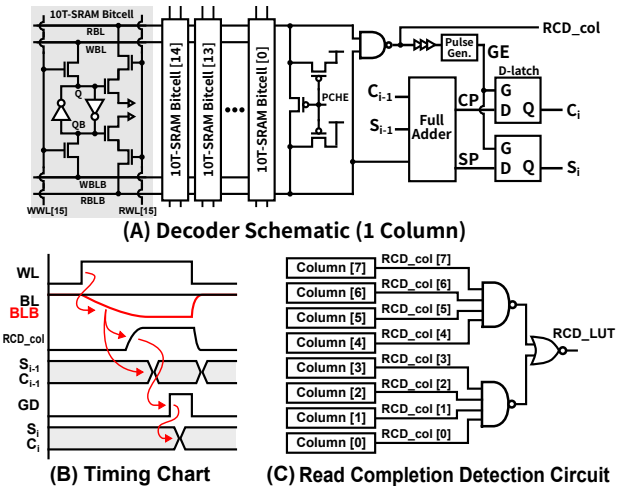


Fig. 5: 10T-SRAM based decoder with CSA unit. (A) Schematic of one SRAM column, (B) Timing chart of the decoding operation, (C) Schematic of the RCD circuit.

between the prototype and the synaptic weights from SRAM and perform accumulation. The proposed decoder consists of a 16-row, 8-column two-port 10T-SRAM array, a 16-bit CSA for accumulation, an output latch, and a 2NAND-1NOR circuit for RCD. This design facilitates high-speed data readout without the need for a sense amplifier (SA), as demonstrated in [10].

Fig. 5(A) and (B) illustrate the schematic and timing chart of a single column. The read operation proceeds as follows: first, based on the encoder output, one RWL in the column is asserted. Depending on the stored value, the selected SRAM cell fully discharges either RBL or RBLB. Subsequently, the full adder (FA) computes the carry and sum, and the output of the NAND gate (RCD_col) transitions to high. After a brief delay, the GE signal activates the latch, holding the FA outputs. When all RCD_col[i] signals in the LUT output high, they are combined into an overall RCD_LUT[i] signal via a NAND-NOR tournament, as depicted in Fig. 5(C). All RCD_LUT[i] signals in the same block are also aggregated into a single RCD[i] signal, as shown in Fig. 2, for the handshake control.

The proposed SRAM array incorporates a column-level RCD function to make the decoder robust to PVT variation. Traditional methods of estimating read latency often use a replica SRAM column, which accurately estimates read speed by mimicking the WL and BL layouts and conditions [27]. However, this approach is prone to column-to-column variability, requiring additional delay circuits for adjustment. In contrast, the proposed design features an independent RCD circuit for each column, enabling accurate detection even under high variability conditions. In addition, latch timing for the CSA outputs is generated simultaneously, effectively preventing setup violations over a wide range of PVT conditions.

IV. Evaluation

The performance of the proposed circuit was evaluated via post-layout simulation using Synopsys HSPICE®, based on a layout implemented in a commercial 22nm bulk-CMOS process. The total chip area and core area of the macro were

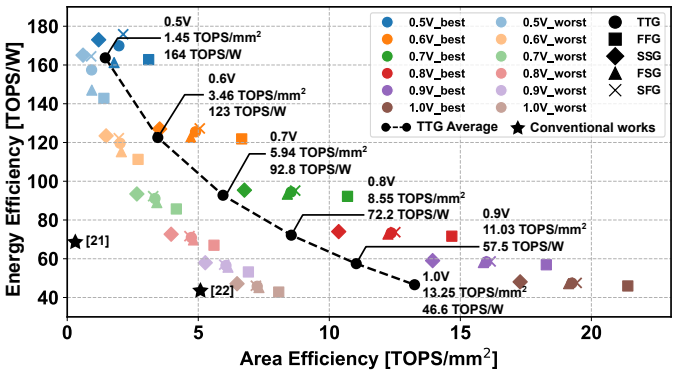


Fig. 6: Energy and area efficiency of the proposed circuit across various supply voltages and process corners. The black dashed line shows the average performance of the worst and best case at TTG corner. The upper right edge indicates better performance.

0.66 mm^2 and 0.20 mm^2, including 64kb SRAM ($N_{\text{dec}} = 16$, $N_{\text{S}} = 32$).

Fig. 6 evaluates the performance of the proposed circuit in terms of energy efficiency (TOPS/W) and area efficiency (TOPS/mm^2) across various supply voltages (0.5V-1.0V) and process corners (TTG, FFG, SSG, SFG, FSG). Simulation settings are 25 °C, $N_{\text{dec}} = 4$, and $N_{\text{S}} = 4$. Since the latency of the BDT encoder varies depending on the input and BDT threshold values, as described in Sec. III-B, both the worst and best cases are shown in the figure. The black dashed line illustrates the average of the worst and best performances at the TTG corner. At 0.5V, the circuit achieves the highest energy efficiency of 164 TOPS/W, while at 1.0V, it demonstrates the highest area efficiency of 13.25 TOPS/mm^{2}on average, highlighting a trade-off between energy consumption and throughput. The energy efficiency of the proposed circuit is determined mainly by supply voltage, which is nearly constant regardless of process corners and BDT encoder latency. Compared to conventional works (marked by black stars), the proposed circuit outperforms in both energy and area efficiency, particularly excelling in low-power scenarios.

Fig. 7 shows the energy, latency, and area breakdown for $N_{\text{dec}} = 4, 16$ and $N_{\text{S}} = 32$ at 0.5V. For energy, over 94 % of consumption for both $N_{\text{dec}} = 4$ and $N_{\text{dec}} = 16$ was attributed to the decoder, including the SRAM array and CSA. Similarly, the decoder accounted for a significant proportion (50 % to 80 %) of the area. This is because the energy and area of the decoder increase as $\mathcal{O}(N_{\text{dec}} \cdot N_{\text{S}})$, while other components increase only as $\mathcal{O}(N_{\text{dec}})$ or $\mathcal{O}(N_{\text{S}})$. In contrast, for latency, the encoder constituted the largest proportion over 40 % to 70 %. Comparing $N_{\text{dec}} = 4$ and $N_{\text{dec}} = 16$, the relative proportion of non-decoder components for energy and area decreases. On the other hand, the decoder's latency increased slightly due to the increased delay in the RCD.

To estimate the impact of N_{dec} values on performance more precisely, Table I shows the area and energy efficiency for $N_{\text{dec}} = 4$, 8, 16, and 32. Simulations were conducted under TTG corner at 25 °C, $N_{\text{S}} = 32$, and two supply voltage levels: 0.5 V for maximum energy efficiency and 0.8 V, the nominal

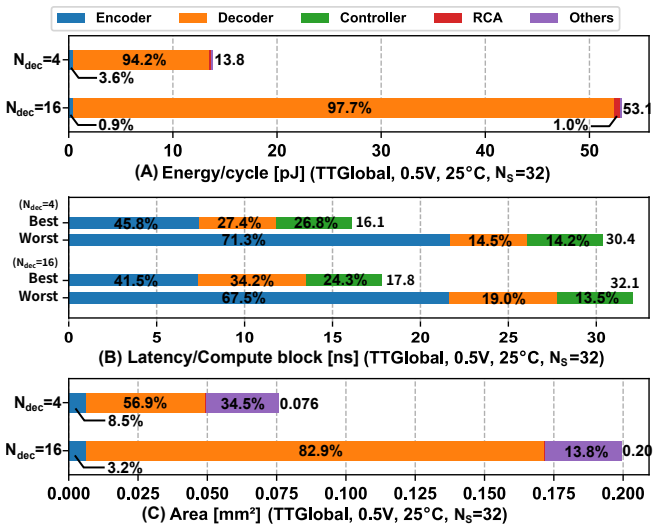


Fig. 7: (A) Energy, (B) latency, and (C) area breakdown based on post-layout simulation results.

TABLE I: Performance for different N_{dec}

Voltage	$N_{\text{dec}}=4$	$N_{\text{dec}}=8$	$N_{\text{dec}}=16$	$N_{\text{dec}}=32$
Energy efficiency [TOPS/W]				
0.5V	167.5	171.8 (+2.6%)	174.0 (+3.9%)	174.9 (+4.4%)
0.8V	73.0	74.4 (+1.0%)	75.1 (+1.0%)	75.4 (+1.0%)
Area efficiency [TOPS/mm^2]				
0.5V	1.4	1.8 (+28.6%)	2.0 (+42.9%)	2.0 (+42.9%)
0.8V	8.7	10.8 (+24.1%)	11.3 (+29.9%)	11.5 (+32.2%)

* The numbers in parentheses indicate the improvement rate from $N_{\text{dec}}=4$.

voltage of 22nm process. The figure indicates increasing N_{dec} improves both area and energy efficiency, with $N_{\text{dec}}=32$ demonstrate 32.2 % to 42.9 % improvements in area efficiency and 1.0 % to 4.4 % improvements in energy efficiency compared to $N_{\text{dec}}=4$. However, further increases in N_{dec} do not yield additional performance enhancements. For instance, the performance gain between $N_{\text{dec}}=32$ and $N_{\text{dec}}=16$ is 0 % to 2 %, which is almost negligible. Since larger N_{dec} values make the circuit vulnerable to local variations, $N_{\text{dec}}=16$ is recommended to achieve an optimal balance between performance and reliability.

Finally, Table II provides a comparison of the proposed circuit ($N_{\text{dec}}=16$, $N_{\text{S}}=32$) with state-of-the-art works. Area efficiency was normalized to 22nm process for a fair comparison. For example, circuits implemented in a 65nm process were scaled by $(65/22)^2$ based on the reported data. However, since [21] employs analog computation in the encoder, area scaling was applied only to the digital parts. The simulation results show that the proposed circuit achieved 2.5× higher energy efficiency and 5× higher area efficiency at a supply voltage of 0.5V compared to [21]. Note that the performance of [21] does not include accumulator overhead. Compared to [22], the proposed circuit achieves 4.0× the energy efficiency, but the area efficiency is 25 % lower. However, at a supply voltage of 0.8 V, the proposed circuit achieved 1.7× increase in the energy efficiency and 4.2× increase in the area efficiency, exceeding in both metrics. Instead of high energy efficiency, the proposed circuit has a lower operating frequency compared to

TABLE II: Comparison to prior accelerators.

	TCAS-I′23 [21]	arXiv′23 [22]	Proposed (N_{dec}=16, N_{S}=32)	
Measured /Simulated	Measured	Simulated	Simulated	
Operation Mode	MADDNESS (Analog)	MADDNESS (Digital)	MADDNESS (Digital)	
Process [nm]	65 (Planar)	14 (FinFET)	22 (Planar)	
Power Supply [V]	0.35/0.6/1.0[2]	0.55	0.5	0.8[1]
Area [mm^2]	0.31	0.57	0.20	
Frequency [MHz]	77	624	31.2-56.2	144-353
LUT Precision	INT8[3]	INT8	INT8	
Throughput [TOPS]	0.089	2.9	0.28-0.51	1.33-3.26
Energy Efficiency [TOPS/W]	69	43.1	174	75.1
Area Efficiency [TOPS/mm^2]	0.29 (0.40)[4]	5.1 (2.70)[4]	2.01	11.34
ResNet9 Acc. (CIFAR-10)	89.0	92.6	92.6	92.6
Energy/op. [fJ] (Encoder)	7.47	1.27	0.054	0.11
Energy/op. [fJ] (Decoder)	7.02[5]	16.47	5.6	14.7

[1] Nominal voltage of the 22nm process. [2] Multiple V_{DD} structure.
[3] Adjustable between INT4 and INT32. [4] Scaled to 22nm process.
[5] Accumulator is not included.

the [22], resulting in lower throughput. However, the smaller LUT layout of the proposed circuit allows it to maintain comparable area efficiency. Therefore, the proposed circuit can be applicable to situations where higher throughput is required by dividing the macros, although an additional adder is required. The performance improvement in energy efficiency is mainly attributed to the energy reduction in the decoder by employing an efficient 10T-SRAM array, which consumes 66 % less energy than standard cell memory-based LUT [22]. Additionally, the encoder reduced energy consumption by 95 % owing to eliminating the threshold readout cost and internal registers. It is also noteworthy that the classification accuracy on the CIFAR-10 dataset using ResNet9 was identical to that reported in [22].

V. Conclusion

This paper proposed a novel MADDNESS-based all-digital accelerator that significantly enhances energy and area efficiency for deep neural network inference. By employing a self-synchronous pipeline architecture, the design eliminates the need for a global clock, ensuring robust operation across various PVT conditions. The integration of a low-power binary decision tree encoder enables efficient PQ, while the implementation of an area-efficient lookup table using two-port 10T-SRAM substantially reduced energy consumption during decoding. Post-layout simulations using a commercial 22nm process demonstrated a 2.5× improvement in energy efficiency (174 TOPS/W) and a 5× increase in area efficiency (2.01 TOPS/mm^2) compared to conventional accelerators.

Acknowledgement

This work was supported by Japan Science and Technology Agency (JST), PRESTO Grant Number JPMJPR22B1 and JST BOOST, Grant Number JPMJBS2407, Japan.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, 84–90, May 2017. DOI: 10.1145/3065386.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[3] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training Very Deep Networks," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2015.

[4] B. Steiner, M. Elhoushi, J. Kahn, and J. Hegarty, *OLLA: Optimizing the Lifetime and Location of Arrays to Reduce the Memory Usage of Neural Networks*, 2022. arXiv: 2210.12924 [cs.LG].

[5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017. DOI: 10.1109/JSSC.2016.2616357.

[6] J. Zhang, Z. Wang, and N. Verma, "In-Memory Computation of a Machine-Learning Classifier in a Standard 6T SRAM Array," *IEEE J. Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017. DOI: 10.1109/JSSC.2016.2642198.

[7] H. Wang, R. Liu, R. Dorrance, D. Dasalukunte, D. Lake, and B. Carlton, "A Charge Domain SRAM Compute-in-Memory Macro With C-2C Ladder-Based 8-Bit MAC Unit in 22-nm FinFET Process for Edge Inference," *IEEE J. Solid-State Circuits*, vol. 58, no. 4, pp. 1037–1050, 2023. DOI: 10.1109/JSSC.2022.3232601.

[8] K. Lee, S. Cheon, J. Jo, W. Choi, and J. Park, "A Charge-Sharing Based 8T SRAM In-Memory Computing for Edge DNN Acceleration," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2021, pp. 739–744. DOI: 10.1109/DAC18074.2021.9586103.

[9] K. Yoshioka, "34.5 A 818-4094TOPS/W Capacitor-Reconfigured CIM Macro for Unified Acceleration of CNNs and Transformers," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 67, 2024, pp. 574–576. DOI: 10.1109/ISSCC49657.2024.10454489.

[10] H. Fujiwara, H. Mori, W.-C. Zhao, *et al.*, "34.4 A 3nm, 32.5TOPS/W, 55.0TOPS/mm^2 and 3.78Mb/mm^2 Fully-Digital Compute-in-Memory Macro Supporting INT12 $\times$ INT12 with a Parallel-MAC Architecture and Foundry 6T-SRAM Bit Cell," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 67, 2024, pp. 572–574. DOI: 10.1109/ISSCC49657.2024.10454556.

[11] H. Kim, J. Mu, C. Yu, T. T.-H. Kim, and B. Kim, "A 1-16b Reconfigurable 80Kb 7T SRAM-Based Digital Near-Memory Computing Macro for Processing Neural Networks," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 4, pp. 1580–1590, 2023. DOI: 10.1109/TCSI.2022.3232648.

[12] M. Horowitz, "1.1 Computing's Energy Problem (and What We Can Do About It)," in *Proc. IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

[13] Y. Zhao, C. Li, X. Liu, R. Qian, R. Song, and X. Chen, "Patient-Specific Seizure Prediction via Adder Network and Supervised Contrastive Learning," *IEEE Trans. Neural Syst. Rehabil. Eng.*, vol. 30, pp. 1536–1547, 2022. DOI: 10.1109/TNSRE.2022.3180155.

[14] Q. Deng, Y. Zhang, M. Zhang, and J. Yang, "LAcc: Exploiting Lookup Table-Based Fast and Accurate Vector Multiplication in DRAM-Based CNN Accelerator," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[15] Y. Jeon, B. Park, S. J. Kwon, B. Kim, J. Yun, and D. Lee, "BiQGEMM: Matrix Multiplication with Lookup Table for Binary-Coding-Based Quantized DNNs," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal. (SC20)*, 2020, pp. 1–14. DOI: 10.1109/SC41405.2020.00099.

[16] D. Blalock and J. Guttag, *Multiplying Matrices Without Multiplying*, 2021. arXiv: 2106.10860 [cs.LG].

[17] J. Ran, R. Lin, J. C. L. Li, J. Zhou, and N. Wong, *PECAN: A Product-Quantized Content Addressable Memory Network*, 2022. arXiv: 2208.13571 [cs.LG].

[18] X. Tang, Y. Wang, T. Cao, *et al.*, "LUT-NN: Empower Efficient Neural Network Inference with Centroid Learning and Table Lookup," in *Proc. 29th Annu. Int. Conf. Mobile Computing Netw. (ACM MobiCom '23)*, 2023. DOI: 10.1145/3570361.3613285.

[19] H. Jégou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, 2011. DOI: 10.1109/TPAMI.2010.57.

[20] C. McCarter and N. Dronen, *Look-ups are not (yet) all you need for deep learning inference*, 2022. arXiv: 2207.05808 [cs.LG].

[21] H. Fuketa, "Lookup Table-Based Computing-in-Memory Macro Approximating Dot Products Without Multiplications for Energy-Efficient CNN Inference," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 70, no. 10, pp. 3954–3963, 2023. DOI: 10.1109/TCSI.2023.3290151.

[22] J. Schönleber, L. Cavigelli, R. Andri, M. Perotti, and L. Benini, *Stella Nera: Achieving 161 TOp/s/W with Multiplier-free DNN Acceleration based on Approximate Matrix Multiplication*, 2023. arXiv: 2311.10207 [cs.AR].

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: 10.1109/5.726791.

[24] C. V. Ramamoorthy and H. F. Li, "Pipeline Architecture," *ACM Comput. Surv.*, vol. 9, no. 1, 61–102, Mar. 1977. DOI: 10.1145/356683.356687.

[25] S. Hauck, "Asynchronous design methodologies: an overview," *Proc. IEEE*, vol. 83, no. 1, pp. 69–93, 1995. DOI: 10.1109/5.362752.

[26] V. W.-Y. Sit, C.-S. Choy, and C.-F. Chan, "A four-phase handshaking asynchronous static RAM design for self-timed systems," *IEEE J. Solid-State Circuits*, vol. 34, no. 1, pp. 90–96, 1999. DOI: 10.1109/4.736660.

[27] B. Amrutur and M. Horowitz, "A Replica Technique for Wordline and Sense Control in Low-Power SRAMs," *IEEE J. Solid-State Circuits*, vol. 33, no. 8, pp. 1208–1219, 1998. DOI: 10.1109/4.705359.

Data Oblivious CPU: Microarchitectural Side-channel Leakage-Resilient Processor

Behnam Omidi
George Mason University
Fairfax, US
bomidi@gmu.edu

Ihsen Alouani
CSIT, Queen's University Belfast
Northern Ireland, UK
i.alouani@qub.ac.uk

Khaled N. Khasawneh
George Mason University
Fairfax, US
kkhasawn@gmu.edu

***Abstract*—Mitigating microarchitectural side channels remains a central challenge in hardware security. Despite substantial research efforts, current defenses are often narrowly tailored to specific vulnerabilities, leaving systems exposed to a broader spectrum of microarchitectural side-channel attacks. In this paper, we propose a generic approach to mitigate side-channel attacks with minimal architectural changes. Unlike traditional approaches that focus on mitigating specific side channels, we propose a dynamic strategy that alters the decoding of the instructions into secure (side-channel resilient) or performance versions of the instructions, based on the data it is processing. Specifically, to minimize performance overhead, decoding to a secure version is selectively applied only when sensitive data are being processed, ensuring optimal performance for instructions operating on non-sensitive data.**

To evaluate our approach, we implement it on the RISC-V out-of-order BOOM processor. Our results demonstrate that the mechanism increases the utilization of FPGA resources by only 2% compared to the original design. Furthermore, it imposes 0% performance overhead for unprotected applications, while maintaining overhead between up to 25% for security-critical workloads. This work represents a scalable and efficient solution for defending against micro-architectural side-channel attacks without compromising system performance.

***Index Terms*—Microarchitectural side-channel attacks, Information Flow Tracking,**

I. Introduction

Modern processors are equipped with several optimization features to boost their performance. Although these features (micro-architectural components), namely cache, branch prediction, etc, are invisible on the architectural level, changes in their states can be observable later by side-channel attacks. For example, in cache side-channel attacks such as Flush+Reload [1], and Prime+Probe [2], the attacker can predict the value of the sensitive data by measuring the difference between the miss and hit time on data residing in the cache.

Although there are numerous works in the literature that aim to mitigate microarchitectural side-channel attacks [3]–[10], they either suffer from high-performance overhead or can prevent only one type of architectural side-channel attack and leave others open to exploit.

In this paper, we address this challenge by enabling the CPU to selectively bypass the use of micro-architectural components when executing instructions that process sensitive data, ensuring robust security without compromising performance for instructions that are processing non-sensitive data. To achieve this aim, the user is only required to annotate sensitive data at the application level. The operating system automatically maps these data to a sensitive location in memory. The processor tracks the propagation of sensitive data using information flow tracking. Lastly, the decoder decodes instructions to secure (bypasses microarchitectural components) or performance (uses micro-architectural components) based on the data the instructions are processing, whether the data are sensitive or not.

Furthermore, in this paper, we evaluate the security of our system against existing microarchitectural attacks [11], [12] and indicate that by annotating the secret in the program, the system blocks the creation of side channels based on microarchitectural states. We also analyze our proposal in terms of hardware area and performance. In terms of area, our proposal takes $< 2\%$ the area of the unmodified BOOM processor [13]. For performance overhead, we run the SPECspeed 2017 integer benchmark and modify some security-critical applications by annotating sensitive data. We show that the system has 0% performance overhead for unprotected applications and the overhead up to 25% for protected applications.

The key contributions of this paper are as follows.

- We introduced the Data Oblivious CPU design, a novel processor architecture that dynamically adapts instruction execution to mitigate microarchitectural side-channel attacks. Specifically, the proposed design employs a hybrid execution model, which decodes instructions into secure or performance versions based on the sensitivity of the data being processed.
- We provide a complete hardware prototype of our ideas, built on top of the RISC-V out-of-order BOOM processor, demonstrating the feasibility and practicality of our approach. All modifications in the BOOM architecture and software stacks are available online: https://github.com/beomidi/data_oblivious_cpu
- We evaluate the security, area, and performance of our methodology and show that it has minor overhead on hardware complexity, blocks micro-architectural side-channel attacks, while having acceptable performance overhead on protected applications (up to 25%), while having 0% performance overhead on unprotected applications.

II. Background and Related Work

A. Berkeley Out-of-Order Processor

RISC-V is an open-source, royalty-free ISA which follows the Reduced Instruction Set Computing (RISC) principle. It offers three privilege levels: machine mode, supervisor mode, and user mode [14] to protect different components of the software stack. Due to its simplicity and flexibility, it allows hardware developers to specify many optional extensions and leverage it in many application areas, ranging from small embedded systems to large-scale, high-performance servers.

The University of California, Berkeley, has been developing and maintaining the Berkeley Out-of-Order Machine (BOOM), an open-source synthesizable parameterized out-of-order RISC-V processor [13]. The architecture of BOOM consists of seven pipeline stages: Fetch, Decode/Rename, Rename/Dispatch, Issue/RegisterRead, Execute, Memory, and Writeback (Commit is not counted in the pipeline due to its asynchronous occurrence). It also implements sv39 page-based virtual memory system that supports 39-bit virtual address spaces. To achieve maximum utilization of the execution unit, the processor leverages several optimization features. The first and most important one is the implementation of out-of-order execution functionality, which allows the processor not to wait for independent instructions to complete their execution in sequential order. It also has two levels of branch prediction- a fast Next-Line Predictor (a kind of Branch Target Buffer) and a slower but more complex Backing Predictor (a complicated structure like a GShare predictor). Moreover, it supports a private non-blocking cache (Hellacache) which is designed for in-order processors and is connected by a $shim$ to the BOOM. Correspondingly, the BOOM has all the convenient characteristics that can be exploited to establish microarchitectural attacks.

B. Micro-architectural Attacks

Micro-architectural attacks compromise the security of the system by monitoring the micro-architectural effects of specific optimization through side channels including cache access pattern [7], [15], [16], branch access patterns [17], faults [18], functional unit timing characteristics [19], power consumption [20], [21] and electromagnetic effects [22]. The attacker chooses an appropriate side-channel based on the characterization of the micro-architectural component. These characterizations can be categorized as state-less, state-full, private, or public [6]. State-less resources are resources such as functional units that don't maintain the state of executed instructions. However, State-full resources like caches and branch predictions keep the state of instruction. Private resources like private L1 cache refer to those that cannot be accessed from other cores. In contrast, public resources, such as the last-level shared cache, are shared among every core.

C. Information Flow Tracking

Several efforts have been made to track the information in the system and mitigate the side-channel attacks. They can be categorized as software-based and hardware-based solutions. Software-based mechanisms attempt to change the way of programming [8], [23] or analyze the code statically to insert some blocking instructions [24], [25] by considering the existing potential side channels. Although their solutions can alleviate some types of attack, they impose a large performance overhead on the system. Moreover, since the microarchitectural components are invisible to the developer, it is impossible to secure the application against all micro-architectural attacks at the software level. However, multiple hardware information flow tracking techniques have been described at various hardware levels. Taram *et al.* [10] propose a mechanism to dynamically alter the decoding of the instruction stream and inject new micro-ops (fences) under specific conditions. They mark any information from user address space IO devices as confidential information, which imposes so much overtainting space on the system, causing a large performance loss. Furthermore, the proposed micro-ops are memory-ordered instructions that can only prevent microarchitectural attacks on the memory hierarchy system. Yu *et al.* [26] attempt to extend the Instruction Set Architecture (ISA) of the processor to support data oblivious programming to be resilient against microarchitectural side-channels. Since they only consider oblivious program vulnerabilities, their implementation terminates the program when it violates a specific condition, making it difficult to debug the program. In addition, they consider part of the memory as storage of confidential information metadata, which increases the size of the application. Schwarz *et al.* [9] proposed a mechanism to alleviate transient execution attacks by tracking information from the application to the hardware. Although they discussed how to track the secret in the hardware, their evaluation is a software-based over-approximation and only considers the cache hierarchy as a source of the leakage. Sabbagh *et al.* [27] introduces an information flow tracking mechanism on the RISC-V architecture to activate the data memory fencing on the secret. Their tracking technique only considers speculative execution behavior and defends against cache-side channels.

III. Threat Model

We consider a scenario where a victim program operates on a shared computing platform alongside potentially adversarial software. The adversary's objective is to exploit microarchitectural side channels to extract private information from the victim program. This private information may include confidential inputs provided by external parties or sensitive internal program states, such as cryptographic keys. The victim program itself is assumed to be public.

We consider contention-based attacks [6], [28], fault injection attacks (for example, DRAM attacks [29], [30]), EM radiation [22], and power side-channel attacks [20] to be out of the scope of our study.

IV. Methodology

In this section, we present the design of Data Oblivious CPU, a microarchitectural side-channel leakage-resilient processor. The key idea behind the Data Oblivious CPU is to dy-

namically adapt the decoding of instructions based on the sensitivity of the data being processed. Specifically, we introduce architectural mechanisms to classify and track sensitive data, selectively switching between secure (side-channel-resilient) and high-performance modes of execution. The secure version of instructions ensures that microarchitectural components do not update or change their state based on sensitive data, effectively eliminating the risk of side-channel leakage. At the same time, high-performance execution is maintained for operations involving non-sensitive data, ensuring minimal impact on overall system performance.

Data Oblivious CPU employs a multi-level defense strategy consisting of the following components:

1) Sensitive memory mapping
2) Sensitive data tracking
3) Dynamic decoding
4) Data-dependent behavior elimination
5) Software support for the hardware feature

A. Sensitive Memory Mapping

In modern processors, there are plenty of reserved bits in the page table entries that are embedded for future use. Figure 1 illustrates the RISC-V architecture page table entry (sv39). Sv39 supports a 39-bit virtual address space divided into 4 KB pages. Each page table entry has 64 bits, including a 44-bit physical address field and several flags. There are 10 bits between bits 54 to 64 that are considered reserved bits, which can be used for the physical page number extension. Since it is always 0, repurposing the last reserved bit as a *Sensitive (S)* bit does not have any side effect on the system.

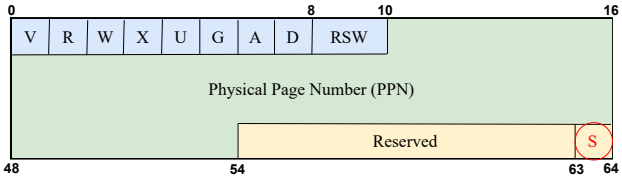


Fig. 1: RISC-V page table entry

B. Sensitive Data Tracking

Sensitive memory mapping ensures that sensitive data in memory are identified. However, to fully secure the system, it is necessary to extend this protection to registers that hold sensitive values. Registers in existing CPUs lack mechanisms for tracking the confidentiality of their content, requiring hardware modifications to implement this functionality. Inspired by prior work on hardware tainting mechanisms, we propose a simple yet effective tainting system to track and protect sensitive data in registers.

Specifically, we introduce an additional taint bit for each register. This bit indicates whether the value stored in the register is sensitive (if the bit is set). The taint propagates dynamically through the processor pipeline as follows: (a) *Memory to Register*: When a load instruction accesses sensitive memory, the taint bit of the destination register is set; (b) *Register to Register*: Operations involving one or more tainted registers, as source operands, result in tainting the destination register. This ensures that results derived from sensitive data are also treated as sensitive.

Registers are untainted in specific scenarios to prevent over-tainting, simplifying the tracking mechanism, and improving performance. Untainting occurs when a register's content is fully replaced with an immediate value or data from an untainted memory location, or when a tainted register is written to non-sensitive memory. This assumes that such writes are either intentional actions by the developer or program bugs to be addressed. For example, the output of a cryptographic cipher may not require continued protection, allowing the register to be untainted after the operation.

Additionally, taint information must be preserved across context switches and interrupts to ensure consistent tracking. We propose a dedicated Control Status Register (CSR) to store the taint state of all registers. During a context switch or interrupt, the operating system saves this register alongside other processor state information. Upon returning from the interrupt or context switch, the taint values are restored to maintain the integrity of the tracking system.

Lastly, since taint information is required at the decoding stage, we ensure accurate taint propagation by recovering the taint state from branch mispredictions by rolling back to the last committed taint state before the mispredicted branch. This can be achieved by encoding the speculative taint state into the reorder buffer as an additional state bit for each entry. When instructions are flushed due to a misprediction, the corresponding speculative taint state is also flushed. The speculative taint state is only moved to the committed taint state in the register file at the commit stage of the pipeline when the instruction is retired.

C. Dynamic decoding

A key innovation in the proposed architecture is its dynamic decoding mechanism, which adapts instruction execution based on the sensitivity of the data being processed. At the decoder stage, instructions are evaluated for their processing of tainted (sensitive) data. If an instruction involves tainted operands or operates on sensitive memory regions, it is dynamically translated into a secure version to ensure side-channel resilience. Conversely, instructions involving untainted data are translated into performance-optimized versions for maximum efficiency.

D. Data-dependent behavior elimination

To safeguard the system against side-channel attacks, it is essential to eliminate any observable microarchitectural footprint influenced by execution based on sensitive data. Therefore, to achieve broad compatibility and efficiency, the instruction set architecture (ISA) should support a range of instruction types designed for data-oblivious execution while minimizing hardware changes. This includes:

- Arithmetic Operations: All integer and floating-point arithmetic instructions should have a constant-time be-

havior version if they only have a data-dependent behavior version.

- Memory operations: The load and store instructions should have a secure version optimized for confidential addresses to implement oblivious memory access. For example, against cache side-channel attacks, all memory instructions accessing sensitive regions must bypass the cache hierarchy, ensuring that they leave no traceable cache or TLB state changes.
- Control Flow Operations: Control flow instructions should include a secure version for sensitive operands, ensuring that sensitive control flow decisions are executed securely and do not leave observable traces in microarchitectural components, enabling robust protection against side-channel leakage. For example, for the branch target buffer (BTB) and return stack buffer (RSB), the processor should not modify these components when executing instructions that modify their states, such as jumps, branches, and returns.

E. Software Support

To support the proposed hardware modifications, changes are required at the application, compiler, and operating system levels. Developers must annotate sensitive information in their applications, marking data that require confidentiality. The compiler processes these annotations, placing the identified sensitive data into a dedicated section within the program binary. The operating system is then responsible for mapping this section to sensitive memory pages, ensuring secure handling of the data.

Additionally, the operating system manages the preservation and restoration of register taint values during context switches and interrupts.

V. Implementation

In this section, we present the implementation of our mechanism on the BOOM processor to make it resilient against microarchitectural side-channel attacks. Since the method is a multi-level countermeasure, we first propose all modifications made to the software and then discuss hardware extensions to support the security of the processor.

A. Software

We propose a runtime library and kernel module and modify the compiler and operating system to follow the secret in the software. This section introduces all changes in the software in more detail.

Runtime Library: We develop a run-time library that can set confidential data in the sensitive memory region via a kernel module. The user is just required to include the library in their application and annotate the secret with the $nouarch$ keyword. The keyword guides the linker to allocate a secure section for the variable in the binary. In addition, the library has a constructor function that initializes the run-time requirement and only activates on application startup. To dynamically mark sensitive data, the library provides $malloc_nouarch()$ and $free_nouarch()$ functions that immediately allocate and release the heap memory as secure.

```
1  _save_context:
2  ...
3  csrr s6, CSR_TAINT //taintCSR -> s6
4  REG_S s6, PT_TAINT(sp) //s6 -> stack
5  ....
6
7  restore_all:
8  ...
9  REG_L a2, PT_TAINT(sp) stack -> a2
10 csrw CSR_TAINT, a2 // a2 -> taintCSR
11 ...
12 sret
```

Listing 1: (Pseudo-)assembly for keeping the taint CSR during a context switch

Compiler: In section IV-E, we mentioned that the compiler is responsible for creating a secure section in the binary. We exploit the existing attribute $__attribute__((annotate("secret")))$ to dedicate a specific section in the elf file. Therefore, all $nouarch$ keywords are replaced by the secret attribute, and the compiler automatically allocates a secret section in the binary.

Page-Table Entry: For backward and future compatibility, we repurposed bit 64 in the RISC-V page-table entry, which is a reserved bit to indicate the secret pages. By setting the bit, the page is considered a secret region in memory mapping.

Kernel Module: We develop some operating system modifications as a kernel module. The module is in charge of mapping the secure section in binary to the sensitive memory pages. It prepares an interface to the runtime library to change the corresponding bit in the entry (bit 64). The corresponding TLB entry and the cache are flushed subsequently to make sure that the sensitive data do not change the micro-architectural states.

Operating system: The operating system keeps and reloads the content of each register taint on the context switch and interrupts. The pseudo-code 1 illustrates the operating system saving and restoring process of the CSR register to/in the stack during the context switch. We spill the taint CSR value to the register taint value later in the hardware on the return instructions that occur at all interrupt routines. Therefore, in this process, we are not concerned with the overwriting of a register with an immediate value, which untaints the value of the register taint.

B. Hardware

We modify the TLB, registers, microarchitectural components, and CSRs to support secret tracking on the hardware level.

Translation Lookaside Buffer: Since the PTEs are cached in TLB, we modify the TLB to cache the secure bit while bringing the entry. Each memory request should translate

its virtual address to the effective address. Therefore, in the BOOM processor, upon calculation of the effective address, the secure bit propagates to the corresponding taint bit in the register. Since this operation (effective address calculation) happens in the execution and we need the information of the taint in decode time, the taint and the instruction are stored in a temporary register and the pipeline would be flushed.

Registers: To track the taint, all physical registers in the architecture are extended by a tain bit. Therefore, in the operation, if one of the physical source registers of the executing instruction is tainted, we propagate the taint value to the destination physical register. During the commit stage, the taint of the physical register is copied to the taint of the committed register. Moreover, the processor monitors all the immediate loads and insensitive memory requests to untaint the corresponding registers.

Micro-architectural components: All memory requests in BOOM pass through the Hellacache (a private cache in the architecture). Therefore, we modify this cache to uncache the sensitive data. Moreover, the processor flushes the BTB and RSB when decoding an instruction having a secret operand to clear the possible changes in the components. For instructions carrying sensitive information, the processor executes them in order by keeping them in the issue stages until reaching the head of the Reorder Buffer.

Control Status Register: As we discussed in section IV-B, tracking the information requires a structure in hardware to keep the status of taints during interrupts and context switches. Hence, we add a new CSR called taintCSR, which stores/restores the taint status on an interrupt (consider the context switch as a type of interrupt). We set the address of the register in the supervisor space (0x190) to be only accessible by the operating system. When one of the interrupt bits is set in the machine mode status register (mstatus), which is a CSR to track the status of the hardware thread's current operating state, all register taint values are loaded into taintCSR to be stored in the memory structure by the operating system. To track back and restore the taint values, the processor checks $sret$ and $mret$ instructions, which execute at the end of each interrupt routine.

VI. Evaluation

In this section, we discuss the security and performance evaluation of the proposed method. We modify the BOOM processor to evaluate our technique. The processor configuration has been shown in the table I. We used Ubuntu 22.04 LTS with modified kernel version 5.19.0 to run on BOOM. The Xilinx KCU105 evaluation board has been leveraged to evaluate our processor. To this end, we modify the configuration of the chipyard, an open-source integrated SoC design and simulation framework, to make it compatible with the existing board.

To compare the compute-intensive performance of the system, we run the Spec2017 benchmark on the original BOOM and the modified one. As we expected, the results demonstrate the 0% performance overhead since we do not annotate any data as secret and the system does not block any microarchitectural components. We use security-critical applications in [9] to measure the performance of our work by annotating the secret values on them.

Parameters	Value
CPU	1 core RISC-V out-of-order sonicBOOM
Core	32 ROB entries, 8 TLB entries, 8 Load Queue entries, 8 Store Queue entries, 52 physical registers, 48 floating point registers, 4 fetch size
L1-I Cache	2KB, 64-set, 4-way
L1-D Cache	2KB, 64-set, 4-way

TABLE I: Configuration of the BOOM

OpenSSL: We evaluated the performance overhead of our technique by encrypting a message using OpenSSL's chacha, and rc4 ciphers. By notating the key using the secure attribute, we protect the key during execution in the system.

OpenSSH: OpenSSH is the premier connectivity tool for remote login with the SSH protocol, consisting of remote operation tools like ssh, scp, sftp. The confidential information in these operations is the private key, which is stored in memory and is susceptible to micro-architectural attacks. Hence, to evaluate the performance of our system, we annotate the private key in the application and measure the time it takes to connect through the ssh or copy the file through scp.

OATH Tool: Open AuTHentication (OATH) one-time password tool is designed to generate and validate one-time passwords. The tool calculates a cryptographic hash over the shared secret and the current time using a shared key between the user and the service. To evaluate the performance of the application, we annotate the shared secret and generate one-time passwords.

Lastpass-cli: The LastPass command-line application is an open-source application that allows users to create, edit, and retrieve passwords. Connects to a remote server and the user needs to provide a master password to access the data. To enhance the security of the tool, we annotate buffers that store the master password and the decryption key with the secure attribute. We repeatedly queried a password to measure the performance overhead of the technique on the application.

VeraCrypt: VeraCrypt is a free open-source disk encryption software for Windows, Mac OSX, and Linux. To store confidential information, VeraCrypt uses the SecureBuffer class. Hence, we mark all SecureBuffer instantiations in the program to protect the application. We analyze the performance of the application by mounting and encrypting the data with a password and key.

NGINX: Engine x is an HTTP web server that can be used for a variety of web server tasks, including Web serving, Load balancing, and reverse proxying, etc. It can also be used as an HTTPS server. Therefore, we modified NGINX to protect the certificate key in HTTPS communication. We use siege load testing and a benchmarking utility to put the server under stress testing to evaluate its performance. We simulate 255 clients for

300 seconds with siege. With this stress test, the number of transactions has decreased from 59 to 56, which means 5.08% performance overhead.

Figure 2 shows the performance overhead for the rest of the security-critical applications. The results indicate increasing the execution time between 1.33% for the *chacha* to 25.63% for the *ssh*

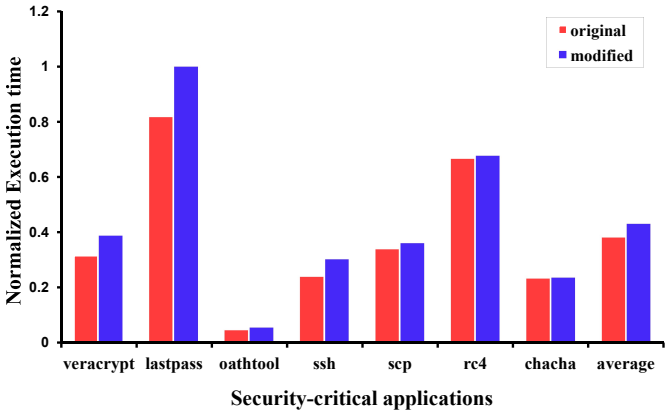


Fig. 2: Performance overhead for security-critical applications

To measure the area overhead of the proposed method, we generate the post-synthesis report of the BOOM processor on the Kintex UltraScale XCKU040-2FFVA1156E device. The different utilizations of the original and modified BOOM resources are shown in Table II, which indicates an increase of 0% to 3% for the modified BOOM resources.

Resources	Original (#)	Modified (#)	Overhead (%)
LUT1	1025	1004	-2.04
LUT2	6268	6127	-2.24
LUT3	14602	15053	+3.08
LUT4	15105	14484	-0.41
LUT5	18155	18403	+1.36
LUT6	65103	65980	+1.34
MUXF7	11300	11576	+2.44
MUXF8	2535	2543	+0.31
FDRE	76841	78201	+1.76
FDSE	1236	1229	-0.56
Total	211145	213596	1.16

TABLE II: Resource utilization of the original BOOM and modified one on KCU105 FPGA (LUT# = # inputs Lookup Table, MUXF# = # inputs multiplexer, FDRE = D-type flip-flop with clock enable and synchronous reset, FDSE= D-type flip-flop with clock enable and synchronous set)

To evaluate the security of the system, we execute all existing microarchitecture side-channel attacks [11], [12] on RISC-V architecture. The system equipped with our mechanism has not shown any leakage in any of them. Figure 3 demonstrates the impact of our approach on cache timing side-channel attacks (one of the common exploitable attacks) represented in pseudo-code 2. The program was written to measure the hit-and-miss rates of a specific address. The experiment shows that the timing for hit and miss is distinguishable for the original system, while the modified one maps both times to a single point.

```
char nouarch *address;

for(int i=0; i<num_of_measurements; i++){
    hit = measure_access_time(address);
    hit_histogram[hit]++;
}

for(int i=0; i<num_of_measurements; i++){
    //Flush the cache in original system
    //Remove the line for modified system
    flushCache(address, sizeof(address))
    miss = measure_access_time(address);
    miss_histogram[miss]++;
}
...
```

Listing 2: Pseudo-code of cache timing side-channel attack

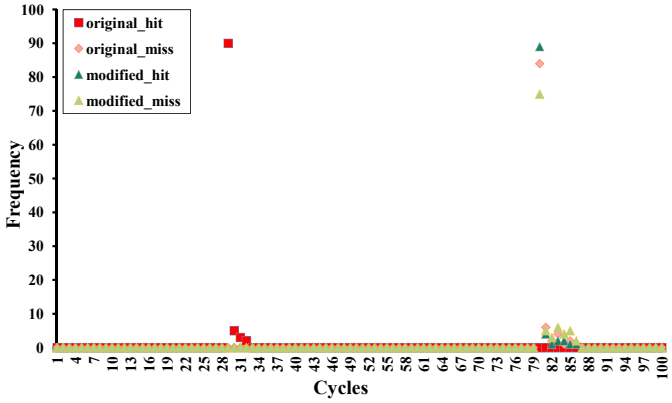


Fig. 3: Cache timing side-channel attack effect on system

VII. Conclusion

In this paper, we introduce a technique to mitigate microarchitectural side-channel attacks on state-of-the-art processors. We modify all levels of abstraction on the BOOM processor, including the application, compiler, operating systems, and hardware, to track sensitive information and eliminate all the footprints leveraged by these attacks. We evaluate our technique on commonly used microarchitectural attacks and some security-critical applications. Our results illustrate that our method can block side-channel attacks and has 0% performance overhead in unprotected applications and up to 25% overhead on average in protected applications.

VIII. Acknowledgment

The work in this paper is partially supported by the National Science Foundation grant CNS-2155002.

References

[1] Y. Yarom and K. Falkner, "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack," in *23rd USENIX security symposium (USENIX security 14)*, 2014, pp. 719–732.

[2] C. Trippel, D. Lustig, and M. Martonosi, "Meltdownprime and spectreprime: Automatically-synthesized attacks exploiting invalidation-based coherence protocols," *arXiv preprint arXiv:1802.03802*, 2018.

[3] M. Kayaalp, K. N. Khasawneh, H. A. Esfeden, J. Elwell, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "Ric: Relaxed inclusion caches for mitigating llc side-channel attacks," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[4] J. Bahrami, V. B. Dang, A. Abdulgadir, K. N. Khasawneh, J.-P. Kaps, and K. Gaj, "Lightweight implementation of the lowmc block cipher protected against side-channel attacks," in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security*, 2020, pp. 45–56.

[5] A. Dhavlle, S. Rafatirad, K. Khasawneh, H. Homayoun, and S. M. P. Dinakarrao, "Imitating functional operations for mitigating side-channel leakage," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 868–881, 2021.

[6] N. Nazari, B. Omidi, C. Fang, H. M. Makrani, S. Rafatirad, A. Sasan, H. Homayoun, and K. N. Khasawneh, "Specscope: Automating discovery of exploitable spectre gadgets on black-box microarchitectures," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[7] J. Zhang, C. Chen, J. Cui, and K. Li, "Timing side-channel attacks and countermeasures in cpu microarchitectures," *ACM Computing Surveys*, vol. 56, no. 7, pp. 1–40, 2024.

[8] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, "Obliviate: A data oblivious filesystem for intel sgx." in *NDSS*, 2018.

[9] M. Schwarz, M. Lipp, C. A. Canella, R. Schilling, F. Kargl, and D. Gruss, "Context: A generic approach for mitigating spectre," in *Network and Distributed System Security Symposium 2020*, 2020.

[10] M. Taram, A. Venkat, and D. Tullsen, "Mitigating speculative execution attacks via context-sensitive fencing," *IEEE Design & Test*, 2022.

[11] L. Gerlach, D. Weber, R. Zhang, and M. Schwarz, "A Security RISC: Microarchitectural Attacks on Hardware RISC-V CPUs Artifact Repository," 2023. [Online]. Available: https://github.com/cispa/Security-RISC

[12] M. Sabbagh, Y. Fei, and D. Kaeli, "Secure speculative execution via risc-v open hardware design," June 2021.

[13] C. Celio, D. A. Patterson, and K. Asanovic, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-167*, 2015.

[14] A. Waterman, K. Asanovic, and J. Hauser, "The risc-v instruction set manual volume ii: Privileged architecture version 1.12," *EECS Department, University of California, Berkeley*, 2021.

[15] S. H. W. Cheng, C. Chuengsatiansup, D. Genkin, D. McNeil, T. Murray, Y. Yarom, and Z. Zhang, "Evict+ spec+ time: Exploiting out-of-order execution to improve cache-timing attacks," *Cryptology ePrint Archive*, 2024.

[16] H. Naghibijouybari, K. N. Khasawneh, and N. Abu-Ghazaleh, "Constructing and characterizing covert channels on gpgpus," in *Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture*, 2017, pp. 354–366.

[17] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev, "Branchscope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 693–707, 2018.

[18] J. J. Hoch and A. Shamir, "Fault analysis of stream ciphers," in *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*. Springer, 2004, pp. 240–253.

[19] Z. Wang and R. B. Lee, "Covert and side channels due to processor architecture," in *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 2006, pp. 473–482.

[20] P. Kocher, "Differential power analysis," in *Proc. Advances in Cryptology (CRYPTO'99)*, 1999.

[21] N. Nazari, C. Fang, H. Mohammadi Makrani, B. Omidi, M. Eslamimehr, S. Rafatirad, A. Sasan, H. Sayadi, K. N. Khasawneh, and H. Homayoun, "Architectural whispers: Robust machine learning models fingerprinting via frequency throttling side-channels," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[22] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Cryptographic Hardware and Embedded Systems—CHES 2001: Third International Workshop Paris, France, May 14–16, 2001 Proceedings 3*. Springer, 2001, pp. 251–261.

[23] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital {Side-Channels} through obfuscated execution," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 431–446.

[24] O. Oleksenko, B. Trach, T. Reiher, M. Silberstein, and C. Fetzer, "You shall not bypass: Employing data dependencies to prevent bounds check bypass," *arXiv preprint arXiv:1805.08506*, 2018.

[25] M. F. A. Kadir, J. K. Wong, F. Ab Wahab, A. F. A. A. Bharun, M. A. Mohamed, and A. H. Zakaria, "Retpoline technique for mitigating spectre attack," in *2019 6th International Conference on Electrical and Electronics Engineering (ICEEE)*. IEEE, 2019, pp. 96–101.

[26] J. Yu, L. Hsiung, M. El Hajj, and C. W. Fletcher, "Data oblivious isa extensions for side channel-resistant and high performance computing," *Cryptology ePrint Archive*, 2018.

[27] M. Sabbagh and Y. Fei, "Secure speculative execution via risc-v open hardware design," in *Fifth Workshop on Computer Architecture Research with RISC-V (CARRV 2021)*, 2021.

[28] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer, and A. Kurmus, "Smotherspectre: exploiting speculative execution through port contention," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 785–800.

[29] O. Mutlu and J. S. Kim, "Rowhammer: A retrospective," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1555–1571, 2019.

[30] A. G. Yağlikçi, M. Patel, J. S. Kim, R. Azizi, A. Olgun, L. Orosa, H. Hassan, J. Park, K. Kanellopoulos, T. Shahroodi *et al.*, "Blockhammer: Preventing rowhammer at low cost by blacklisting rapidly-accessed dram rows," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 345–358.

TetrisLock: Quantum Circuit Split Compilation with Interlocking Patterns

Qian Wang*, Jayden John*, Ben Dong*, Yuntao Liu†
*Department of Electrical Engineering, University of California, Merced, CA, USA
†Department of Electrical and Computer Engineering, Lehigh University, PA, USA
qianwang@ucmerced.edu, jjohn92@ucmerced.edu, cdong12@ucmerced.edu, yule24@lehigh.edu

***Abstract*—In quantum computing, quantum circuits are fundamental representations of quantum algorithms, which are compiled into executable functions for quantum solutions. Quantum compilers transform algorithmic quantum circuits into one compatible with target quantum computers, bridging quantum software and hardware. However, untrusted quantum compilers pose significant risks. They can lead to the theft of quantum circuit designs and compromise sensitive intellectual property (IP). In this paper, we propose TetrisLock, a split compilation method for quantum circuit obfuscation that uses an interlocking splitting pattern to effectively protect IP with minimal resource overhead. Our approach divides the quantum circuit into two interdependent segments, ensuring that reconstructing the original circuit functionality is possible only by combining both segments and eliminating redundancies. This method makes reverse engineering by an untrusted compiler unrealizable, as the original circuit is never fully shared with any single entity. Also, our approach eliminates the need for a trusted compiler to process the inserted random circuit, thereby relaxing the security requirements. Additionally, it defends against colluding attackers with mismatched numbers of qubits, while maintaining low overhead by preserving the original depth of the quantum circuit. We demonstrate our method by using established *RevLib* benchmarks, showing that it achieves a minimal impact on functional accuracy (less than 1%) while significantly reducing the likelihood of IP inference.**

I. Introduction

The advancement of quantum computers ushers a new age of computational power and methodologies. At the core of each quantum computer are qubits, which differ from classical bits by operating based on principles of quantum mechanics. Unlike classical bits, which are either 0 or 1, qubits can exist simultaneously in multiple states due to their property of superposition, a feature that gives quantum computers their unique and powerful capabilities. Quantum gates manipulate qubits much like logic gates manipulate bits in classical computing. Key quantum gates include the Hadamard, Pauli, controlled, and phase gates, which are essential for executing complex quantum algorithms. By combining these gates, quantum circuits are created, linking the software layer (such as compilers) to the hardware (quantum processors). Today, several cloud providers, including IBM Quantum [1], Amazon Braket [2], and Microsoft Azure [3], provide access to quantum computing resources. To run a quantum algorithm, users submit their designs to a quantum compiler, such as Qiskit [4], which optimizes and schedules the circuits for specific quantum hardware.

Although quantum computers have shown significant advancements, designing quantum circuits remains resource-intensive and requires many iterations to achieve optimal functionality. As a result, quantum circuit designs are highly valuable and are recognized as a critical IP [5], facing similar threats as the classical IC designs. This paper primarily focuses on the IP threats to quantum circuits during the compilation process. The compilation process transforms the original quantum circuit into one compatible and can be executed with the quantum computer. This includes substituting all gates with gates supported by the quantum computing hardware and adding swap gates so that the entanglement among physical qubits supports the multi-qubit gates in the circuit. In addition, various optimizations can be incorporated to improve the performance of compiled circuits. However, vulnerabilities exist in this process, as malicious compilers could potentially access and misuse the quantum designs, such as insertion of trojans [6], [7] and counterfeiting of quantum designs [8]. As these quantum circuits represent valuable IP [5], it is essential to protect them from unauthorized use and IP theft.

In IC design, split manufacturing is commonly used to protect against IP theft by ensuring that no single foundry has access to the complete design, thereby safeguarding the design's IP [9], [10]. Similarly, the "splitting" can be adapted for quantum circuits. To conceal information from quantum compilers, the circuit can be divided into two or more sub-circuits, which are then compiled separately using different quantum compilers that support the target quantum computer. In this way, none of the compilers will see the entire circuit. This is analogous to split manufacturing of classical IC chips which is a well-studied area of circuit IP protection [11].

In this paper, we propose a novel method to split quantum circuits for secure compilation and our paper has the contributions as follows,

- We propose TetrisLock, a novel split compilation method designed for untrusted compilers, aimed at safeguarding the design of the original quantum circuits. This approach eliminates the need for a trusted compiler in the process.
- We introduce a process to add random quantum gates that mask the correct function while ensuring the obfuscated circuit retains the same depth as the original, without adding any depth overhead.
- We propose to split the obfuscated quantum circuit with

an interlocking pattern, drastically increasing the attack complexity of colluding untrusted compilers.
- Experimental results on the *RevLib* benchmark set demonstrate that TetrisLock achieves 0% depth increase, a 20% gate count increase (with a total of 1–4 gates inserted), and less than 1% accuracy loss for the obfuscated circuits.

II. Background & Related Work

A. Split Manufacturing for Electronic Circuits

Split manufacturing of a silicon chip separates the fabrication process across multiple foundries. Hence, no foundry has the full picture of the design, and the design IP is protected. This is important because most IC design companies do not own fabs and have to outsource the tape-out process, which puts the IP in the chip designs at risk. Hence, the primary security goal of split manufacturing is to prevent any foundry from inferring the unseen portion of the design from the portion that they are presented with.

There are multiple ways to split chip designs given the manufacturing technology. For ordinary silicon chips, the manufacturing can be split into a front end of line (FEOL) process and a back end of line (BEOL) process, where the FEOL primarily handles transistors and the BEOL handles the metal layers for interconnection. The FEOL can be done in an untrusted fab with advanced technology node. Since the BEOL usually does not require a technology node as advanced as the FEOL, it can be done in a trusted fab, hiding the interconnection among transistors to the untrusted fab thus preserving the IP in the chip design [11]. The feasibility of split manufacturing has been demonstrated in multiple research articles [12]–[14]. For example, in heterogeneously integrated 3D chip designs, it is often possible to split them into a "control" layer that does not need very advanced technology nodes and can be manufactured by a trusted foundry, and a "computation" layer that requires advanced technology nodes offered by an untrusted external foundry [15].

B. A Primer on Quantum Computing

This section provides an overview of the fundamentals of quantum computing, detailing the essential components such as qubits, quantum gates, and quantum circuits.

1) Qubits: The quantum bit (a.k.a. qubit) is the fundamental building block of quantum computing and is conceptually similar to the classical computing bit. A qubit has two basis states, denoted by the bracket notation as $|0\rangle$ and $|1\rangle$. The basis states for one qubit can be expressed as two-dimensional vectors, e.g., $|0\rangle = [1, 0]^T$ and $|1\rangle = [0, 1]^T$. As a result, the state $|\psi\rangle$ above can be written as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = [\alpha, \beta]^T$. For multi-qubit states, a similar situation exists. More specifically, there are 2^n basis states in the space of n-qubit states, ranging from $|0 \ldots 0\rangle$ to $|1 \ldots 1\rangle$, and a n-qubit state $|\psi\rangle$ can be expressed by:

$$|\psi\rangle = \sum_{i=0}^{2^n-1} a_i\,|i\rangle$$

where $\sum_{i=0}^{2^n-1} |a_i|^2 = 1$.

2) Quantum Gates: Similar to classical logic gates, the fundamental actions at the logic level in quantum computing are performed by quantum gates. These gates carry out unitary operations, which transform the states of input qubits. Quantum algorithms are composed of sequences of these quantum gates, designed to evolve input qubits into desired quantum states. For example, A quantum gate U must satisfy the equations $UU^\dagger = U^\dagger U = I$ ($U^\dagger$ is the conjugate transpose of U), meaning that a quantum gate must be a unitary operation. A quantum gate U operating on a qubit $|\psi\rangle$ can be written down as $|\psi\rangle \rightarrow U\,|\psi\rangle$. Single qubit gates operate on a single qubit and are analogous to the elementary logic gates in classical computing, such as the NOT gate. These gates manipulate a qubit by rotating its state vector on the Bloch sphere, a geometric representation of the qubit's state. The most common single qubit gates include the Pauli gates (X, Y, Z), the Hadamard gate (H), and phase shift gates (S, T).

Multi-qubit gates are essential in quantum computing because they enable interactions between qubits, facilitating complex operations in quantum algorithms. They manipulate the states of two or more qubits simultaneously, creating entanglement, a uniquely quantum phenomenon where qubit states become interdependent. For example, the Controlled-NOT (CNOT) gate operates on two qubits: a control qubit and a target qubit. The control qubit would determine whether the operation is applied to the target qubit. The target qubit's state is flipped by the control qubit in the $|1\rangle$ state. Many quantum algorithms rely heavily on the CNOT gate for their implementation, especially in constructing entangled states and performing conditional operations.

3) Quantum Gate Reversibility: Quantum gate reversibility is a fundamental property of quantum computing, implying that the quantum operations could be canceled out by applying the computation in reverse order. Unlike classical logic gates, which can lose information (e.g., an AND gate reduces two input bits to one output bit), quantum gates are inherently reversible, meaning that their operations can be undone to recover the original state of the system. This reversibility arises from the unitary nature of quantum gates. For example, if a quantum circuit applies a gate U followed by another gate V, the reversal of the circuit involves applying $V^\dagger$ followed by $U^\dagger$. This property is critical to the random gate insertion technique that enhances the security for split compilation.

4) Quantum Circuits: A quantum circuit is a sequence of quantum gates arranged to perform a computation on one or more qubits. It typically starts with qubits initialized in a known state (e.g., $|0\rangle$), followed by the application of single and multiple qubit gates. The circuit can be visualized as a timeline with wires representing qubits and gate operations along them.

5) Quantum Circuit Compilation: Quantum circuit compilation transforms a circuit representing a quantum algorithm into a hardware-executable circuit. This process adapts circuits to the specific topology and error characteristics of quantum hardware, similar to compiling classical computer programs but under unique constraints like superposition and entangle-

ment. Compilation ensures efficient and accurate execution of quantum algorithms, bridging the gap between algorithm designs and physical implementation on quantum processors.

C. Related Work and Limitations

Multiple avenues have been explored by researchers to secure quantum circuits against third-party compilers. These include random reversible circuit insertion [16]–[18], locking the quantum circuit by adding extra key qubits [19], [20], and splitting the quantum circuit and compiling each portion in a different compiler [21]. In [16] and [18], a random reversible quantum circuit was generated and inserted at the front, middle, or end of the original quantum circuit to achieve obfuscation. To restore the circuit's functionality after compilation, the reverse of the random circuit was added in the same location. A similar strategy was employed in [17]. However, the primary limitation of these approaches is that the topology of the original circuit remains fully exposed, making it possible for an adversary to identify the boundary between the original circuit and the inserted random portion. The technique proposed in [19] adopts an approach akin to classical logic locking, where the circuit's functionality is controlled by multiple key bits. For each key bit, an additional qubit is introduced into the quantum circuit. While this method adds a layer of security, it is impractical for real-world applications due to the high cost of qubits and the limited number of qubits available in current quantum computers.

Compared to random circuit insertion and quantum circuit locking, split compilation has some unique advantages. Since each compiler only sees the part of the circuit that it compiles, other parts of the quantum circuit are completely hidden from the compiler. Saki *et. al.* proposed to split the quantum circuit into two or more cascading sections and compile separately [21]. To ensure compatibility with quantum computers and resist attacks when the same compiler is used for more than one compilation, they add a swap network between each pair of split-compiled quantum circuits. This approach may still suffer from a brute-force attack when the same compiler compiles two adjacent parts of the circuit. As current quantum computers have a very limited number of qubits, it is still feasible for the adversary to try all permutations of the qubit mapping between the two split circuits. In this work, we present a novel split compilation technique that uses dummy gates and un-matched qubit splitting to mitigate such brute-force attacks.

III. Threat Models

In this paper, we consider the third-party quantum compilers as potential adversaries. These third-party quantum compilers offer numerous benefits, such as support for multiple quantum computing platforms, advanced optimization, error correction, etc. [22], [23]. Popular third-party quantum compilers include Qulic [22], TKET [24], etc., and some hardware-specific compilers, like IBM's Qiskit [4] and Google's Cirq [25], also support other platforms that can function as a third-party compiler.[1] Although third-party compilers offer flexibility, they also pose serious threats to the IP in quantum circuits. During the compilation process, the quantum circuit design is fully exposed to the compiler, leaving its intellectual property vulnerable to infringement or counterfeiting [26]. An adversary with unauthorized access can replicate the original design and duplicate the circuit's functionality. These threat models are illustrated in Figure 1. The threat model considered in this paper is consistent with those used in previous quantum adversaries approaches [16]–[19], [21]. As far as split compilation is concerned, we also consider the case where the split circuits are compiled using the same compiler, or using colluding compilers.

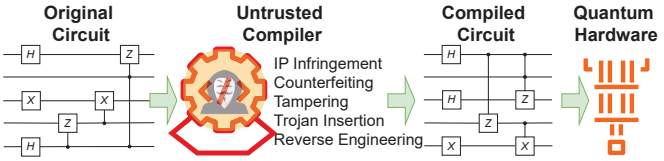


Fig. 1: Illustration of the untrusted compiler threat model. During the compile process, the circuit's IP is exposed.

IV. Obfuscation for Quantum Circuits

Our defense strategy focuses on limiting IP exposure through Quantum circuit obfuscation. We first apply an interlocking method to split the circuit for compilers to process. Since the untrusted compiler does not have the whole piece of the design, the correct functionality of the quantum circuit remains hidden from the compiler. After compilation, a "de-obfuscation" process is applied to the circuits, restoring the functionality accordingly. This flow is shown in Figure 2.

A. Interlocking Obfuscation

TetrisLock, our proposed interlocking obfuscation method, consists of multiple steps to obscure the circuit design before it is sent to the untrusted compiler, as illustrated in Figure 2. This process involves segmenting the circuit into interdependent parts, each of which lacks the full functionality of the original circuit before splitting. By obfuscating and distributing these parts across different compilers, we ensure that no single compiler has access to the complete circuit logic, effectively safeguarding the design's integrity and intellectual property.

To facilitate the obfuscation while defending against the collude attacks, we generate a random circuit R to insert before and cover the functionality of the original circuit C. To avoid significant increases in circuit depth or gate count, we employ an algorithm to identify empty slots in the original circuits at each layer. The random circuits are then inserted into these empty slots, as detailed in Algorithm 1.

To reserve the function of the original circuit, we also insert the inverse of the random circuit, denoted by $R^\dagger$ or R^{-1}. This insertion maintains the original circuit's functionality due to the reversibility property of quantum circuits discussed in

[1]These compilers are only listed as examples to demonstrate the popularity of third-party quantum compilers. We by no means suggest that these compilers are malicious.

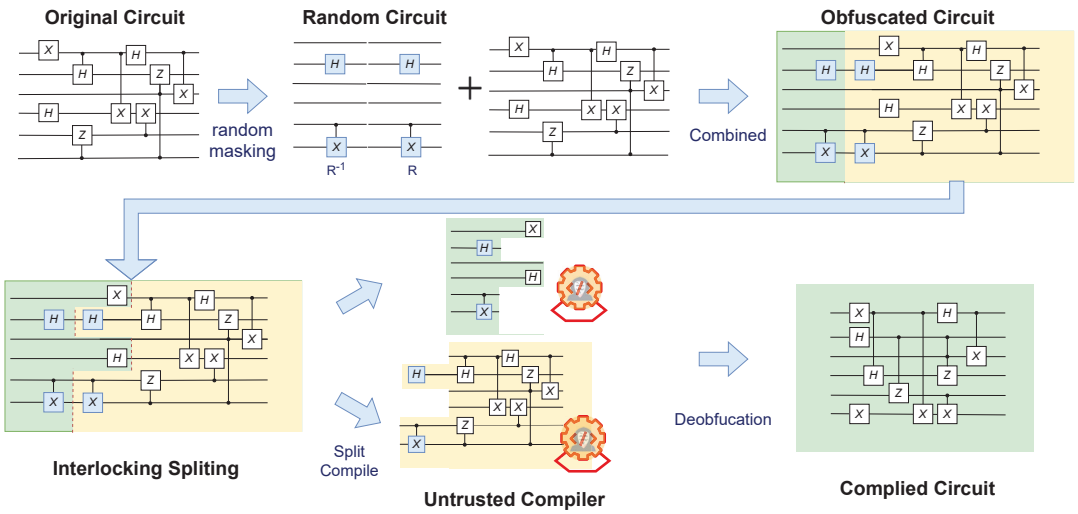


Fig. 2: The TetrisLock framework begins by appending the original circuit C with a random circuit R, followed by splitting it based on interlocking patterns. Untrusted compiler only has partial of the original circuit. After compilation, the segments are recombined during the de-obfuscation process, fully restoring functionality.

Sec.II-B3. Specifically, the reversibility of quantum circuits allows us to implement the inverse by applying the conjugate transpose of each gate in reverse order. For example, if the circuit applies a gate sequence A followed by B as $A \cdot B$, the reversed circuit will apply the gates in the order B^{-1} followed by A^{-1}, as $B^{-1}A^{-1}$ where $^{-1}$ denotes the conjugate transpose (adjoint) of each gate matrix. TetrisLock, which adds R and R^{-1} before the original circuits, will not alter the function but introduce additional gates as masking. It ensures the obfuscated circuit appears substantially different while retaining the original functionality to safeguard the design from unauthorized access during the compilation process. Thus, the total obfuscated circuit can be represented as $R^{-1}RC$. Ideally, the inverse circuit R^{-1} would cancel out the effect of R, ensuring that the modified circuit $C' = R^{-1}RC$ maintains the exact functionality of the original circuit C. To defend against the potential attacks to extract the original circuit C from RC, we propose an interlocking splitting method between R and C. In our approach, portions of the original circuit denoted as C_l are integrated within the random inverse circuit R^{-1}, as illustrated in Figure 2. Instead of a straight partition, we implement an interlocking boundary to enhance the obfuscation properties of the circuit and thus increase the security level of the process. In Figure 3, we split the same obfuscated quantum circuit as in Figure 2 with a different interlocking pattern. By embedding interlocking gates within the boundary, we create a circuit structure in which, even if an attacker were to isolate the portion of $\tilde{C}_r$, they would still lack access to the complete original circuit C. This is because crucial circuit logic segments are now interwoven within $R|C_l$. Consequently, this interlocking strategy significantly reduces the risk of reverse-engineering the original circuit, ensuring the design remains secure.

B. De-Obfuscation Process

Our de-obfuscation process combines two separately compiled segments of the circuit as $R^{-1}C_l$ and RC_r together.

Algorithm 1: Random gate insertion into empty positions

```
Data: C: quantum circuit, empty_pos: List to store empty
      positions
Input: total_qubits, gate_limit
/* Step1:Get empty positions */
Convert circuit to DAG representation and Extract layers
for each layer in layers do
    Get operations in the current layer
    Initialize empty set for used qubits
    for each operation in layer do
        used_qubits ← GetQubitIndices(operation)
    empty_positions ← sorted(list(all_qubits \ used_qubits))
/* Step2: randomly insertion into circuits */
foreach column ∈ quantum circuits do
    if added_gates > gate_limit then
        Break
    available_qubits ← available_qubits ∩ empty_positions
    if Random(0,1) < 0.5 and find(q1,q2) == true then
        AddCXGate(circuit, q1, q2)  // insert CX gate
        available_qubits ← available_qubits \{q1,q2}
        added_gates ← added_gates +1
    else
        // insert X gate
        random_pos ← RandomChoice(available_qubits)
        AddXGate(circuit, random_pos)
        available_qubits ← available_qubits \{random_pos}
        added_gates ← added_gates +1
return C: quantum circuit
```

Since the original circuit was modified by adding a random circuit R along with its inverse R^{-1}, the effects of R cancel out, effectively restoring the original functionality of circuit C. This combined circuit preserves the original function without revealing it to the separated untrusted compilers.

C. Improvements over Prior Work

TetrisLock mitigates significant security vulnerabilities identified in prior work on quantum circuit split compilation [21] and random circuit insertion for compilation [16]. The primary vulnerability of [21] is colluding compilers that could

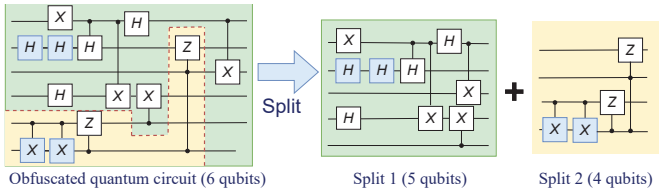


Fig. 3: Another way to split the same obfuscated circuit as in Figure 2 with an interlocking pattern. The two split circuits have different numbers of qubits and not every original qubit needs to be split.

collectively reconstruct the original circuit from split segments. Specifically, the adversary can execute an attack by identifying two split circuits with the same number of qubits and subsequently matching the corresponding qubit across both segments. Hence, the complexity of one compiler to find the original circuit is $k_n n!$, where n is the number of qubits, and k_n is the number of segments for the compiled quantum circuits with n qubits from the other compiler. In contrast, as shown in Figure 3, TetrisLock enables the split circuit to have different amounts of qubits. This means that the piece of the circuit from the other compiler can consist of any number of qubits and does not need to be equal to the first piece. The attack complexity is then given by

$$\sum_{i=1}^{n_{max}} k_i \sum_{j=0}^{\min(n,i)} C_n^j C_i^j j! \tag{1}$$

let n_{max} denote the maximum number of qubits supported by the target quantum computer, i represent the number of qubits in the other split circuit, and j is the number of qubits that need to be connected. Equation 1 calculates the number of possible qubit mapping by considering the enumeration process across each candidate split circuit k_i, each possible number of connected qubits C_n^i, each combination of connected qubits from both segments C_n^j and each possible mapping between them $j!$. Consequently, the attack complexity of [21] constitutes only a minor fraction of the total attack complexity in our approach. Furthermore, the original circuit is structurally obfuscated through the random circuit R. This additional layer of masking effectively protects the original design C exposed to the untrusted compiler.

The random circuit insertion approach of [16] suggests placing a boundary between R^{-1} and R as $R^{-1}|RC$. This configuration ensured that only the right segment, RC, is processed by the untrusted compiler. However, despite the addition of random circuit R, an attacker might still be able to identify the boundary between R and C. This identification could enable the attacker to recognize and isolate the original circuit C from the one RC sent to the compiler. Our approach has the following advantages: (1) No need for a trusted compiler to compile the inserted random circuit. (2) No single compiler have access to the entire original circuit, and (3) Low overhead, as our approach does not increase the overall depth of the quantum circuit.

V. Experiments Evaluation

A. Experimental Setup

We conducted our experiments using the IBM Qiskit framework to compile and simulate quantum circuits. We utilized benchmark circuits from the *RevLib* benchmarks [27] for our experiments, which has been widely utilized in prior work on quantum circuit compilation. These benchmark circuits encompass a diverse set of gate operations, with the number of gates ranging from 4 to 32 and qubit sizes varying across 4, 5, 7, 10, and 12 qubits. To replicate realistic simulation conditions, we employed the *FakeValencia* backend from Qiskit [4], which incorporates the noise model of the actual *ibmq-valencia* device. All simulations were performed with 1,000 shots to generate statistically significant results. Both the original and the obfuscated circuits were simulated using the same backend, ensuring that any differences observed are attributable to the mechanism rather than variations in the simulation environment.

As described in Section IV-A, we insert a random circuit, composed of a specific set of quantum gates, at the beginning of the original circuit. We strategically selected gate types for insertion based on the operations present in the benchmarks. For instance, in the *RevLib* benchmarks, which predominantly feature arithmetic operations such as adders, ALUs, counters, and comparators, we primarily use NOT (X) and CNOT (CX) gates to obfuscate the circuit and camouflage the circuits from potential attackers. For other types of circuits, such as those implementing Grover's algorithm, we opted to insert Hadamard (H) gates. This tailored approach ensures that the obfuscating mechanism is appropriately aligned with the nature of the operations in each circuit, reducing structural leakage and enhancing the effectiveness of the obfuscation.

B. Metrics for Evaluation

Total Variation Distance (TVD) is a metric used to quantify the distance between two probability distributions. This metric is particularly suitable for quantum circuits measurement because unlike traditional computers, which produce deterministic results, the output of a quantum computing circuit is inherently probabilistic distributions. For example, the output of a 1-bit circuit simulation with noise can be represented as a distribution, such as {"0": 95, "1": 5}, based on 100 shots. In this context, TVD measures the discrepancy between the output distributions of the correct (original) circuit and the obfuscated circuit. This discrepancy reflects the effectiveness of the obfuscation in concealing the correct output. It is calculated as the sum of absolute differences between the counts of each outcome in the two distributions, normalized by the total number of shots. The formula for TVD is:

$$TVD = \frac{\sum_{i=0}^{2^b-1} |y_{i,orig} - y_{i,alter}|}{2N} \tag{2}$$

Where N represents the total number of shots in this run, b represents the number of output qubits, resulting in 2^b possible output types. $y_{i,alter}$ and $y_{i,orig}$ represent the count of value i in the altered and original quantum circuits respectively.

Circuit	Depth	Depth Obfuscated	Gate Count	Gate Obfuscated	Gate change(%)	Accuracy	Accuracy restored	Accuracy change(%)
mini_ALU	8	8	9	11	22.2%	0.974	0.974	0.06%
4mod5	5	5	6	8	33.3%	0.973	0.967	0.6%
1-bit adder	5	5	7	8	14.2%	0.976	0.976	0.12%
4gt11	13	13	13	15	15.4%	0.986	0.983	0.30%
4gt13	4	4	4	6.7	67.5%	0.976	0.977	0.95%
rd53	16	16	19	22	15.7%	0.88	0.869	1.09%
rd73	13	13	23	26	13.0%	0.892	0.884	0.73%
rd84	15	15	32	36	12.5%	0.867	0.863	0.42%

TABLE I: Comparison of circuit parameters: depth, count, accuracy, and fidelity change before and after alterations, data shown here are the averages of 20 iterations. The original circuit is from the *RevLib*.

C. Result Analysis

In this section, we present our experimental results, starting with the outcomes from the *RevLib* benchmarks simulated using the Qiskit backend. These results encompass both 1-bit and multi-bit circuit scenarios. Figure 4 illustrates a comparison of TVD values across different circuits for the obfuscated case and restored case. TVD is calculated as the variation distance with the theoretical output. For instance, in the case of a 1-bit Adder with 100 shots, we use the outcome as {"0": 100, "1": 0 } as the reference to compare relative distance.

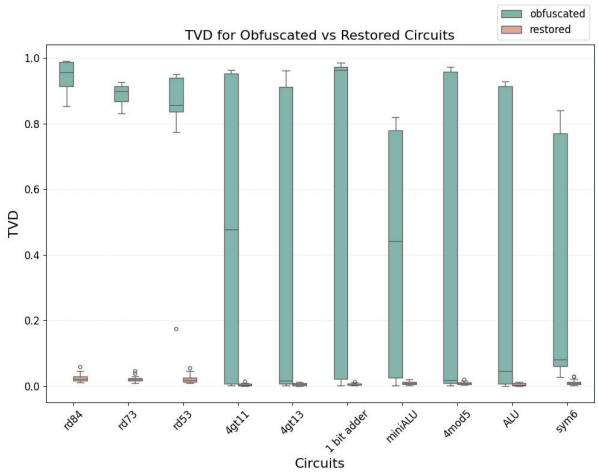


Fig. 4: Distribution of Total Variation Distance (TVD) of benchmark circuits: TVD of obfuscated circuit and restored circuit are calculated and shown respectively. Selected circuits are simulated using Qiskit and FakeValencia backend, incorporating noise into the simulation

First, significant variations in TVD values are observed for the obfuscated circuits, indicating functional alterations caused by the insertion of the random circuit. For circuits such as rd84, rd73, and rd53, the TVD values approach 1, indicating significant changes in the output distribution. This occurs because these circuits produce multi-bit outputs and are relatively large and deep, which provide ample opportunities for the insertion of random gates. More insertion of random gates results in more flips in the output. In contrast, smaller circuits with 1-bit output, such as the 1-bit adder (4-qubit) and 4gt13 (4-qubit) have less significant changes in TVD value. This is because their simpler structure provides limited space for inserting random gates compared to more complex circuits. For the restored case, R^{-1} is applied to the obfuscated circuit RC, effectively canceling the disturbance introduced by random circuit R. TVD values in this case are consistently close to zero, confirming that the circuits retain the same functionality as the original. Overall, Figure 4 shows that the obfuscated circuits differ significantly from the original circuits in TVD values. The consistent TVD values across benchmarks indicate that our TerisLock method uniformly impacts all circuits.

D. Cost and Overhead Analysis

1) Gate Counts and Circuit Depth: Results regarding gate counts and circuit depth across various circuits are presented in Table I, revealing several trends in how alterations impact circuit complexity and performance. First, we applied a random insertion algorithm to ensure that the inserted gates occupy existing empty slots of the circuit layers, keeping the circuit depth unchanged after obfuscation. This demonstrates the effectiveness of the obfuscation method in preserving depth while covering the circuits' functions. The number of gates inserted across the circuit benchmarks ranges from 2 to 4, resulting in an average 20% increase in the total gate count. However, for larger and deeper circuits, the relative impact of these additional gates is minimal, as they represent a small proportion of the original gate count.

2) Fidelity of Unlocked Circuits: In this section, we evaluate the performance of circuits after their restoration following the split compilation process. Splitting a circuit into two parts and introducing additional gates and layers can affect its accuracy. To assess this, we measured the accuracy (the ratio of correct outcomes to the total number of shots) both before and after the circuits were restored. As shown in Table I, most circuits exhibited only a slight decrease in accuracy post-restoration, with fidelity changes typically remaining below 1%. Overall, while these modifications increase the circuit's complexity, their impact on performance is negligible. This demonstrates that the circuits maintain their robustness and reliability even after undergoing the introduced alterations.

VI. Conclusion

In this paper, we introduce TetrisLock, a novel split compilation technique that enhances quantum circuit security by inserting random quantum gates and splitting the circuit using an interlocking pattern. We demonstrate that this approach has a significant advantage over existing split compilation and random gate insertion-based methods. Our experiments across various quantum circuits show that TetrisLock achieves a high functional corruption when obfuscated and has minimal impact on accuracy when the split circuits are correctly recombined. Hence, TetrisLock is a viable solution for protecting quantum circuit IP through split compilation.

REFERENCES

[1] J. Chow, O. Dial, and J. Gambetta, "Ibm quantum breaks the 100-qubit processor barrier," *IBM Research Blog*, vol. 2, 2021.

[2] C. Gonzalez, "Cloud based qc with amazon braket," *Digitale Welt*, vol. 5, no. 2, pp. 14–17, 2021.

[3] K. Prateek and S. Maity, "Quantum programming on azure quantum—an open source tool for quantum developers," in *Quantum Computing: A Shift from Bits to Qubits*. Springer, 2023, pp. 283–309.

[4] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with Qiskit," 2024.

[5] M. Aboy, T. Minssen, and M. Kop, "Mapping the patent landscape of quantum technologies: Patenting trends, innovation and policy implications," *IIC-International Review of Intellectual Property and Competition Law*, vol. 53, no. 6, pp. 853–882, 2022.

[6] S. Das and S. Ghosh, "Trojannet: Detecting trojans in quantum circuits using machine learning," *arXiv preprint arXiv:2306.16701*, 2023.

[7] R. Roy, S. Das, and S. Ghosh, "Hardware trojans in quantum circuits, their impacts, and defense," in *2024 25th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2024, pp. 1–8.

[8] M. Yang, X. Guo, and L. Jiang, "Multi-stage watermarking for quantum circuits," *arXiv preprint arXiv:2404.18038*, 2024.

[9] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 1259–1264.

[10] Y. Xie, C. Bao, and A. Srivastava, "Security-aware design flow for 2.5 d ic technology," in *Proceedings of the 5th International Workshop on Trustworthy Embedded Devices*, 2015, pp. 31–38.

[11] T. D. Perez and S. Pagliarini, "A survey on split manufacturing: Attacks, defenses, and challenges," *IEEE Access*, vol. 8, pp. 184 013–184 035, 2020.

[12] K. Vaidyanathan, B. P. Das, E. Sumbul, R. Liu, and L. Pileggi, "Building trusted ics using split fabrication," in *2014 IEEE international symposium on hardware-oriented security and trust (HOST)*. IEEE, 2014, pp. 1–6.

[13] K. Vaidyanathan, R. Liu, E. Sumbul, Q. Zhu, F. Franchetti, and L. Pileggi, "Efficient and secure intellectual property (ip) design with split fabrication," in *2014 IEEE international symposium on hardware-oriented security and trust (HOST)*. IEEE, 2014, pp. 13–18.

[14] B. Hill, R. Karmazin, C. T. O. Otero, J. Tse, and R. Manohar, "A split-foundry asynchronous fpga," in *Proceedings of the IEEE 2013 Custom Integrated Circuits Conference*. IEEE, 2013, pp. 1–4.

[15] J. Valamehr, T. Sherwood, R. Kastner, D. Marangoni-Simonsen, T. Huffmire, C. Irvine, and T. Levin, "A 3-d split manufacturing approach to trustworthy system development," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, pp. 611–615, 2013.

[16] S. Das and S. Ghosh, "Randomized reversible gate-based obfuscation for secured compilation of quantum circuit," *arXiv preprint arXiv:2305.01133*, 2023.

[17] A. Suresh, A. A. Saki, M. Alam, R. Onur Topaloglu, and S. Ghosh, "Short paper: A quantum circuit obfuscation methodology for security and privacy," in *Proceedings of the 10th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2021, pp. 1–5.

[18] S. Das and S. Ghosh, "Secure quantum circuit compilation methodology for untrusted compilers," 2024.

[19] R. O. Topaloglu, "Quantum logic locking for security," *MDPI Multidisciplinary Scientific Journal*, vol. 6, no. 3, pp. 411–420, 2023.

[20] Y. Liu, J. John, and Q. Wang, "E-loq: Enhanced locking for quantum circuit ip protection," 2025. [Online]. Available: https://arxiv.org/abs/2412.17101

[21] A. A. Saki, A. Suresh, R. O. Topaloglu, and S. Ghosh, "Split compilation for security of quantum circuits," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–7.

[22] R. S. Smith, E. C. Peterson, M. G. Skilbeck, and E. J. Davis, "An open-source, industrial-strength optimizing compiler for quantum programs," *Quantum Science and Technology*, vol. 5, no. 4, p. 044001, 2020.

[23] M. Salm, J. Barzen, F. Leymann, B. Weder, and K. Wild, "Automating the comparison of quantum compilers for quantum circuits," in *Symposium and Summer School on Service-Oriented Computing*. Springer, 2021, pp. 64–80.

[24] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, and R. Duncan, "t— ket¿: a retargetable compiler for nisq devices," *Quantum Science and Technology*, vol. 6, no. 1, p. 014003, 2020.

[25] A. Hancock, A. Garcia, J. Shedenhelm, J. Cowen, and C. Carey, "Cirq: A python framework for creating, editing, and invoking quantum circuits."

[26] S. Ghosh, S. Upadhyay, and A. A. Saki, "A primer on security of quantum computing," *arXiv preprint arXiv:2305.02505*, 2023.

[27] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in *38th International Symposium on Multiple Valued Logic (ismvl 2008)*. IEEE, 2008, pp. 220–225.

NN-AdderNet: Nonnegative and Sparse Weight Optimization Towards Ultra-Low Bitwidth AdderNet Quantization and Compression

Yunxiang Zhang[1*], Gengchen Sun[2*], Lizhi Fang[2], Biao Sun[2], and Wenfeng Zhao[1]
[1]Department of Electrical and Computer Engineering, Binghamton University, Binghamton, New York, USA
[2]School of Electrical and Information Engineering, Tianjin University, Tianjin, China
[*]These authors contributed equally to this work.
sunbiao@tju.edu.cn, wzhao@binghamton.edu

***Abstract*—Emerging efficient deep neural network (DNN) models, such as AdderNet, have shown great promise in significantly improving hardware efficiency compared to traditional convolutional neural networks (CNNs). However, achieving low bitwidth quantization and effective model compression remains a major challenge. To this end, we introduce Nonnegative AdderNet (NN-AdderNet), a quantization- and compression-friendly AdderNet variant that enables model compression down to 4 bits or even lower. We begin by proposing an equivalent transformation of the sum-of-absolute-difference (SAD) kernel in AdderNet, which allows for the formulation of nonnegative weights. This transformation effectively eliminates the need for a sign bit, thus saving 1 bit per weight. Next, we propose to exploit the dual-sparsity pattern in the weights of the activation-oriented NN-AdderNet quantized model. This inherent sparsity enhances the lossless compression performance over NN-AdderNet. Experimental results show that NN-AdderNet can achieve an average compressed weight bitwidth down to 4 bits or even lower, while achieving negligible accuracy loss as compared to full-precision AdderNet models. Such benefits are further illustrated with hardware-level energy and latency improvements in designing DNN inference accelerators. Consequently, the NN-AdderNet model exhibits both algorithmic and hardware efficiency, thus making it a promising candidate for resource-limited applications.**

I. Introduction

Deep learning algorithms, particularly convolutional neural networks (CNNs), have achieved remarkable success in various computer vision and machine intelligence tasks [1]–[3]. However, the growing complexity of these networks, driven by the underlying deeper and wider neural network architecture and large-scale computations involving multiplication-accumulation (MAC) operations and memory accesses. To expedite computation at hardware level, current deep learning accelerators utilize massively-parallel processing elements, which has posed significant challenges for deployment on resource-constrained edge devices. [4], [5] Consequently, extensive research efforts have been dedicated to minimizing the computation and energy consumption of CNNs. Hardware-efficient CNN models, such as bottleneck architecture and point/depth-wise convolution [3], [6], [7], have been devised to amortize the computational cost. Model pruning and quantization have been widely explored [8]–[11] to reduce the MAC operations and the arithmetic/memory bitwidth, such as INT8 [12] and INT4 [13]. Nevertheless, the MAC hardware is still inevitable for the aforementioned techniques.

To address the challenge, AdderNet [14] has emerged as a promising alternative to CNNs. It replaces the costly MAC operations with hardware-efficient sum-of-absolute-difference (SAD) operations. AdderNet achieves substantial hardware resources and power consumption reduction [15]–[21] while maintaining competitive accuracy compared to CNNs.

Despite promising, AdderNet implementation remains challenging at hardware level, especially low-bitwidth quantization and model compression [22]. Existing quantization strategies for CNNs cannot be directly applied to AdderNet due to its unique additive operation. The SAD operations in AdderNet require shared quantization scale factor between weights and activations, which complicates the quantization algorithms and hardware design. Several AdderNet-based accelerators have been demonstrated on FPGA [15]–[17] and Compute-in-Memory ASIC [20], [21] to improve the efficiency of AdderNet-based accelerators. However, these designs are mostly limited to high model quantization bitwidth (INT16), and still sub-optimal in resource utilization. WSQ-AdderNet [18] employs weight standardization technique to achieve INT8 model quantization and the collaborative packaging of parallel SAD operations in DSP-LUT hardware. Nevertheless, WSQ leads to slight model accuracy degradation. In [23], a new quantization method was introduced by employing group-shared scaling and dynamic clipping to handle weights and activations. By clustering weights and applying custom clipping techniques, this approach indeed improves the performance of low-bit AdderNet models, however, it substantially increases the computational and hardware implementation complexities. Alternatively, an activation-oriented quantization (AOQ) technique was proposed [19], aiming at reducing the bitwidth of weights and activations to INT6 with minimal model accuracy degradation. A fused-bias removal strategy was also introduced to minimize on-chip memory requirements by eliminating redundant bias accumulation.

Despite the consistent efforts, AdderNet models are still limited to INT6 quantization. Moreover, the signed Laplacian distribution of weights render it challenging to achieve high compression ratio using lossless entropy coding scheme. To further optimize AdderNet quantization and compression, we propose a novel nonnegative AdderNet (NN-AdderNet) framework. NN-AdderNet further incorporates sparse weight optimization based on a lossless compression scheme, enabling substantial improvement in model compression performances and ultimately can be exemplified with accelerator hardware level benefits. Specifically, we reformulate the signed SAD of AdderNet into nonnegative SAD operations. Combined with activation-oriented quantization, this results in a novel dual-sparsity weight distribution in NN-AdderNet. When incorporating Huffman encoding, our approach ensures sub-INT4

quantization across a wide collection of AdderNet-based DNN models. The main contributions are summarized as follows:

- We introduce a novel nonnegative AdderNet formulation that leads to unsigned SAD kernel, thereby facilitating model quantization with sign-bit savings;
- We propose an activation-oriented nonnegative AdderNet quantization that can promote a novel dual-sparsity in the nonnegative weights, which ultimately enhances the entropy coder performance;
- We demonstrate substantial model compression benefits of NN-AdderNet, with 1.5∼1.67× model size reduction on uncompressed 8-bit models, thereby leading to sub-4bit compressed models with merely 0.3%∼1.6% model accuracy drop;
- We show significant NN-AdderNet accelerator latency improvement of up to 34.8%∼38.6% over CNN accelerators for sub-4bit compressed models, and 13%∼16.2% latency improvement for sub-3bit compressed models.

This paper is organized as follows. Section II introduces the background and motivation. Section III details proposed Nonnegative AdderNet. Section IV uncovers the dual-sparsity profiles in the NN-AdderNet weights. Section V presents the experimental results. Section VI concludes the paper.

II. Background and Motivation

A. Preliminaries on AdderNet

In a conventional CNN model, for a given input feature map $X \in \mathbb{R}^{h\times w\times c_{\text{in}}}$ and weight $W \in \mathbb{R}^{k\times k\times c_{\text{in}}\times c_{\text{out}}}$, the convolutional operation is defined as:

$$Y(m,n,c_{\text{out}}) = \sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{c_{\text{in}}} X(m+i,n+j,l)\times W(i,j,l,c_{\text{in}}), \quad (1)$$

where h and w are the height and width of the input, $k \times k$ is the size of the filter kernel, c_{in} and c_{out} are the number of input and output channels, m and n are the height and width of the output, respectively. In AdderNet [14], convolution operations are replaced by SAD operations, [24]–[26] aiming to reduce the computational overhead of the expensive MAC operations:

$$Y(m,n,c_{\text{out}}) = -\sum_{i=1}^{k}\sum_{j=1}^{k}\sum_{l=1}^{c_{\text{in}}} |X(m+i,n+j,l) - W(i,j,l,c_{\text{in}})|, \quad (2)$$

where $|\cdot|$ is the absolute function, which involves only subtraction (addition equivalent), multiplexer, and negate. In the following part of the paper, we will simplify the subscripts and write the SAD operation in a more compact form as

$$Y = -\sum |X - W|. \quad (3)$$

B. AdderNet Quantization Challenge

A significant number of early-stage AdderNet quantization studies are limited to INT16. This limitation arises as AdderNet presents fundamentally different quantization requirements compared to convolutional kernels. Existing CNN model quantization techniques are not directly applicable to AdderNet. In particular, the exchange law does not hold in the SAD operation, meaning that independent scaling factors for weights (s_w) and activations (s_x) cannot be used for AdderNet quantization, as is typically done in CNNs:

$$\text{SAD}(s_x X, s_w W) \neq (s_x s_w) \times \text{SAD}(X, W). \quad (4)$$

Therefore, AdderNet quantization requires the use of a shared scale s_{common} to quantize weights and activations. So, quantization and dequantization weights are defined as:

$$\text{SAD}(sX, sW) = s_{\text{common}} \times \text{SAD}(X, W) \quad (5)$$

Hence, due to the limitation of the shared scale factor during model quantization in AdderNet, existing quantization techniques do not perform well, which results in a higher bitwidth ($\geq$ 10 bits) of the SAD kernel compared to the convolutional kernel (6 bits) for the same inference accuracy. As a result, the potential hardware gains of AdderNet may diminish.The source of quantization error in AdderNet is mainly caused by the value difference between the weights and the input activations. The weights in AdderNet obey the Laplace distribution with a mean of zero, but the variance is much larger than Activations.

One approach to improve weight quantization is to reduce the weight dynamic range, as demonstrated in [18], as the dynamic range or distributions of the ReLU activations can barely be easily altered. WSQ-AdderNet [18], which employs weight normalization (WS) technique reduces the weight dynamic range by approximately a factor of 4, thereby achieving INT8 weight quantization. Nevertheless, this normalization results in a decrease in model accuracy even in full-precision training, with a drop of about 1% compared to baseline AdderNet. To achieve model quantization below INT8 while retaining model accuracy, AdderNet 2.0 [19] introduces an AOQ technique to achieves INT6 model quantization. This is accomplished by quantizing activations to 5 bits, while the weights are clipped to the range of -32 to 31, resulting in an effective 6-bit weight quantization. To ensure correct SAD operation, a weight bias is generated during weight clipping process, and the significant weight bias is handled by either adding it back to the feature map or fusing it into the batch normalization bias, as it represents a portion of the mean offset. However, further reducing the quantization bitwidth leads to increased activation quantization errors, causing a sharp drop in model accuracy.

III. Proposed Nonnegative AdderNet

In this section, we detail the mathematical foundations of nonnegative AdderNet.

A. Equivalent Transformation of SAD to Nonnegative SAD

Theorem 1. *Given an SAD operation with signed weights W, which has been trained to achieve high performance on a training set, and access to a representative subset of the input data, there exists a family of equivalent transformations that convert the signed SAD operation to a nonnegative SAD*

operation with nonnegative weights W_+, without any loss in performance. One typical transformation is:

$$\sum |X - W| = \sum |X - \mathrm{ReLU}(W)| + \sum |W_-|, \tag{6}$$

where W_- represents negative weights and ReLU denotes the rectified linear unit function. $\sum |W_-|$ represents the sum of the absolute values of all negative weights, which can be precomputed without incurring extra computation after model training.

Proof. In AdderNet, each layer's activation function is ReLU, meaning that the activations X are always nonnegative, whereas the weights W in the original AdderNet can be signed, taking positive, zero, or negative values. To better illustrate the relationships between X and W under different conditions, we can simplify the scenario by assuming that both X and W correspond to points on a sine wave, as shown in Fig. 1. The figure is divided into five regions, each representing a different relationship between X and W, including both magnitude and sign variations. These five regions encompass all possible relationships between X and W. For example, region 1 represents the case where both X and W are positive, with W being greater than X, while region 3 represents the case where X is positive and W is negative, in which case it is clear that X is greater than W. Based on the above regional division, an SAD operation can be decomposed into the summation of the results from the five regions as:

$$\begin{aligned}
\sum |X - W| &= \sum_{1} |X_1 - W_1| \\
&\quad + \sum_{2} |X_2 - W_2| + \sum_{3} |X_3 - W_3| \\
&\quad + \sum_{4} |X_4 - W_5| + \sum_{5} |X_5 - W_5| \\
&= \sum_{1} W_1 - \sum_{1} X_1 + \sum_{2} X_2 - \sum_{2} W_2 \\
&\quad + \sum_{3} X_3 - \sum_{3} W_3 + \sum_{4} W_4 - \sum_{5} W_5 \\
&= \sum_{1} W_1 - \sum_{1} X_1 + \sum_{2} X_2 - \sum_{2} W_2 + \sum_{4} W_4 \\
&\quad + \sum_{3} X_3 \\
&\quad + \left(- \sum_{3} W_3 - \sum_{5} W_5 \right) \\
&= \sum_{1,2,4} |X_{1,2,4} - W_{1,2,4}| + \sum_{3,5} |X_{3,5} - 0| \\
&\quad + \sum_{3,5} |W_{3,5}| \\
&= \sum |X_+ - W_+| + \sum |X_- - 0| + \sum |W_-| \\
&= \sum |X - \mathrm{ReLU}(W)| + \sum |W_-|,
\end{aligned} \tag{7}$$

where the subscript of the summation symbol indicates the region over which the summation is performed. W_+ and W_- represent the positive and negative weights, while X_+ and X_- are the corresponding activations, respectively. The $+$ under the summation symbol indicates that the summation is performed over the region where W is positive, while the $-$ under the summation symbol indicates that the summation is performed over the region where W is negative. □

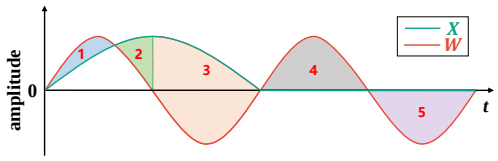


Fig. 1. Illustration of X and W with varying signs and amplitudes. The numbers in the figure represent the regions when X and W take different signs and amplitudes.

Remarks. (i) The theorem mentions a family of transformations since the transformation is not unique, that is, different nonnegative AdderNet formulations give rise to an exact transformation. In this paper, we propose the transformation in Eq. (6) because it is most conducive to quantization. Although other literature has introduced various transformations applicable to different scenarios, they are not suitable for either quantization or excessive computation [21]. (ii) A consequence of the exact transformation is that both signed AdderNet and nonnegative AdderNet have exactly the same performance on a sample-by-sample basis: if for a specific sample the prediction of the signed AdderNet is wrong then this is also the case for the nonnegative AdderNet, and vice versa.

B. Activation-Oriented Quantization for NN-AdderNet

For a q-bit quantization in Adder layer, the quantized weights can be obtained from the original floating-point weights based on a quantization step size, which typically results in a bitwidth much larger than q. Subsequently, a weight clipping process [19] yields clipped weights W_{clip} with a bitwidth of q.

$$W_{\mathrm{clip}}(w) = \begin{cases} -2^{q-1}, & w < -2^{q-1} \\ w, & w \in [-2^{q-1}, 2^{q-1} - 1] \\ 2^{q-1} - 1, & w > 2^{q-1} - 1 \end{cases} \tag{8}$$

The difference between the original weights W and the clipped weights W_{clip} is defined as the weight bias W_{bias}.

$$W_{\mathrm{bias}} = -|W - W_{\mathrm{clip}}|, \tag{9}$$

Hence, the SAD operations outlined in Eq. 3 can be reformulated as:

$$-\sum |X - W| = -\sum |X - W_{\mathrm{clip}}| + \sum W_{\mathrm{bias}}, \tag{10}$$

where $\sum |X - W_{\mathrm{clip}}|$ represents q-bit clipped SAD operations, and $\sum W_{\mathrm{bias}}$ is a constant after model training and can be precomputed without incurring extra hardware overheads.

For a ReLU-activation based AdderNet model, the computation to obtain one output feature pixel is the SAD operations between the input features and the weight kernels across all input channels. After model training, both positive and negative weights are fixed, leaving only the feature maps changing with respect to different inputs. Based on Theorem

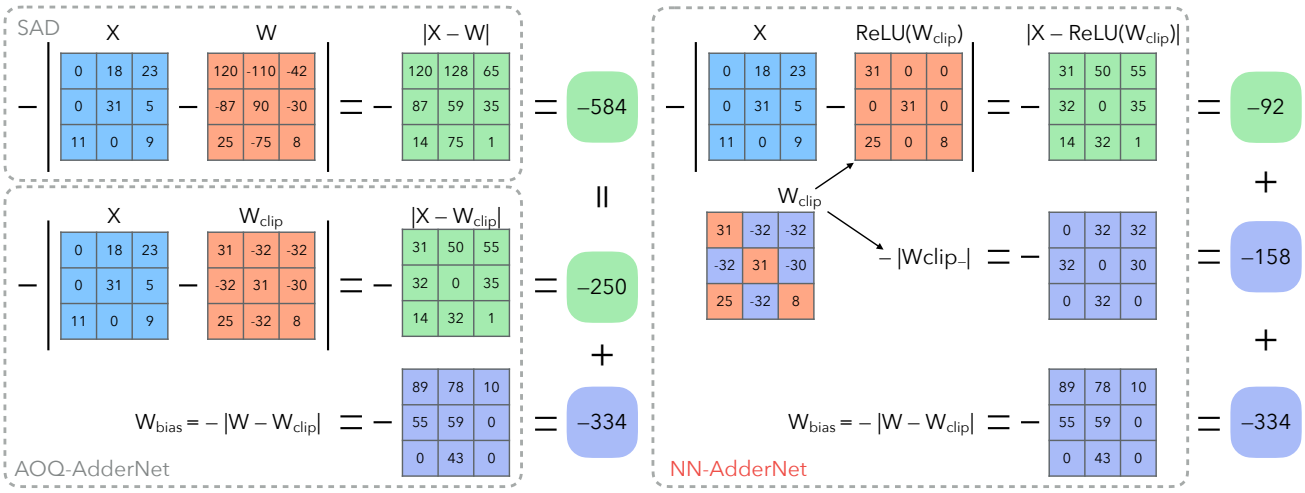


Fig. 2. A conceptual illustration of basic SAD operations in AdderNet (top left), AOQ with weight clipping (bottom left) in AOQ-AdderNet, and the proposed NN-AdderNet (right).

1, Eq. (6) holds valid with the assumption of ReLU activations, which is the widely adopted activation function used in popular DNN models like AdderNet. Obviously, Eq. (6) suggests that the basic SAD computation with signed weights is mathematically equivalent to the unsigned SAD computation with ReLU weights plus the sum of the absolute of all negative weights $\sum|W_-|$. Note that $\sum|W_-|$ can be precomputed after model training and ultimately handled by fuse-bias removal technique, which will not incur additional computation overheads during model inference. Hence, Eq. (6) demonstrates the feasibility of NN-AdderNet, where negative weights contribution to the AdderNet model computation will be all fused to *a feature bias* that can be removed by BN, leaving only positive weights matter to the computation. Moreover, Eq. (6) holds valid for both floating-point and quantized operations, which can be seamlessly applied with the activation-oriented quantization scheme.

Hence, combining Eq. (6) and Eq. (10), the SAD operations in AdderNet can be computed as:

$$-\sum|X-\mathrm{ReLU}(W_{\mathrm{clip}})|+\sum W_{\mathrm{bias}}-\sum|W_{clip-}|, \quad (11)$$

where the term $(\sum W_{\mathrm{bias}}-\sum|W_{clip-}|)$ is a constant after model training, and can be handled by fused bias removal technique, leaving the first term $-\sum|X-\mathrm{ReLU}(W_{\mathrm{clip}})|$ performing the essential quantized NN-AdderNet operation. A conceptual illustration of the vanilla SAD of AdderNet, Activation-oriented quantization of AOQ-AdderNet, and the proposed Activation-oriented NN-AdderNet have been demonstrated in Fig. 2.

IV. Dual-Sparsity Profiles in NN-AdderNet

In this section, we present details on NN-AdderNet to achieve improved quantization and compression performance.

Lossless compression of weights using entropy coding scheme, such as Huffman coding or Run Length Encoding (RLE), achieves limited compression ratio due to Gaussian- or Laplacian-like signed weight distributions in CNN and

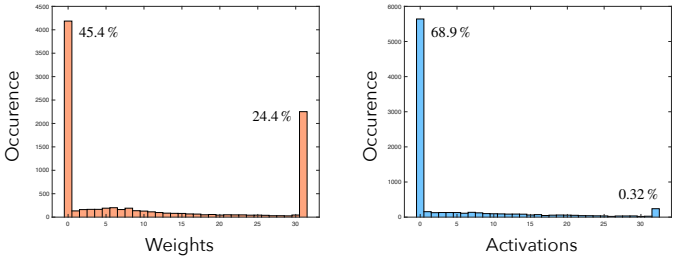


Fig. 3. A representative weight and activation distributions in quantized NN-AdderNet. For weights, a dual-sparsity profile can be observed, while activations exhibit the nominal '0' sparsity.

AdderNet, respectively. Basically, entropy coders rely on the knowledge of the probability of the source symbols to construct optimal coding for high compression ratio. Taking Huffman coding [27] as an example, this lossless compression algorithm constructs an optimal prefix code by analyzing the frequency of symbols in the source data, thereby reducing the total number of bits required to represent the data. Hence, to improve the compression performance of Huffman coder, the key is to skew the probability of symbols. Equivalently, the sparsity of certain symbols should be enhanced.

As we have already demonstrated in Eq. (11), vanilla AdderNet execution can be equivalently achieved via NN-AdderNet. In this way, we consider the lossless compression of quantized NN-AdderNet, i.e., performing Huffman encoding over the $\mathrm{ReLU}(W_{\mathrm{clip}})$. Fig. 3 shows the typical weight and activation distributions of NN-AdderNet after UINT5 activation-oriented quantization. As can be observed, after the nonnegative ReLU and weight clipping operations during the quantization process, the quantized weights of NN-AdderNet exhibit a unique distribution profile. Unlike previous approximate symmetrical signed weight distributions, the weights distribution of NN-AdderNet show a strong dual-sparsity profile, where a great portion of '0' symbols (45.4%, caused by ReLU operation), and significantly increased amount of '31' symbols (24.4%,

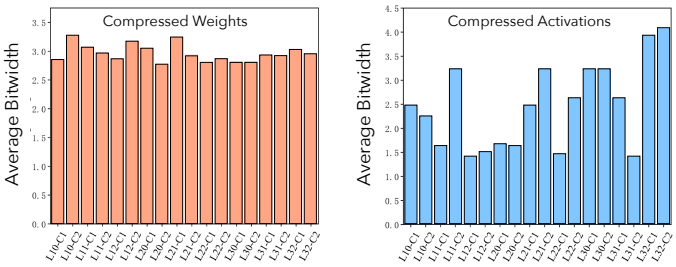


Fig. 4. ResNet-20 AdderNet on CIFAR-10 dataset: cross-layer compressed bitwidth of weights (left) and activations (right), respectively.

mainly from weight clipping) co-exist. It becomes apparent that the weight sparsity in the quantized NN-AdderNet has been significantly promoted, which will ultimately lead to improved compression performance of lossless entropy coders like Huffman coder. Moreover, the UINT5 activations are mainly from quantized ReLU operation, hence a more pronounced portion of '0' symbol (68.9%) is observed. Hence, ReLU activations can also be efficiently handled by entropy coders to achieve bitwidth reduction.

Fig. 4 shows the averaged number of compressed bits required for the weights $\mathrm{ReLU}(W_{\mathrm{clip}})$ and ReLU activations across Adder layers in ResNet20, respectively. Due to the unique dual-sparsity feature in quantized weights, it becomes possible for all the weights in the Adder layers to be compressed down to UINT3 (2.97-bit on average). For ReLU activations, due to a varying probability of '0's, the average compressed bits across layers also vary significantly (1.42 $\sim$ 3.25-bit).

V. Experimental Results

A. DNN Model Preparation & Setup

To evaluate the performance of the proposed NN-AdderNet, we conducted experiments using the ResNet20 [2], ResNet32 [2], and ResNet50 [2] backbones across three datasets: CIFAR-10, CIFAR-100 [28], and ImageNet, respectively. The ResNet20-based NN-AdderNet was trained from scratch on the CIFAR-10 dataset. The training setup utilized a GeForce RTX 3090 GPU with CUDA 12.6, Python 3.9, and PyTorch 2.0, running on an Ubuntu 22.04 LTS operating system. The training configuration included an initial learning rate of 0.05, a batch size of 256, and the SGD optimizer with a momentum of 0.9 and a weight decay of 5×10^{-4}. Additionally, we applied an adaptive learning strategy with cosine learning rate decay, as detailed in [29]. The ResNet32-based NN-AdderNet was also trained from scratch, but on the CIFAR-100 dataset, using the same setup as the ResNet20 experiment. The learning rate, batch size, and optimizer settings were kept consistent to ensure comparability. For ResNet50-based NN-AdderNet, we initially used a pre-trained model on the ImageNet dataset [14]. We then applied the proposed non-negative quantization technique, which involves nonnegative weight transformation and quantization, to obtain the final NN-AdderNet models. We consider uncompressed 8-bit and 5-bit post-training quantization for all DNN models, including the original CNN, AOQ-AdderNet, and NN-AdderNet to highlight the advantages and limitations of the proposed method. For uncompressed 8-bit models, quantization bitwidth of INT8, INT8, and UINT8 were adopted for CNN, AOQ-AdderNet, and NN-AdderNet weights, respectively. For uncompressed 5-bit models, CNN and NN-AdderNet adopts INT5 and UINT5 quantization bitwidth, respectively.

TABLE I
Summary of Accelerator Architectural Specifications

Architecture	PE	DRAM BW	SRAM Buffer
ResNet20/32	16×16	64-bit	100-KB
ResNet50	32×32	256-bit	10-MB

B. Hardware Validation Setup

For hardware-level evaluation, including energy and processing latency, we resort to CNNergy [30], an analytical simulator model for CNN performance estimation and optimization. Essential modifications were made to CNNergy simulator to support AdderNet evaluation. For the sake of fairness, register-transfer-level (RTL) designs for the multiply-accumulate (MAC) and the sum-of-absolute-difference (SAD) with varied bitwidth were implemented in Verilog HDL. The RTL designs were synthesized using Synopsys Design Compiler using a commercial 65-nm low-power CMOS process, all with a peak operating frequency of 200-MHz at nominal supply voltage 1.2-V and room temperature of 27°C. The synthesized results were then normalized to 45nm and used as the inputs of the CNNergy simulator.

The accelerator architectural specifications are summarized in Table I. The granularities of the processing elements (PEs) were determined with respect to different DNN architectures, where 256 and 1024 PEs were adopted for ResNet20/32 and ResNet50 architectures, respectively. The total off-chip communication bandwidth to the DRAM and the on-chip SRAM buffer size were set to be 64-bit and 256-bit, 100-KB and 10-MB for ResNet20/32 and ResNet50 architectures, respectively. The weights and activation compression were benchmarked directly using the test datasets across CIFAR-10/100 and ImageNet, which will be used to accurately assess both the energy and latency benefits obtained from the communication bandwidth reduction between the off-chip DRAM and on-chip global buffer. Huffman codec was used for lossless data compression and decompression.

C. Experimental Result

1) Model Compression Performance: Fig. 5 shown the experimental results on weight and activation lossless compression performance of CNN, AOQ-AdderNet, and NN-AdderNet across different model architectures, datasets, quantization levels, and the respective quantized model accuracies.

For model quantization and compression, NN-AdderNet possesses greater weight compression potential due to its unique dual-sparsity characteristics as compared to CNN and

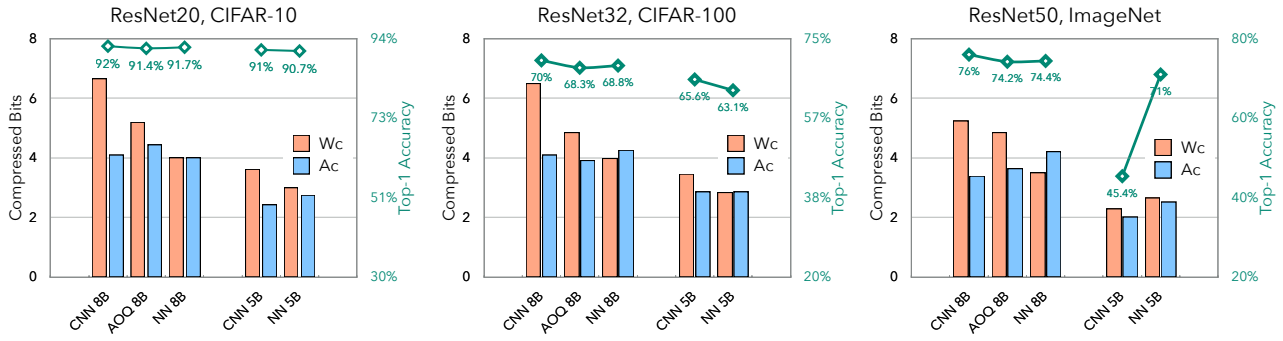


Fig. 5. The lossless compressed weights and activations (averaged) of the original 8-bit and 5-bit CNN, AOQ-AdderNet, and NN-AdderNet models. The corresponding model accuracies are also annotated.

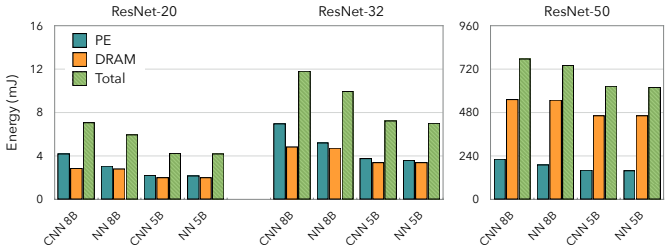


Fig. 6. Energy consumption (lossless compressed) of 8-bit and 5-bit CNN and NN-AdderNet models across ResNet20, ResNet32, and ResNet50.

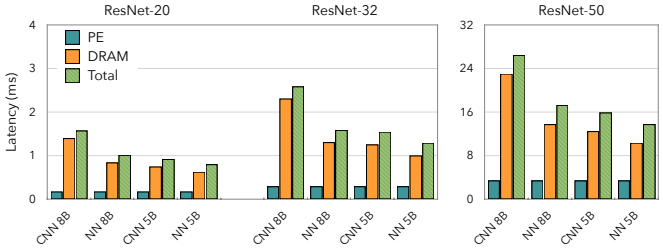


Fig. 7. Inference latency (lossless compressed) of 8-bit and 5-bit CNN and NN-AdderNet models across ResNet20, ResNet32, and ResNet50.

AOQ-AdderNet that only have signed weights with low sparsity. As illustrated in Fig. 5, for 8-bit models, NN-AdderNet achieves 2.65-bit, 2.52-bit, and 1.74-bit savings on ResNet20, ResNet32, ResNet50 as compared to the CNN baseline. On average, the compressed weights achieve UINT4 or lower averaged compressed bits, as opposed to 5.2 $\sim$ 6.6-bit in CNN models. The corresponding model accuracy drop due to employing NN-AdderNet is only 0.3%, 1.2%, and 1.6% on ResNet20, ResNet32, ResNet50, respectively. For 5-bit models, NN-AdderNet achieves 0.61-bit (ResNet20) and 0.63-bit (ResNet32) savings as compared to the CNN baseline. On average, the compressed weights achieve UINT3 or lower averaged compressed bits, as opposed to 3.6-bit for compressed CNN weights. The corresponding model accuracy drop of NN-AdderNet is 0.3% and 2.5% on ResNet20 and ResNet32, respectively.

2) Hardware-Level Demonstration: For compressed ReLU activations illustrated in Fig. 5, NN-AdderNet exhibits comparable but slightly lower sparsity compared to its CNN counterpart. While this won't pose a significant challenge from a model compression standpoint, compressed activations can still impact the overall accelerator energy consumption and latency during inference.

The energy consumption of different DNN accelerators are illustrated in Fig. 6. While PE energy is higher than the off-chip communication to DRAM energy for both CNN and NN-AdderNet on ResNet20/32, off-chip communication to DRAM energy becomes dominant for larger models like ResNet50. The total energy savings for 8-bit NN-AdderNet can be up to 16% over the CNN counterpart, whereas the energy benefits become marginal for 5-bit models.

The inference latency of the NN-AdderNet accelerator are clearly advantageous, mainly originated from the significantly reduced off-chip communication to DRAM. As shown in Fig. 7, the PE latency remains constant for a given model architecture, as all models were assumed to operate at 200-MHz. As expected, due to higher attainable compression ratio of the NN-AdderNet model, 8-b NN-AdderNet model achieves 35.7%, 38.6%, and 34.8% total latency reduction, while 5-bit NN-AdderNet model achieves 13%, 16.2%, and 13.5% total latency reduction, all compared with their CNN counterparts.

VI. Conclusion

In this paper, we present a novel quantization- and compression-friendly nonnegative AdderNet, which is a mathematically equivalent formulation of vanilla AdderNet with signed weights. The nonnegative formulation, together with the activation-oriented quantization scheme, yields ultra-low bitwidth AdderNet quantization and compression. Experimental results demonstrate that NN-AdderNet achieves unsigned integer quantization, and further promote a novel dual-sparsity distribution in NN-AdderNet weights. Hence, model compression of sub-4bit becomes possible. Experimental results show uncompressed 8-bit and 5-bit NN-AdderNet can achieve sub-UINT4 and sub-UINT3 bitwidth, respectively. Correspondingly, this leads to up to 36% latency and 16% energy improvement of NN-AdderNet based accelerators over the CNN counterpart. These results demonstrate that NN-AdderNet can be advantageous in both algorithm and hardware efficiency.

REFERENCES

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[4] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017.

[5] Zhi Tian, Xiangxiang Chu, Xiaoming Wang, Xiaolin Wei, and Chunhua Shen. Fully convolutional one-stage 3d object detection on lidar range images. *Advances in Neural Information Processing Systems*, 35:34899–34911, 2022.

[6] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[7] Haoran You, Xiaohan Chen, Yongan Zhang, Chaojian Li, Sicheng Li, Zihao Liu, Zhangyang Wang, and Yingyan Lin. Shiftaddnet: A hardware-inspired deep network. *Advances in Neural Information Processing Systems*, 33:2771–2783, 2020.

[8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[9] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2704–2713, 2018.

[10] Mengshu Sun, Zhengang Li, Alec Lu, Yanyu Li, Sung-En Chang, Xiaolong Ma, Xue Lin, and Zhenman Fang. Film-qnn: Efficient fpga acceleration of deep neural networks with intra-layer, mixed-precision quantization. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 134–145, 2022.

[11] Junsong Wang, Qiuwen Lou, Xiaofan Zhang, Chao Zhu, Yonghua Lin, and Deming Chen. Design flow of accelerating hybrid extremely low bit-width neural network in embedded fpga. In *2018 28th international conference on field programmable logic and applications (FPL)*, pages 163–169. IEEE, 2018.

[12] Xilinx. Wp486: Deep learning with int8 optimization on xilinx devices. *White Paper*, 2017.

[13] AmirAli Abdolrashidi, Lisa Wang, Shivani Agrawal, Jonathan Malmaud, Oleg Rybakov, Chas Leichner, and Lukasz Lew. Pareto-optimal quantized resnet is mostly 4-bit. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3091–3099, 2021.

[14] Hanting Chen, Yunhe Wang, Chunjing Xu, Boxin Shi, Chao Xu, Qi Tian, and Chang Xu. Addernet: Do we really need multiplications in deep learning? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1468–1477, 2020.

[15] Yunhe Wang, Mingqiang Huang, Kai Han, Hanting Chen, Wei Zhang, Chunjing Xu, and Dacheng Tao. Addernet and its minimalist hardware design for energy-efficient artificial intelligence. *arXiv preprint arXiv:2101.10015*, 2021.

[16] Mingyong Zhuang, Xinhui Liao, Huhong Wu, Jianyang Zhou, and Zichao Guo. Vlsi architecture design for adder convolution neural network accelerator. In *2021 IEEE 15th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 5–9. IEEE, 2021.

[17] Ke Ma, Lei Mao, and Shinji Kimura. A novel fpga-based convolution accelerator for addernet.

[18] Yunxiang Zhang, Biao Sun, Weixiong Jiang, Yajun Ha, Miao Hu, and Wenfeng Zhao. Wsq-addernet: Efficient weight standardization based quantized addernet fpga accelerator design with high-density int8 dsp-lut co-packing optimization. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[19] Yunxiang Zhang, Omar Al Kailani, and Wenfeng Zhao. Addernet 2.0: Optimal fpga acceleration of addernet with activation-oriented quantization and fused bias removal based memory optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pages 1–6, 2024.

[20] Soyeon Um, Sangyeob Kim, Sangjin Kim, and Hoi-Jun Yoo. A 43.1 tops/w energy-efficient absolute-difference-accumulation operation computing-in-memory with computation reuse. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 68(5):1605–1609, 2021.

[21] Haikang Diao, Yifan He, Xuan Li, Chen Tang, Wenbin Jia, Jinshan Yue, Haoyang Luo, Jiahao Song, Xueqing Li, Huazhong Yang, et al. A multiply-less approximate sram compute-in-memory macro for neural-network inference. *IEEE Journal of Solid-State Circuits*, 2024.

[22] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 143–159. Springer, 2020.

[23] Ying Nie, Kai Han, Haikang Diao, Chuanjian Liu, Enhua Wu, and Yunhe Wang. Redistribution of weights and activations for addernet quantization. *Advances in Neural Information Processing Systems*, 35:22739–22751, 2022.

[24] Xinghao Chen, Chang Xu, Minjing Dong, Chunjing Xu, and Yunhe Wang. An empirical study of adder neural networks for object detection. *Advances in Neural Information Processing Systems*, 34:6894–6905, 2021.

[25] Dehua Song, Yunhe Wang, Hanting Chen, Chang Xu, Chunjing Xu, and DaCheng Tao. Addersr: Towards energy efficient image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15648–15657, 2021.

[26] Bowen Li, Yao Xin, Chao Li, Youneng Bao, Fanyang Meng, and Yongsheng Liang. Adderic: towards low computation cost image compression. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2030–2034. IEEE, 2022.

[27] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[28] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[29] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[30] Susmita Dey Manasi, Farhana Sharmin Snigdha, and Sachin S Sapatnekar. Neupart: Using analytical models to drive energy-efficient partitioning of cnn computations on cloud-connected mobile clients. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(8):1844–1857, 2020.

High Energy-efficiency and Low latency In-Memory Computing using Analog Accumulator and In-Memory ADC with shared References

Junyi Yang, Shuai Dong, Zhengnan Fu, Hongyang Shang, Arindam Basu
Department of Electrical Engineering, City University of Hong Kong, Hong Kong, China

***Abstract*—This article proposes a 256×128 in-memory computing array using reconfigurable in-memory analog-to-digital conversion with shared references for high area efficiency (area overhead of 3% is $9X$ better than traditional). A dual-8T SRAM bitcell is used to achieve read-write decoupling and store ternary weights. Read World Line Under Drive enabled Cascode helps to minimize current variations producing high linearity. Multi-bit input is handled with low latency and high energy-efficiency by using bit-slicing (BS) with near-memory charge-sharing based binary weighted accumulator (CHA). Using noise resilient training, we show software comparable performance for a MLP on MNIST, VGG-8 on CIFAR-10, and graph attention network on Cora, with respective accuracy reductions of only 0.1%, 0.8% and 0.5% due to non-idealities. The proposed macro demonstrates high energy/area efficiency (1146 TOPS/W, 27 TOPS/mm^2 at 1/2/1b) in 65 nm CMOS. It increases throughput (by $1.9X$) and linearity (by $23X$) compared to input pulse-width modulation by using BS and CHA. Compared to conventional BS with digital accumulation after ADC, this method has $1.7X/6.6X$ better energy-efficiency/throughput by reducing ADC operations.**

***Index Terms*—In-memory computing, In-memory ADC, SRAM, Charge-sharing, Multi-bit**

I. Introduction

Deep Neural Networks (DNNs) produce state-of-the-art solutions across different domains such as speech processing [1], image classification [2], natural language processing [3] etc. However, the bottleneck in implementing DNNs on current hardware stems from the frequent memory access requirements imposed by the von Neumann architecture, which is known as the memory wall or von Neumann bottleneck [4], [5]. Moreover, as DNN models expand in scale, the heightened energy consumption and latency associated with storing the parameters of sizable models further exacerbate this challenge.

In-memory computing (IMC) has emerged as a promising approach to address these limitations [6]. IMC involves conducting computations within the memory itself, specifically on its bit line, rather than repeatedly reading and writing data from memory during each cycle. Furthermore, IMC enables the execution of multiply-and-accumulate (MAC) operations within each column by activating multiple rows, thereby significantly diminishing the generation of intermediate data and enabling highly parallelized computation [7]. Among the various IMC techniques that have been investigated, analog/mixed-signal IMC utilizing SRAM has shown substantial enhancements in terms of latency, energy/area efficiency, and throughput [8], [9]. However, several challenges remain to be solved such as resilience to PVT (process-voltage-temperature) variations, reducing energy and area overhead of peripheral data converters, reducing non-linearity and enabling high bit-precision MAC operations with low-latency, energy and area overhead.

In this work, we propose a multi-bit, highly reconfigurable IMC (1-7/2-4/1-7b, input/weight/output) by using bit-slicing (BS) with charge-sharing based binary weighted accumulator (CHA) and reconfigurable (1-7b) In-memory Analog to Digital Converter (IMADC) with shared references. A dual 8T SRAM bitcell based on Read World Line Under Drive Cascode (RWLUDC) has been employed in the design of a macro to minimize current variations with bitline voltage change enabling larger signal margin. Our work results in 1.9X and 6.6X improvement in latency over conventional pulse width modulation (PWM) mode and BS mode respectively under 7-bit input/output by using bit-slicing with charge-sharing based binary weighted accumulator (BSCHA) as shown in Fig. 1(a). It also increases MAC linearity by $\approx 23X$ compared to PWM mode. IMADC with shared references diminishes the hardware overhead without reducing throughput. Our ADC area overhead (ADC area/MAC array area) is merely 3%, which is much lesser than prior work in Fig. 1(b) (9X improvement compared to 27% for traditional IMADC). We also show inference accuracies at par with software baselines due to noise aware training and show requirement for high input/output bit resolution (7b) for complex networks like graph attention.

II. Background and Related Work

A major issue limiting the application of analog IMC is the area, energy and latency overhead of ADC [10]. To mitigate the area overhead of traditional ADCs [11], [12], column multiplexing is used where M columns share one ADC. However, this comes at the cost of reduction in system throughput by a factor of M. Therefore, IMADC was proposed [13], [14], where each column can operate a slow ramp ADC simultaneously. This has similar or better throughput as earlier

This work was supported by the Innovation Technology Fund Mid-Stream Research Program (ITF MSRP) under Grant ITS/018/22MS. Any opinions, findings, conclusions, or recommendations expressed in this material do not reflect the views of the Government of the Hong Kong Special Administrative Region, the Innovation and Technology Commission, or the Innovation and Technology Fund Research Projects Assessment Panel. (Junyi Yang and Shuai Dong contributed equally to this work.) (Corresponding authors: Junyi Yang; Arindam Basu. e-mail: junyiyang8-c@my.cityu.edu.hk; arinbasu@cityu.edu.hk.)

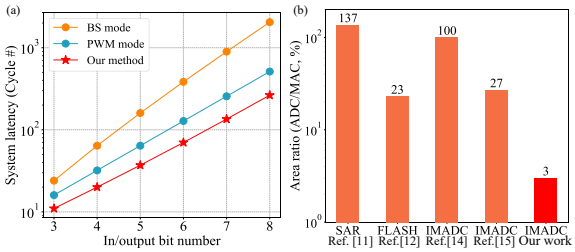


Fig. 1. System latency and ADC area overhead: (a) System latency (Our method, PWM mode and BS mode). (b) ADC area overhead (ADC area/MAC array area).

designs [11], [12] but with the advantage of reconfigurability (of resolution and MAC size) as well as robustness to PVT variations due to replica biasing. However, only binary NNs could be implemented, which fall short of meeting the accuracy demands of intricate neural networks [13], [14]. Also, integrating an ADC into each column significantly increases the hardware overhead, especially as the resolution of the ADC, n_o, increases. For instance, achieving a 5-bit ADC requires 64 extra bit cells in every column. Fig. 1(b) shows ADC area overhead defined as the ratio of ADC area over MAC array area in previous works [11], [12], [14], [15].

BS is a popular approach for increasing the input bit resolution (n_i), where input bits are applied serially and partial sums (psums) are combined after digitization. However, when employing a ramp ADC for output quantization, the total latency of BS required becomes $n_i \times 2^{n_o}$ clock cycles, since each bit demands 2^{n_o} clocks. Also, this incurs n_i times energy overhead of ADC as well as additional digital circuits for combining psums. Another alternative, PWM for multi-bit input [16]–[18] incurs a time cost of $2^{n_i} + 2^{n_o}$, which is also notably high. PWM also requires Digital-Analog converters (DAC) at the input or counters with very fast clocks. Fig. 1(a) illustrates this tradeoff between system latency and input/output bit resolution (assuming $n_i = n_o$).

III. Proposed Architecture and Methodology

A. Overall Hardware Architecture

Fig. 2(a) shows the top-level schematic of the IMC and IMADC based on SRAM, which includes 256×127 dual 8T SRAM bitcells for MAC, one column (256×1) for references of the reconfigurable ADC (shared by 127-column MAC array), charge-sharing circuits, sense amplifiers(SA), ripple counters, input drivers and voltage buffers. To clearly illustrate how it works, the timing of MAC operations and a set of ADC timing diagrams are depicted in Fig. 2(b). Input bits b_i are applied serially to create $V_{MAC} = V_{MACP} - V_{MACN} = \sum_{k=1}^{N} W_k X_k^i$ on the bitlines (using current-mode operation) where $N = 256$ is the dimension of the input vector. The near memory capacitors $C_{X1} = C_{X2}$ perform binary weighted accumulation of partial sums in charge mode to create the final accumulated voltage V_{Acc}. Next, the ADC column creates a reference ramp voltage ΔV_{ADC} that increases every clock cyle. This is shared with the remaining columns so that each of the 127 SAs can simultaneously compare the ramp voltage with their corresponding V_{Acc} values. Due to the shared nature of the ramp voltage, differential voltage buffers are required to enhance the driving capability of the ramp reference voltage. The ripple counter converts the output thermometer code of the SAs into a binary code. Details of each block are described next.

B. Dual 8T SRAM based In-Memory Binary Multiplication

A dual 8T SRAM bitcell has been utilized for the implementation of the macro as shown in Fig. 2(c). It performs binary multiplication of an input with a ternary weight using a decoupled read path, which includes the four blue NMOS transistors (two transistors on each side as shown in Fig. 2(c)). The read wordline (RWL) receives either a 1 (a high DC voltage) or a 0 (a low DC voltage) signal according to the input bit, while a ternary weight (-1: Q_L is low and Q_R is high, 0: both Q_L and Q_R are low, +1: Q_L is high and Q_R is low) is stored in the SRAM bitcell. Initially, both RBLL and RBLR voltages are pre-charged (by signal *PCH_MAC* in Fig. 2(b)) to a high voltage using two pre-charge PMOS transistors before the binary multiplication operation commences. A table in Fig. 2(c) illustrates the multiplication results of the bitcell. When the input is 0 (RWL is low), no current discharges the RBLL or RBLR, regardless of the weight stored in the SRAM bitcell. Conversely, When the input is 1 (RWL is high), either RBLL or RBLR is discharged via the corresponding NMOS transistor, where the gate is linked to the internal 6T-SRAM cell with a high voltage (0.5 V). The resulting voltage disparity between RBLL and RBLR (+ΔV or -ΔV) signifies the binary multiplication outcome. The layout of the dual 8T cell shown in Fig. 2(c) occupies 3.6 μm x 1.8 μm.

Accumulation occurs due to all bitcells in a column simultaneously discharging the bitline capacitor C_{BL}. The MAC value is directly proportional to the voltage drop induced by the capacitor discharge process. A more detailed explanation of this process will be provided in Section III-D.

C. RWL Underdrive enabled Cascode for enhanced linearity

A major problem of current based MAC architecture is maintaining linearity of C_{BL} discharge due to variation in discharge current of a cell, I_u, with change in V_{DS} as the voltage on BL changes [13], [14]. This can be improved by increasing the output impedance of the effective current source created by transistors M0(L/R) and M1(L/R) in Fig. 2(c) by making M1 behave as a cascode [19]. We propose to achieve this by RWL underdrive voltage enabled cascode (RWLUDC) technique–the supply voltages for RWL is set to 0.8 V while RBLL/RBLR pre-charging are set at 1 V while gate voltage of M0 is at 0.5 V (SRAM supply voltage). The minimum RBL voltage for which M1 acts as a cascode is given by $V_{min} = V_{RWL} - V_{T1}$ where V_{T1} denotes the threshold voltage of M1, i.e. lower V_{RWL} leads to higher RBL swing. As a result, we achieve a voltage swing range of 700 mV on the RBL for a 1% change in I_u (Fig. 2(c))–7X less than the variation in I_u over the same range for the conventional case.

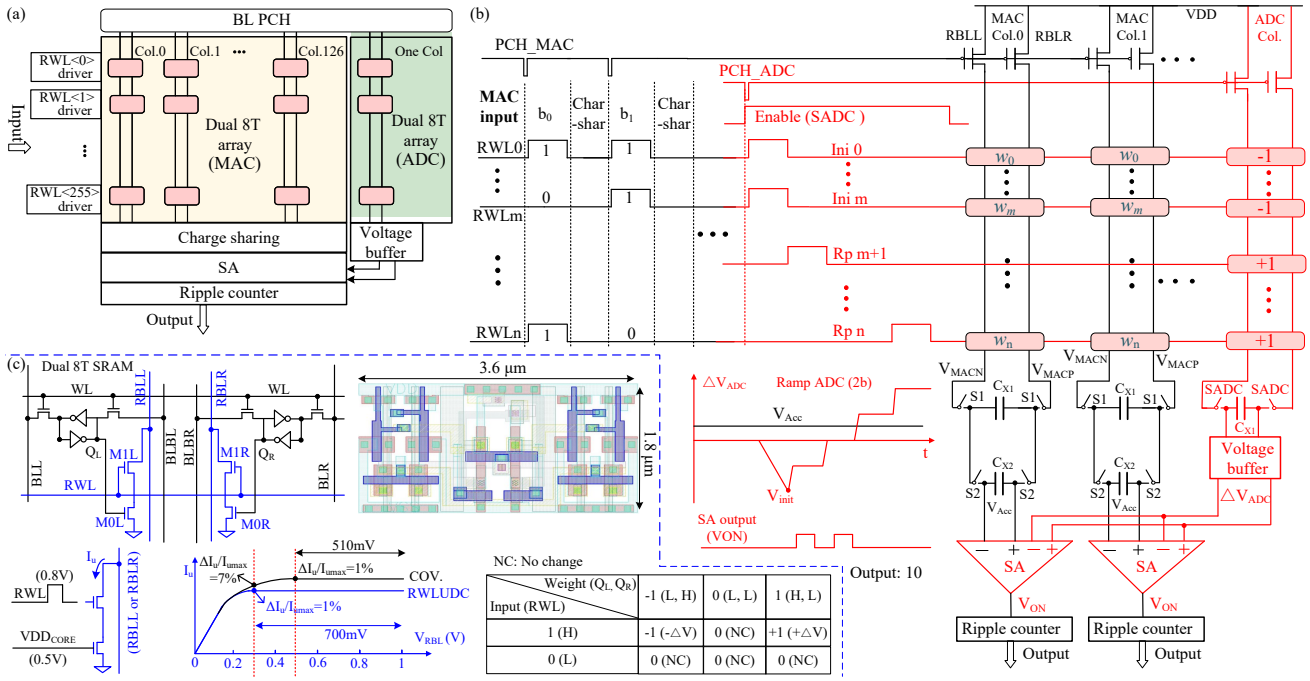


Fig. 2. Overall structure of the proposed dual 8T SRAM IMC and IMADC: (a) system hardware structure. (b) Detailed circuit and timing diagram of MAC columns and IMADC. (c) Information of dual 8T SRAM cell.

Alternatively, the conventional method needs to reduce signal swing by $1.4X$ to $\approx$ 510 mV for the same 1% variation. Single transistor (in 7T cell) based discharge paths have an even smaller signal swing range of $\approx$ 200 mV [15].

D. Bit Slicing with Charge Sharing based Binary Weighted Accumulator (BSCHA) for Multi-bit Input

Conventional BS in IMC applies the input bits b_0 to b_{n_i-1} serially over n_i cycles while the MAC voltage is digitized separately for every bit to produce psums P_0 to P_{n_i-1}. The final MAC result is obtained by weighted summation in the digital domain as $P_{n_i-1} + \frac{P_{n_i-2}}{2} + ... + \frac{P_0}{2^{n_i-1}}$. The resulting energy ($E_{conv}$) and latency ($T_{conv}$) in terms of clock cycles of the conventional BS approach are given by:

$$E_{conv} = n_i(E_{MAC} + E_{ADC} + E_{dig}) \qquad (1)$$
$$T_{conv} = n_i(T_{MAC} + T_{ADC} + T_{dig})$$

where $E(T)_{MAC}$, $E(T)_{ADC}$ and $E(T)_{dig}$ denote the energy (latency) of MAC, ADC and digital weighted accumulation respectively for every cycle. This incurs a heavy penalty in terms of energy and latency due to the repeated operation of the ADC over n_i cycles.

To overcome this issue, we propose a binary weighted accumulation before digitization technique accomplished by the addition of two capacitors and four switches between RBLL and RBLR (Fig. 2(b)). The simplified circuit diagram is illustrated in Fig. 3(a), where C_{BL} represents the parasitic capacitance on RBL, and I_{MACN} and I_{MACP} are the total discharge currents of each column. Intuitively, C_{X1} is used to sample the MAC voltage in every cycle for every input bit, and then redistribute it with an **equally sized** capacitor C_{X2}. This process results in the binary weighted charge sharing based accumulation (CHA) required to combine the partial sums for every bit. It is similar in principle to charge redistribution DACs [19]; however, instead of an input digital code, a different analog MAC voltage is input in every cycle.

The detailed process is shown in Fig. 3(b) including the timing diagram of pre-charging, input pulse, and charge-sharing control switches S1 and S2. We analyze the case for the **i-th** bit of the input (assuming both capacitors were reset before the first input bit was presented):

1) During pre-charging, Switch S1 is closed, and S2 is open. After pre-charging, the voltages of the two C_{BL}s and the voltages across C_{X1} reach VDD resetting the previously stored voltages.

2) Subsequently, RWL is enabled for time Δt, initiating the discharge of C_{BL}. The voltage on C_{X1} changes as the two C_{BL}s discharge. By applying Kirchhoff's Current Law, we can derive Eq. (2) for the voltage $V_{MAC} = V_{MACP} - V_{MACN}$ as:

$$V^i_{MAC} = \frac{\left(I^i_{\text{MACP}} - I^i_{\text{MACN}}\right)\Delta t}{2C_{X1} + C_{BL}} = \frac{I_u \sum_{k=1}^{N} W_k X^i_k}{2C_{X1} + C_{BL}} \Delta t \qquad (2)$$

where V^i_{MAC} denotes the MAC voltage developed on the capacitors C_{BL} and C_{X1} when the i-th input bit is presented to the circuit. The term $2C_{X1}$ can be understood from Miller's effect applied to a floating capacitor [19].

3) Subsequently, S1 opens, and S2 closes, initiating the charge-sharing between capacitors C_{X1} and C_{X2}. The voltage on capacitor C_{X2} denoting the final accumulated voltage V_{Acc} becomes a combination of half of its previous value V^{i-1}_{Acc} and

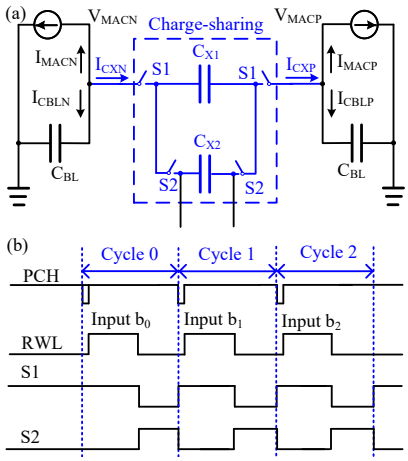


Fig. 3. (a) Effective circuit model for generation of differential MAC voltage and charge sharing. (b) Timing diagram of charge sharing. The MAC voltage is sampled on C_{X1} when S1 and RWL are enabled, while charge redistribution between C_{X1} and C_{X2} occurs when S2 is enabled.

half of the the new MAC value V^i_{MAC} as shown in in Eq. (3):

$$\begin{cases} C_{X2}V^{i-1}_{Acc} + C_{X1}V^i_{MAC} = V^i_{Acc}\,(C_{X1}+C_{X2}) \\ V^i_{Acc} = \dfrac{V^{i-1}_{Acc}}{2} + \dfrac{V^i_{MAC}}{2} = \displaystyle\sum_{k=0}^{i} \frac{V^k_{MAC}}{2^{i+1-k}} \end{cases} \tag{3}$$

Compared to other charge sharing methods [11], our proposed method does not require binary weighted capacitors, reducing area and matching requirements. In our implementation, $C_{X1} = C_{X2} = 50$ fF is chosen to make it approximately similar to $C_{BL}/2$ (balancing two parts in denominator of Eq. (2)) since it provides a good tradeoff between reducing noise and maintaining large enough MAC voltages. The cell array layout is designed to utilize metal layers M1 to M4. To optimize the overall chip area, metal layers M6 to M7 are allocated for constructing MOM capacitors C_{X1} and C_{X2}, while metal layer M5 is grounded to shield the cell array from the capacitors. Each capacitor has an area of $3 \times 47\mu m^2$, enabling the straightforward placement of two MOM capacitors on top of every column in the cell array without incurring any additional area overhead.

The energy and latency for our approach can be written as:

$$\begin{aligned} E_{proposed} &= n_i(E_{MAC} + E_{ana}) + E_{ADC} \\ T_{proposed} &= n_i(T_{MAC} + T_{ana}) + T_{ADC} \end{aligned} \tag{4}$$

where $E(T)_{ana}$ denotes the energy (latency) of analog weighted accumulation and other terms are as defined earlier. The major advantage of our approach stems from the fact that the ADC is now operated only once as opposed to n_i times as in Eq. (1). The errors induced by the analog accumulation can be compensated by co-design of the algorithm as shown in Sec. III-F.

In contrast to the PWM mode where the input latency is 2^{n_i}, our method demonstrates clear advantages in terms of input latency, as shown in Fig. 1(a) (18X improvement for 7-bit input). Furthermore, in comparison to the PWM mode for multi-bit input, our method exhibits superior linearity in generation of V_{Acc} due to smaller voltage swing on $V_{RBLL/R}$. As mentioned in Section III-C, Early effect of discharge transistor of the bit cell (M0, M1) results in a decrease in I_u, subsequently diminishing V_{Acc} and giving rise to considerable errors.

E. Reconfigurable 1-7 bit In-Memory Analog-Digital Converter (IMADC) with Shared References

The red part in Fig. 2(b) shows the architecture and operation of IMADC. The replica cells for ADC are exactly the same as the MAC part. Conceptually the operation is split into two parts: (1) generation of initial ramp voltage V_{init} (Fig. 2(b) bottom) in one clock cycle by first enabling 2^{n_o-1} bit-cells with weight of -1 , and (2) generation of ramp voltage by enabling 2^{n_o} bit-cells with weight of $+1$, one cell enabled every clock followed by SA based comparison each time. Fig. 2(b) depicts the schematic outline of the global ramp reference voltage (ΔV_{ADC}) generation for a 2-bit ADC and the quantization process of the voltage in the MAC column (V_{Acc}).

Earlier work [14] used ramp generators in every column leading to $\approx 50\%$ overhead in terms of area. In the proposed architecture, we replace it with a global reference generator. Due to shared RWL lines between MAC and ADC ramp generator, an *Enable* signal (*SADC*) is used to ensure the ADC starts after the MAC operation ends, thus saving switching and voltage buffer power dissipation. *SADC* is used to gate the *PCH* and *RWL* to create *PCH_ADC* and *RP* signals for the ADC column. It also enables the SA. Depending on the required resolution, different number of cells can be used for the ramp generation. A maximum resolution of $n_o = 7$ can be obtained with the $N = 256$ rows in the macro; this uses 192 cells while the remaining 64 can be used for calibration to ensure the ramp crosses "0" at the mid-code [14].

While the proposed method reduces area overhead to $\approx 3\%$ (Fig. 1(b)), it requires two modifications–(1) the SA has to be converted to a double differential one [20] that compares the differential ramp voltage ΔV_{ADC} with a differential V_{Acc} and (2) a voltage buffer has to be used after the ramp generator to drive the input capacitance of 127 SAs. The output signal V_{ON} of the SA is the input of the ripple counter. In the scenario where $V_{Acc} = 2$, as illustrated in Fig. 2(b) (bottom), the output of the ripple counter is 10. The ripple counter acts as a thermometer to binary code converter, eliminating the need for large register arrays to store the thermometer code, thereby enhancing the area efficiency of the system.

Two self-biased voltage buffers is utilized in the IMADC (Fig. 4(a)). They are designed to operate at a frequency of 200 MHz with load capacitance of 100 fF while dissipating 40 μW from a power supply of 1 V. In the Monte Carlo (MC) simulation depicted in Fig. 4(b), the offset voltage determined by the output difference of the two voltage buffers minus the input difference is shown. The MC results indicate that the mean offset is 3.3 mV, with a standard deviation of 0.1 mV.

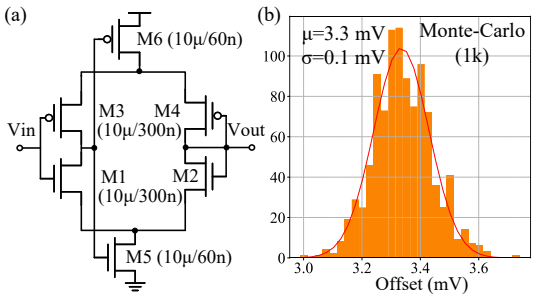


Fig. 4. Self-biased voltage buffer (a) Circuit Diagram. (b) Monte Carlo simulation for voltage offset showing it to be less than ADC LSB.

Algorithm 1 Noise resilient training.

Input: X_t, W_t and σ
Output: W_{t+1}
loop
  Feed forward propagation:
  $Y_\mathbf{t} \leftarrow Q(W_t \cdot X_t)$
  $Z_\mathbf{t} \leftarrow f(Y_t + \sigma)$
  Back-propagation propagation:
  $g_\mathbf{t} \leftarrow$ calculate gradient based on W_t and $f(W_t \cdot X_t)$
  $W_{\mathbf{t+1}} \leftarrow W_\mathbf{t} + \eta \times g_\mathbf{t}$
end loop

This is smaller than the ADC LSB of 4.8 mV and removes the need for calibration. Even with the added buffers and more complex SA, the proposed method dissipates $\approx 3\%$ less energy than having a ramp generator in every column [13], [14] due to the reduced bitline capacitances and switching energy of our shared reference scheme.

F. Noise-resilient Training (NRT)

To mitigate the impact of non-idealities (such as mismatch between C_{X1} and C_{X2}, discharging currents of the bitcells etc.) in the proposed architecture, noise-resilient training (NRT) is used to compensate for the accuracy loss, inspired by quantization-aware training. Details of the method are shown in Algorithm 1 where X_t, W_t, Y_t, Z_t and g_t denote the input, weight, ideal quantized MAC value, non-ideal activation value and gradient on the t-th iteration. $Q()$ and $f()$ are the ADC quantization and activation functions. σ denotes samples from a noise distribution mimicking the hardware non-idealities and is obtained from circuit simulation (Fig. 5(a)). As shown in Algorithm 1, in the feed-forward propagation stage, ADC output with random noise is sent to the activation function. However, in back-propagation, ideal MAC values without noise are used to calculate the more accurate gradient and then update the weight.

IV. Simulation Result

A. Software results

We choose three networks to evaluate the proposed method: one MLP network (784-128-128-10) on MNIST dataset, VGG-8 on CIFAR-10 dataset, and graph attention (GAT) network [21] on Cora dataset.

To conduct noise-resilient training (NRT), the proposed architecture is evaluated by using SPICE simulation in 65 nm CMOS technology. In Fig. 5(a), we compare 4-bit ADC outputs with the theoretical MAC values at different temperatures (0 °C, 27 °C and 70 °C) and the extracted errors approximately follow normal distributions. It can be seen that the error is always small, thereby indicating the robustness of the IMADC architecture in the face of temperature fluctuations due to replica biasing. Then the nominal error distribution ($N(-0.12, 0.83)$ at 27 °C) is used to inject noise into the three networks. 1) the MLP has less than 0.2% accuracy decrease for 2-4 bit ADC compared to the quantized models due to the usage of NRT and shallow architecture of the model, which prevents deep accumulation of noise. 2) VGG-8 is a challenging model with deeper architecture. However, NRT can still restore precision to a comparable level of accuracy, with a decrease of less than 0.9% compared to 3-5 bit quantized model. 3) GAT is a novel graph neural network that requires a higher n_o for good performance. NRT demonstrate the ability to restore accuracy to within 0.9% for this model. The highest ADC bit resolution n_o used for these three models results in 0.8%, 1.2% and 0.7% drop in accuracy from their floating point baselines (98.0%, 93.4% and 84.0%), making them comparable to SOTA accuracy for quantized models [13], [21], [22].

B. Hardware results

Our hardware architecture supports multi-bit weights to cater to the requirements of complex networks. Multiple-bit weights can be realized by connecting multiple cells in parallel. For instance, in a 4-bit configuration, excluding the sign bit, the remaining three bits can be implemented using 1, 2, or 4 cells in parallel (for a total of 7 cells for one 4-bit weight). The sign of the weight can be intrinsically implemented using a dual 8T architecture based on usage of the left or right cells.

Latency and throughput: We assessed the latency of the system across three modes: proposed, PWM, and conventional BS, with respective latencies of $n+2^n$, 2^{n+1}, and $n2^n$, assuming the utilization of ramp ADC output and $n_i = n_o = n$. Fig. 1(a) illustrates that our work yields 1.9X and 6.6X improvement of throughput over PWM mode and BS mode under 7-bit input/output resolution.

Enhanced linearity of MAC: During on-chip inference, the MAC ($MACP - MACN$) in every column is computed separately as MACP ($\text{MACP} = \sum_{k=1}^{N} w_k x_k \quad (w_k \geqslant 0)$) and MACN ($\text{MACN} = \sum_{k=1}^{N} w_k x_k \quad (w_k < 0)$), and subsequently input to SA as a differential signal. To determine the maximum voltage drop across RBL in on-chip inference, the distribution of MACP and MACN across all mini-batches in the test set for MNIST is used for the proposed and PWM method for $n_i = 3$). In Fig. 6 (top), the distribution of MACP in the worst-case scenario is depicted, while Fig. 6 (bottom) illustrates the corresponding RBL simulated voltage values,

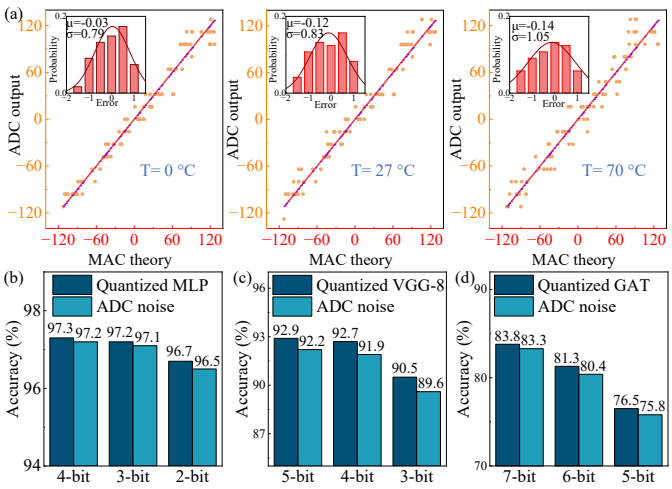


Fig. 5. Simulated results: (a) ADC output vs.theoretical result of MAC at 0 °C, 27 °C and 70 °C. Test results for (b) MLP, (c) VGG-8 and (d) GAT.

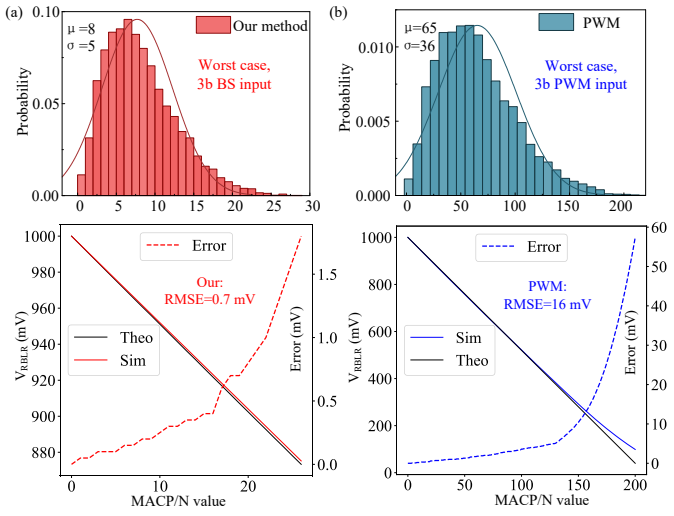


Fig. 6. The partial MACP/N distribution based on 3-bit input for MNIST and its corresponding voltage on RBL: (a) Proposed method. (b) PWM mode.

the theoretical values, and the error between them. Notably, the MACP/N distribution range in the PWM method is $7X$ greater than that in the our method reducing signal margin [18]. Furthermore, our method demonstrates a RBL voltage linearity that is $\approx 23X$ greater than that achieved by the PWM mode, as quantified by the Root Mean Squared Error (RMSE).

Energy and Area efficiency: Fig. 7(a) shows simulated energy breakdown of our work when $n_i = 4$ and weight is 2-bit. Pre-charging and SAs consume most of the energy and the energy consumed by the added voltage buffer is negligible, accounting for less than 2%. The layout of core area is depicted in Fig. 7(b). The core size is 0.24 mm^2 and 127 IMADCs only occupy only 3% of the size of MAC array (8X improvement compared to 23% in the best work [12]). Table I summarizes a performance comparison with the state-of-the-art analog IMC. The proposed macro achieves the best combination of energy/area efficiency (1146 TOPS/W (1.7X

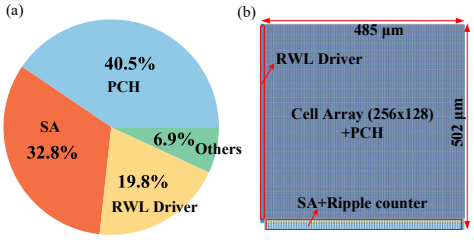


Fig. 7. (a) Energy breakdown (4-bit input/output and 2-bit weight). (b) Core layout of the macro.

TABLE I
COMPARISON WITH STATE-OF-THE-ARTS

	TCASI'24 [15]	DAC'21 [11]	JSSC'20 [12]	Prop.
Tech. (nm)	65	28	65	65
Supp. (V)	0.45/0.8	0.5-1.1	0.6/0.8/1.0	0.5/0.8/1.0
Freq. (MHz)	200	150	50	200
Bitcell	Dual 7T	8T	8T1C	Dual 8T
Accum.	Current	Charge	Charge	Current+Charge
Array size	528×128	256×64	256×64	256×128
I/O bit #	1/1-6	2/5-6	1/5	1-7/1-7
Wgt. bit #	1.6/2.8/3.9	8	1	2-4
ADC type	Ramp (IMADC)	SAR	Flash	Ramp (IMADC)
ADC ovhd.*	100%	137%	23%	3%
Neuro # per ADC	1	8	1	1
GOPS	300-3900	21	1638	14-6502
TOPS/mm^2	0.9-10.4	0.75	20.2	0.06-27
TOPS/W	23.9-259	8.3-36	671.5	11-1146

* ADC ovhd.=ADC area/MAC area

than the best one), 27 TOPS/mm^2 (1.4X than the best one) at 1/2/1b and 14 TOPS/W, 0.06 TOPS/mm^2 at 7/4/7b) among the listed works and other recent publications [13], [23]. It is noteworthy that the proposed macro offers the highest level of reconfigurability in terms of precision for input (1-7b), weight (2-4b), and output (1-7b).

V. Conclusion

This article addresses important issues of IMC: multi-bit input, high linearity of MAC and large overhead of ADC. A dual 8T RWLUDC-based SRAM bitcell has been used to minimize current variations with bitline voltage change enabling larger signal margin. We enable multi-bit (1-7) input using BS with Charge-based accumulation of partial sums and multi-bit (1-7) output using a reconfigurable IMADC that reduces area overhead by shared references. The 256×128 in-memory computing array can achieve high efficiency (1146 TOPS/W, 27 TOPS/mm^2 at 1/2/1b for in/weight/out). Using NRT, it suffers minimal accuracy reductions of 0.1%, 0.8% and 0.5% due to circuit non-idealities for the MLP model (3/2/3b) on MNIST, VGG-8 model (4/2/4b) on CIFAR-10 and GAT network (7/3/7b) on Cora respectively. It increases throughput (1.9X) and linearity (23X) compared to PWM mode by BSCHA. Compared to conventional BS with digital accumulation after ADC, this method has $1.4X/1.7X/6.6X$ better area-efficiency/energy-efficiency/throughput respectively.

REFERENCES

[1] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*. Ieee, 2013, pp. 6645–6649.

[2] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *The International Conference on Learning Representations (ICLR)*, 2021.

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[4] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-SRAM: Enabling in-memory Boolean computations in CMOS static random access memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4219–4232, 2018.

[5] W. Jo, S. Kim, J. Lee, S. Um, Z. Li, and H.-J. Yoo, "A 161.6 tops/w mixed-mode computing-in-memory processor for energy-efficient mixed-precision deep neural networks," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2022, pp. 365–369.

[6] S. Shubham, S. Pandit, K. Prasad, and J. Mekie, "PVC-RAM: Process Variation Aware Charge Domain In-Memory Computing 6T-SRAM for DNNs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–2.

[7] X. Si, J.-J. Chen, Y.-N. Tu, W.-H. Huang, J.-H. Wang, Y.-C. Chiu, W.-C. Wei, S.-Y. Wu, X. Sun, R. Liu *et al.*, "A twin-8T SRAM computation-in-memory unit-macro for multibit CNN-based AI edge processors," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, 2019.

[8] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6T SRAM array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017.

[9] B. Zhang, J. Saikia, J. Meng, D. Wang, S. Kwon, S. Myung, H. Kim, S. J. Kim, J.-S. Seo, and M. Seok, "MACC-SRAM: A Multistep Accumulation Capacitor-Coupling In-Memory Computing SRAM Macro for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, 2023.

[10] K. Roy, I. Chakraborty, M. Ali, A. Ankit, and A. Agrawal, "In-memory computing in emerging memory technologies for machine learning: An overview," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[11] K. Lee, S. Cheon, J. Jo, W. Choi, and J. Park, "A charge-sharing based 8T SRAM In-Memory Computing for edge DNN acceleration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 739–744.

[12] Z. Jiang, S. Yin, J.-S. Seo, and M. Seok, "C3SRAM: An in-memory-computing SRAM macro based on robust capacitive coupling computing mechanism," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 7, pp. 1888–1897, 2020.

[13] C. Yu, T. Yoo, K. T. C. Chai, T. T.-H. Kim, and B. Kim, "A 65-nm 8T SRAM compute-in-memory macro with column ADCs for processing neural networks," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 11, pp. 3466–3476, 2022.

[14] J. Mu, H. Kim, and B. Kim, "SRAM-based in-memory computing macro featuring voltage-mode accumulator and row-by-row ADC for processing neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2412–2422, 2022.

[15] C. Yu, H. Jiang, J. Mu, K. T. C. Chai, T. T.-H. Kim, and B. Kim, "A Dual 7T SRAM-Based Zero-Skipping Compute-In-Memory Macro With 1-6b Binary Searching ADCs for Processing Quantized Neural Networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[16] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W.-S. Khwa, H.-J. Liao, Y. Wang, and J. Chang, "A 351 TOPS/W and 372.4 GOPS compute-in-memory SRAM macro in 7nm FinFET CMOS for machine-learning applications," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2020, pp. 242–244.

[17] S. K. Gonugondla, M. Kang, and N. Shanbhag, "A 42 pJ/decision 3.12 TOPS/W robust in-memory machine learning classifier with on-chip training," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 490–492.

[18] C.-J. Jhang, C.-X. Xue, J.-M. Hung, F.-C. Chang, and M.-F. Chang, "Challenges and Trends of SRAM-based computing-in-memory for AI edge devices," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 5, pp. 1773–1786, 2021.

[19] Allen, P. E and Holberg, D., *CMOS Analog Circuit Design*. Oxford University Press, 2011.

[20] T. Marisa, T. Niederhauser, A. Haeberlin, R. A. Wildhaber, R. Vogel, J. Goette, and M. Jacomet, "Pseudo asynchronous level crossing ADC for ECG signal acquisition," *IEEE transactions on biomedical circuits and systems*, vol. 11, no. 2, pp. 267–278, 2017.

[21] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, Y. Bengio *et al.*, "Graph attention networks," *stat*, vol. 1050, no. 20, pp. 10–48 550, 2017.

[22] H. Jiang, W. Li, S. Huang, and S. Yu, "A 40nm analog-input ADC-free compute-in-memory RRAM macro with pulse-width modulation between sub-arrays," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*. IEEE, 2022, pp. 266–267.

[23] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "XNOR-SRAM: In-memory computing SRAM macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

VQT-CiM: Accelerating Vector Quantization Enhanced Transformer with Ferroelectric Compute-in-Memory

Xuchu Huang[1], Haonan Du[1], Min Zhou[1], Zheyu Yan[1*], Cheng Zhuo[1*] and Xunzhao Yin[1,2*]
[1]Zhejiang University, Hangzhou, China; [2]Key Laboratory of CS&AUS of Zhejiang Province, Hangzhou, China;
[*]Corresponding author email: {zyan2, czhuo, xzyin1}@zju.edu.cn

ABSTRACT

Transformer models have achieved state-of-the-art performance in various natural language processing (NLP) and computer vision (CV) tasks. To meet their substantial computational demands, the compute-in-memory (CiM) architectures, which alleviate the memory wall problem and enable efficient vector-matrix multiplication (VMM), have been adopted for transformer accelerators. However, the dynamic VMM involved in the attention mechanism, which necessitates runtime write operations, presents significant challenges for non-volatile memory (NVM)-based CiM designs. High write overhead, complex compute-write-compute (CWC) dependencies, and limited endurance reduce their effectiveness. In this paper, we propose VQT-CiM, a ferroelectric FET (FeFET)-based CiM design that accelerates vector quantization (VQ) enhanced transformers by eliminating the runtime write operations. VQT-CiM quantizes keys and values in self-attention to convert dynamic VMMs in inner-product and weighted-sum into static VMMs with the codebooks, enabling efficient calculations with CiM crossbars. However, directly applying VQ hinders the accuracy of transformer model due to its limited representation capability. To address this, we introduce a vector quantization scheme that integrates residual VQ (RVQ) and product VQ (PVQ) for enhanced representation space. We present an efficient hardware implementation for the proposed VQT-CiM with optimized dataflow in RVQ, which incorporates the FeFET-based CiM crossbars and peripheral digital circuits. Evaluation results suggest that VQT-CiM achieves the 3.54× and 4.53× improvements in energy efficiency and throughput, respectively, compared to state-of-the-art NVM-based CiM transformer designs.

1 INTRODUCTION

Attention-based transformers [1] have demonstrated remarkable performance in natural language processing (NLP) [2, 3] and computer vision (CV) [4] tasks. Leveraging the global perceptive field enabled by self-attention mechanism, transformers excel at capturing long-range dependencies and complex relationships in data, leading to outstanding modeling capabilities. However, the extensive matrix multiplications involved in transformers, coupled with the quadratic computational complexity with respect to the sequence length, result in significant overhead in data communication and computation. Consequently, the specialized hardware accelerators for transformers have become increasingly essential.

Compute-in-memory (CiM) has emerged as a promising architecture, which alleviates the memory wall problem by integrating memory and computation. Leveraging the high parallelism of crossbar structures, CiM performs vector-matrix multiplication (VMM) with high efficiency, making it well-suited for deep learning tasks. Several studies have proposed CiM implementations as neural network accelerators, primarily categorized into non-volatile memory (NVM)-based [5, 6] and SRAM-based [7, 8] designs.

Different from Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which rely on static VMM between dynamic inputs and static weights, transformers utilize dynamic VMM in self-attention calculations. The inner-product and weighted-sum in self-attention involve VMM between two dynamically generated matrices, leading to runtime write operations to CiM crossbars and complex compute-write-compute (CWC) dependencies. This is especially challenging for NVM devices, which suffer from high writing overhead and limited endurance. To mitigate this issue, researchers have designed SRAM-based [9, 10] and SRAM-NVM hybrid architectures [11], utilizing SRAM's low write overhead and sufficient endurance. However, the 6T-SRAM cell, combined with additional logical gates for multiplication and area-consuming adder trees for accumulation, results in lower storage density compared to NVM-based designs. Moreover, the volatile nature of SRAM limits its use in embedded devices, where data loss occurs when power turns off. The hybrid architectures also introduce design complexity and manufacturing challenges due to the heterogeneous integration, limiting their widespread adoption. Thus, designing fully static solutions that support NVM-based CiM implementations for transformers is of great importance.

A viable solution is to adopt vector quantization (VQ) technique into self-attention mechanism [12–14]. By substituting keys with their closest matches from a pre-trained codebook, Transformer-vq [14] achieves self-attention with linear complexity. Following this approach, the runtime write operations in the inner-product can be avoided, while the dynamically generated values still need to be written into CiM crossbars for weighted-sum. In this paper, we propose VQT-CiM, a ferroelectric FET (FeFET)-based CiM design for VQ-enhanced transformers, which fully eliminates the runtime write operations in self-attention. VQT-CiM employs VQ to both keys and values, converting dynamic VMMs in inner-product and weighted-sum into static VMMs. These static VMMs, which handle similarity measurement with the codebooks and post-substitution self-attention, are efficiently implemented using FeFET crossbars.

One of the main challenges with this approach is model performance degradation due to the limited representation capability of the quantized keys and values. Simply increasing the codebook size does not effectively mitigate this issue and instead raises the associated hardware overhead. To handle this problem, VQT-CiM integrates residual VQ (RVQ) [15] and product VQ (PVQ) [16]. RVQ utilizes a multi-stage VQ approach which refines the residuals between the input vectors and the quantized results, while PVQ employs multi-space VQ for more accurate approximations.

However, RVQ's iterative process, which involves residual calculations and similarity checks at each stage, presents challenges for hardware implementation due to its sequential nature. To address this, VQT-CiM optimizes the dataflow to enable parallelization of RVQ process. By introducing an additional look-up table (LUT) that stores pre-calculated similarity scores between codes, the codebooks across all stages can be accessed simultaneously for parallel

similarity measurements with the input vector. We present the hardware design based on these algorithm optimizations, which includes the FeFET-based CiM crossbars for static VMMs and digital circuits for additional VQ operations, resulting in an efficient and high-performance solution for VQ-enhanced transformers.

The contributions of this work are summarized as follows:

- We eliminate runtime write operations in CiM implementations for transformers by converting the dynamic VMMs into static, codebook-based VMMs through vector quantized keys and values in self-attention.
- We introduce a vector quantization scheme that significantly enhances VQ's representation capability by integrating RVQ and PVQ.
- We propose an efficient hardware design for VQT-CiM with optimized dataflow in RVQ, which comprises FeFET-based CiM crossbars and peripheral digital circuits.
- Evaluation results show that the VQT-CiM achieves 3.54× and 4.53× improvements in energy efficiency and throughput, compared to the state-of-the-art NVM-based CiM transformer designs.

2 BACKGROUND

In this section, we discuss basics of self-attention and related CiM implementations, followed by preliminaries of VQ, FeFET and CiM.

2.1 Self-attention Basics and Existing CiM Implementations

Fig. 1 illustrates the self-attention mechanism [1] in transformers, which consists of three stages: feature transformation, inner-product and weighted-sum. Feature transformation projects the input sequence into queries (Q), keys (K) and values (V) through three distinct weight matrices, W_Q, W_K and W_V. As described in Eq. (1), in the inner-product stage, the attention scores are calculated using $Q \cdot K^T$, capturing the relevance between input tokens. These scores are then normalized using scaled softmax, where d is the feature dimension. In the weighted-sum stage, the attention scores are applied to V to generate the output. Tokens with higher attention scores contribute more to the output Z.

$$Z = Softmax(\frac{Q \cdot K^T}{\sqrt{d}}) \cdot V \tag{1}$$

Several NVM-based CiM implementations for transformers have been proposed [17], [18], [19]. ReTransformer [17] optimizes the dataflow to reduce dynamic write operations to the CiM crossbar. ReBERT [18] mitigates data hazards by applying local attention, focusing on adjacent tokens in the attention score computation. CPSAA [19] achieves enhanced sparsity by computing the attention mask at runtime, merging W_Q and W_K to facilitate mask generation. Despite these advancements, NVM-based CiM designs still face challenges due to the dynamic VMM in self-attention. The high overhead of write operations, coupled with the complicated CWC dependencies, hinder their performance. Furthermore, the endurance issue of NVM devices is often neglected.

2.2 Vector Quantization Basics

Vector Quantization (VQ) [12, 13] maps continuous vectors into discrete codes from a predefined codebook C. As described in Eq.

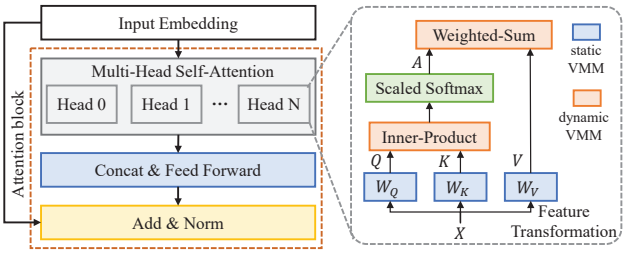


Figure 1: Self-attention block in transformer.

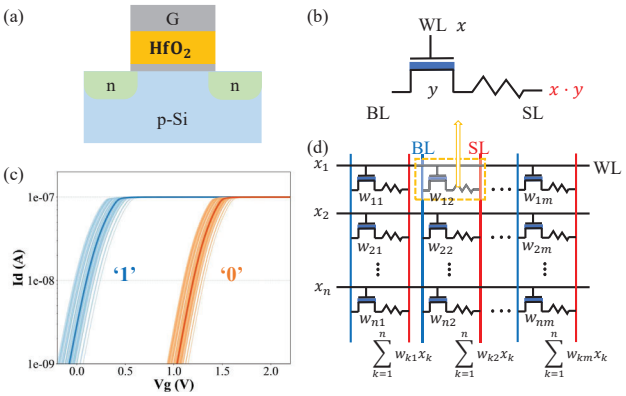


Figure 2: (a) FeFET structure. (b) The 1FeFET1R structure for CiM cell. (c) I_D-V_G curves with the series-connected resistor. (d) 1FeFET1R-based CiM crossbar.

(2), VQ selects code with the highest similarity to represent the input vector. In this paper, we employ the hardware-friendly inner product for similarity measurement, rather than Euclidean distance.

$$z = \underset{i}{argmin}\, ||x - C[i]||^2 \approx \underset{i}{argmax}\, x \cdot C[i] \tag{2}$$

To integrate VQ into DNN training, the straight-through estimator (STE) [13, 20] is employed to address the non-differentiability of the maximum operation in VQ. During forward propagation, the quantized code $C[z]$ is utilized, while in backward propagation, the original gradient is applied, as detailed in Eq. (3), where SG indicates the stop gradient operation.

$$\overline{x} = x + SG(C[z] - x) \tag{3}$$

The codebook is updated using the exponential moving average (EMA) method [21], as described in Eq. (4). In each batch, codes are updated based on the assigned vectors. Here, $n[i]$ represents the number of vectors assigned to code i, and $x[j]$s are the corresponding input vectors. The parameter γ controls the decay rate, determining the update speed of the codebook.

$$\begin{cases} N[i]^{(t)} = N[i]^{(t-1)} * \gamma + n[i]^{(t)} * (1-\gamma) \\ M[i]^{(t)} = M[i]^{(t-1)} * \gamma + \sum_{j}^{n[i]^{(t)}} x[j]^{(t)} * (1-\gamma) \\ C[i]^{(t)} = \frac{M[i]^{(t)}}{N[i]^{(t)}} \end{cases} \tag{4}$$

2.3 FeFET and CiM Basics

FeFET has emerged as a competitive candidate in NVM devices for CiM implementations, owing to its CMOS compatibility, high ON/OFF ratio, and voltage-driven mechanism [22–24]. As illustrated in Fig. 2(a), the HfO_2 layer serves as ferroelectric dielectric

in the gate, allowing for adjustable threshold voltage V_{th}. Different gate pulses are employed to alter the polarization states of HfO_2 to store the data. Specifically, positive gate pulses correspond to Vth_{low} ('1'), while negative gate pulses correspond to Vth_{high} ('0'). Fig. 2(b) shows the 1FeFET-1R structure [25–27], which incorporates a series-connected resistor to mitigate current variation. Simulation results for the 1T1R cell, based on the experimental calibrated FeFET compact model [28], are presented in Fig. 2(c).

For CiM calculations, as described in Fig. 2(b), when applying an input voltage to WL, the read current reflects the multiplication result of input and stored weight. In the crossbar structure of CiM, as depicted in Fig. 2(d), the read currents in SL are naturally summed according to Kirchhoff's Law, enabling efficient multiply-accumulate (MAC) operations [29, 30].

3 ALGORITHM OPTIMIZATION

In this section, we present the algorithm optimization in VQT-CiM. We introduce how VQ is integrated into self-attention in Sec. 3.1, and methods to enhance VQ's representation capacity in Sec. 3.2.

3.1 Self-Attention with VQ

Fig. 3(a) illustrates the dataflow of the self-attention mechanism. The dynamic VMMs in both inner-product and weighted-sum, necessitate runtime write operations for keys and values, leading to significant overhead and complicated CWC dependencies. To address this issue, we propose an algorithm optimization that approximates keys in inner-product and values in weighed-sum using VQ. This approach converts the dynamic VMMs into hardware-friendly static VMMs, as described in Fig. 3(b) and Fig. 3(c).

Key-VQ: The Key-VQ process starts with initializing a codebook $C_k \in R^{C,D}$, where C represents the number of the codes in the codebook, and D denotes the feature dimension. As described in Eq. (5), the similarity scores between keys and codes are calculated, resulting in $D_k \in R^{N,C}$, where N is the number of tokens. The one-hot matrix $\Delta_k \in R^{N,C}$ is then generated through a row-wise maximum operation, setting the position of the highest score in each row to 1, with the rest to 0. The quantized key, $\overline{K}$, which selects the corresponding code based on the one-hot matrix, substitutes K for inner-product calculation, as shown in Eq. (6) and Fig. 3(b).

$$D_k = K \cdot C_k^T \overset{one-hot}{\longrightarrow} \Delta_k \in R^{N,C} \tag{5}$$

$$K \overset{VQ}{\approx} \overline{K} = \Delta_k \cdot C_k \in R^{N,D} \tag{6}$$

To align with the fixed codebooks in CiM crossbars, we reformulate the inner-product calculation as shown in Eq. (7), which computes attention scores between Q and codebook C_k instead of the original $\overline{K}$. As illustrated in Fig. 3(c), the optimized dataflow achieves reduced latency in self-attention by enabling parallel computation of D_q and D_k. This approach also converts the original dynamic VMM in inner-product into two static VMMs with codebook C_k, which can be efficiently implemented with FeFET crossbars. The additional VQ operations, including the row-wise maximum for code selection and reshaping $Q \cdot C_k^T$ based on the one-hot matrix, are handled by a digital circuit with minimal hardware overhead.

$$\begin{aligned} A = Softmax(S) &\approx Softmax(Q \cdot \overline{K}^T) \\ = Softmax(Q \cdot (C_k^T \cdot \Delta_k^T)) &= Softmax((Q \cdot C_k^T) \cdot \Delta_k^T) \\ = Softmax((Q \cdot C_k^T) \cdot Onehot(K \cdot C_k^T)^T) & \end{aligned} \tag{7}$$

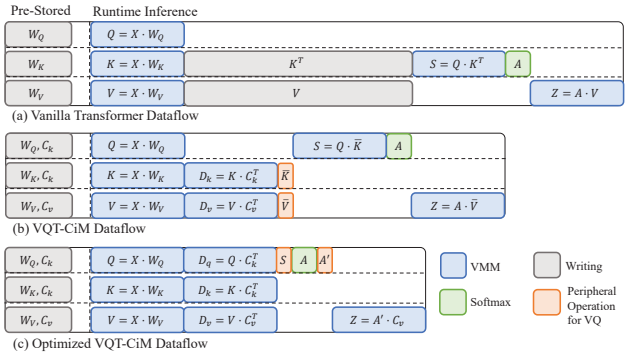


Figure 3: (a) Dataflow for vanilla self-attention. (b) Dataflow of VQT-CiM. (c) Optimzied dataflow of VQT-CiM.

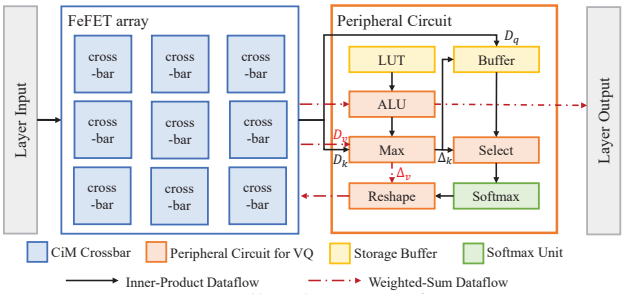


Figure 4: Overall architecture of VQT-CiM.

Value-VQ: Similarly, the Value-VQ process utilizes the codebook $C_v \in R^{C,D}$ for quantization. Eq. (8) describes the similarity scores calculation and row-wise maximum operation for code selection. Using the one-hot matrix Δ_v, V is approximated by $\overline{V}$ in the weighted-sum calculation by picking the most similar code from the codebook, as shown in Eq. (9) and Fig. 3(b).

$$D_v = V \cdot C_v^T \overset{one-hot}{\longrightarrow} \Delta_v \in R^{N,C} \tag{8}$$

$$V \overset{VQ}{\approx} \overline{V} = \Delta_v \cdot C_v \in R^{N,D} \tag{9}$$

To facilitate CiM implementations, the weighted-sum with Value-VQ is reformulated, as shown in Eq. (10) and Fig. 3(c), converting the dynamic VMM into two static VMMs based on the codebook C_v. It's important to note that the attention scores A, obtained from the inner-product step, are firstly transformed using the one-hot matrix Δ_v before participating in the static VMM. As with the hardware implementations for Key-VQ, the static VMMs are executed through FeFET crossbars, while the extra operations introduced by VQ are handled by digital circuits with negligible overhead.

$$\begin{aligned} Z = A \cdot V &\approx A \cdot (\Delta_v \cdot C_v) = (A \cdot \Delta_v) \cdot C_v \\ &= A' \cdot C_v = (A \cdot Onehot(V \cdot C_v^T)) \cdot C_v \end{aligned} \tag{10}$$

3.2 VQ Representation Capacity Enhancement

To mitigate the approximation error between the original vector and its quantized result, RVQ and PVQ are utilized to enhance the codebook's representation capacity. We employ RVQ and PVQ to both keys and values, as will be discussed in Sec. 4.2 and Sec. 4.3.

Residual Vector Quantization (RVQ): RVQ applies a multi-stage VQ process, where each stage sequentially refines the approximation by quantizing the residuals from the previous stage. Distinct codebooks are assigned to each VQ layer, denoted as $\{C_1, C_2, \ldots, C_R\}$,

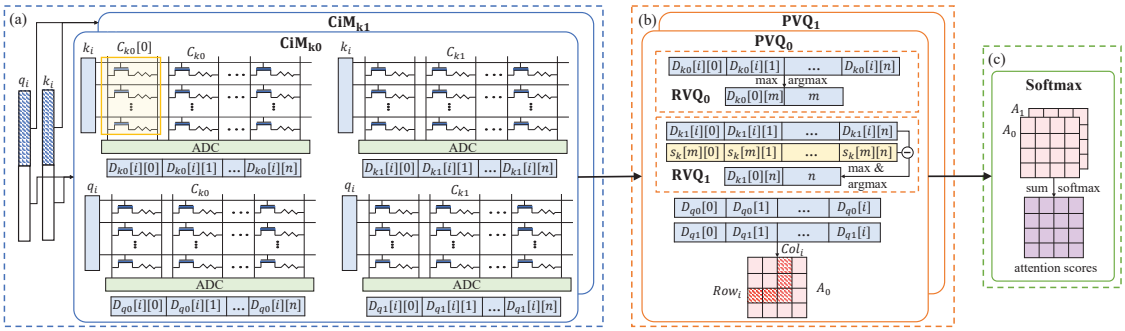


Figure 5: Hardware implementation of inner-product part in self-attention. (a) Static VMM with the codebook stored in FeFET crossbars. (b) Peripheral operations required by VQ. (c) Softmax for attention score normalization.

where R represents the number of layers. In the first layer, the input vector x is quantized using the codebook C_1, producing the code index z_1. The residual, defined as the difference between the input vector x and the quantized result $C_1[z_1]$, is then passed to the next VQ layer for further quantization, as described in Eq. (11). This process continues iteratively across all VQ layers, progressively reducing the residual error. As expressed in Eq. (12), the input vector x is ultimately approximated as the sum of the quantized vectors from all R VQ layers. Through this approach, RVQ substantially improves flexibility and expressiveness by combining codes across different codebooks. With C codes in each codebook, RVQ achieves the representation capacity of $O(C^R)$.

$$\begin{cases} z_1 = \underset{i}{argmax}\, x \cdot C_1[i] \\ z_k = \underset{i}{argmax}(x - \sum_{j=1}^{k-1} C_j[z_j]) \cdot C_k[i], 1 < k \le R \end{cases} \tag{11}$$

$$x \approx C_1[z_1] + C_2[z_2] + ... + C_R[z_R] \tag{12}$$

Product Vector Quantization (PVQ): PVQ divides the input vector into several shorter subvectors to achieve more precise quantization. This process is expressed as $x = concatenate(x_1, x_2, \ldots, x_P)$, where P is the number of subvectors. Each subvector x_i is assigned its own codebook, $\{C_1, C_2, \ldots, C_P\}$, and quantized independently, as described in Eq. (13). By concatenating the VQ results from each subspace, the final quantized vector is obtained, as shown in Eq. (14). PVQ significantly expands the representation space to $O(C^P)$ by combining the codes from different subspaces, where C represents the number of codes in each codebook.

$$z_i = \underset{j}{argmax}\, x_i \cdot C_i[z_j] \tag{13}$$

$$x \approx concatenate(C_1[z_1], C_2[z_2], ..., C_P[z_P]) \tag{14}$$

4 HARDWARE DESIGN

In this section, we introduce the hardware design of VQT-CiM, including overall architecture in Sec. 4.1, hardware implementation of inner-product part in Sec. 4.2 and weighted-sum part in Sec. 4.3.

4.1 VQT-CiM Overview

Fig. 4 illustrates the overall architecture of VQT-CiM, which consists of two main parts, the FeFET arrays and the peripheral circuits. The FeFET arrays contain CiM crossbars, which perform static VMM for transformer calculations. The peripheral circuits comprise the ALU, Max Unit, Select Unit, and Reshape Unit for additional operations required by VQ, along with the LUT and Buffer for data storage, and the Softmax Unit for attention normalization. The Buffer stores the D_q and Δ_k, which are dynamically generated to represent the similarities and VQ results, respectively. The LUT holds pre-calculated similarity scores between codes from different codebooks, participating in the RVQ process. The dataflows of inner-product and weighted-sum in self-attention are depicted in the black solid arrow and red dash-dot arrow in Fig. 4, respectively.

4.2 Inner-Product Design

The hardware implementation of the inner-product part in self-attention is detailed in Fig. 5, with both RVQ and PVQ employed to enhance the model's representation capacity. For simplicity, the parameters R in RVQ and P in PVQ are both set to 2.

In the PVQ process, the inputs k_i and q_i are divided into two segments and sent to CiM$_{k0}$ and CiM$_{k1}$ for separate computations, as described in Fig. 5(a). To mitigate the significant latency in RVQ, which relies on sequential residual calculations and multi-stage quantization, we reformulate RVQ process as expressed in Eq. (15). The second term in this formulation, representing the similarity between codes from different codebooks, is pre-calculated and stored in the LUT, with s_{jk} denoted for similarity scores between the codebook j and k. This approach enables parallel processing with the different codebooks in RVQ, specifically C_{k0} and C_{k1} in Fig. 5(a).

$$\begin{aligned} z_k &= \underset{i}{argmax}\, x \cdot C_k[i] - (\sum_{j=1}^{k-1} C_j[z_j]) \cdot C_k[i] \\ &= \underset{i}{argmax}\, x \cdot C_k[i] - (\sum_{j=1}^{k-1} s_{jk}[z_j][i]) \end{aligned} \tag{15}$$

Fig. 5(b) illustrates the peripheral operations required for VQ. The maximum operation is utilized in RVQ$_0$ and RVQ$_1$ to identify the most similar vector for representation. In the residual layer RVQ$_1$, as specified in Eq. (15), the similarity scores D_{k1} are adjusted by subtracting the pre-calculated value $s_k[m]$ from the LUT, before applying the maximum operation. Since the attention score $A[i][j]$ cannot be determined until both q_i and k_j have participated in the static VMM with the codebook, the i-th row and column of the attention score matrix are obtained during the i-th cycle, by selecting the corresponding value in D_q, as described in Eq. (16). As a result, D_q, along with the maximum positions m and n, need to be stored in the buffer for future updates of attention scores.

$$\begin{cases} A[i,j] = D_{q0}[i][m[j]] + D_{q1}[i][n[j]], & j \le i \\ A[j,i] = D_{q0}[j][m[i]] + D_{q1}[j][n[i]], & j \le i \end{cases} \tag{16}$$

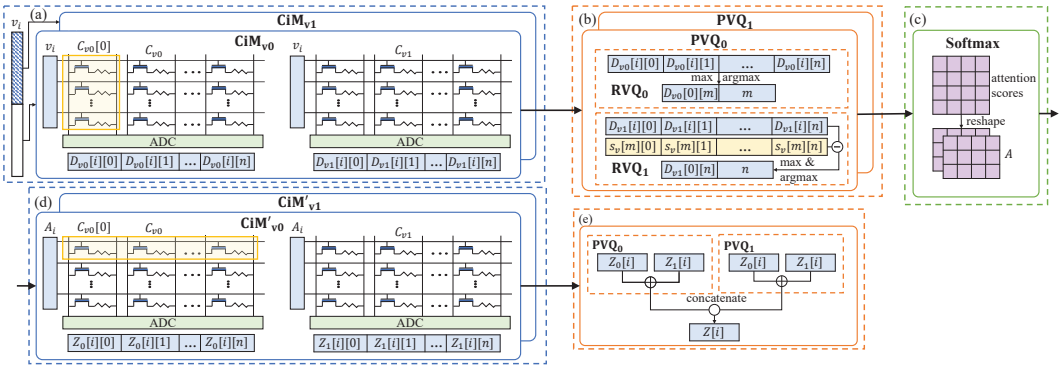


Figure 6: Hardware implementation of weighted-sum part in self-attention. (a) Static VMM with the codebook stored in FeFET crossbars. (b) Peripheral operations required by VQ. (c) Reshape the attention scores. (d) Static VMM with the codebook for weighted-sum. (e) Summation for RVQ and concatenation for PVQ.

In Fig. 5(c), the partial attention scores from different PVQ subspaces are summed and then executed by softmax unit for normalization. Apart from the conventional static VMM in the inner-product, the peripheral operations introduced by VQ lead to minimal hardware overhead. The additional operations, including maximum selection, subtraction, and summation, can be easily implemented by digital circuits. Furthermore, the LUT and buffer incur the negligible area overhead.

4.3 Weighted-Sum Design

Fig. 6 illustrates the hardware design of the weighted-sum part in self-attention, utilizing both RVQ and PVQ. The Value-VQ process, depicted in Fig. 6(a) and (b), is similar to the Key-VQ process described in Sec. 4.2, which involves splitting the input vector for PVQ, performing static VMM in the FeFET crossbar with the codebooks in RVQ, and applying peripheral operations. In Fig. 6(c), after selecting the most representative vector for value from the codebook, the attention scores obtained from the inner-product part are reshaped, corresponding to the role of Δ_v in Eq. (10). Since Δ_v varies across different subspaces in PVQ, the attention scores are replicated and reshaped accordingly, and then processed separately in Fig. 6(d) through CiM$'_{v0}$ and CiM$'_{v1}$ for static VMM in weighted-sum. Fig. 6(e) details the additional operations performed after obtaining the weighted-sum results, including the summation across different VQ layers in RVQ and the concatenation of different subspaces in PVQ.

Consistent with the analysis in Sec. 4.2, the additional operations introduced by the Value-VQ process, such as summation, concatenation, and reshaping, add minimal hardware overhead.

5 EVALUATION

In this section, we evaluate the proposed VQT-CiM at both algorithm and hardware level. The experimental setup is illustrated in Sec. 5.1, followed by performance analysis for algorithm and hardware in Sec. 5.2 and Sec. 5.3, respectively.

5.1 Experimental Setup

At the algorithm level, we evaluate VQT-CiM on BERT [3] with five datasets, including GLUE [31] (MRPC, QNLI), SQUAD-1.1 [32], SQUAD-2.0 [33] and IMDb [34]. We fine-tune the pre-trained BERT-base model from the Huggingface library [35]. The default sequence lengths are set to 128 for GLUE, 384 for SQUAD, and 512 for IMDb,

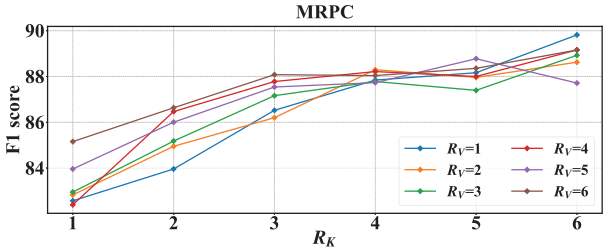


Figure 7: Model performance on MRPC under different configurations for RVQ.

with a learning rate of 2e-5 and a batch size of 32. For VQ implementations, we employ STE to address the non-differentiability in gradient backpropagation. The codebooks are initialized with k-means clustering and updated using EMA with a decay rate of 0.99. We quantize the linear transformation, inner-product, and weighted-sum in self-attention to 8 bits, employing quantization-aware training in PyTorch to maintain model performance.

At the hardware level, we evaluate the FeFET-based CiM implementations through NeuroSim [36]. VQT-CiM converts the dynamic VMM in self-attention into static VMM through VQ, making it highly compatible with NeuroSim. We set the subarray size to 32×32 and the technology node to 32nm. The read latency and ON/OFF resistances are obtained from the simulation results of Preisach FeFET model [28] through Cadence Spectre. The softmax operations are implemented using the approximation methods in I-ViT [37] and block-wise computation in SOLE [38], along with the digital peripheral circuits introduced by VQ. We estimate the energy and latency of these digital circuits with Synopsys Design Compiler through the 40nm library of UMC.

5.2 Algorithm Performance

Impact of VQ configurations. We investigate the effect of VQ configurations on model performance, focusing on the number of VQ layers in RVQ. Specifically, we explore R_K and R_V in the Key-VQ and Value-VQ processes, while keeping the codebook size C in each layer at 32 and the number of subvectors in PVQ at 2. Fig. 7 illustrates the F1 scores for the MRPC task across various configurations. The results indicate that the model performance on MRPC is more influenced by R_K than by R_V. When R_K is below 3, increasing either R_K or R_V enhances performance, with R_K having

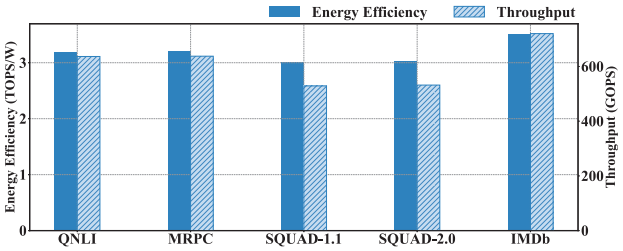


Figure 8: Energy efficiency and throughput of VQT-CiM.

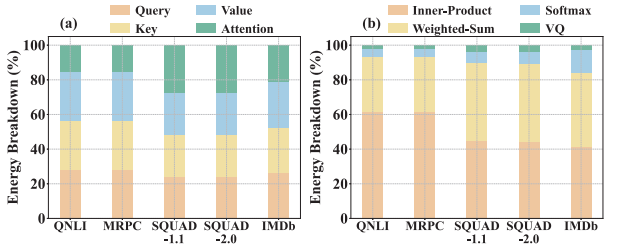


Figure 9: Energy breakdown of VQT-CiM. (a) Multi-head self-attention, including feature transformation and attention mechanism. (b) Attention mechanism, including inner-product, weighted-sum, softmax, and VQ.

a more pronounced impact. When R_K exceeds 3, VQT-CiM achieves comparable performance with the baseline, while further increasing R_K and R_V may introduce quantization noise that degrades performance. Considering the trade-off between model performance and hardware overhead, we set R_K to 4 and R_V to 2 for MRPC, leading to a 0.6% decrease in the F1 score.

Algorithm Performance Comparison. The selected configurations for various tasks and their corresponding algorithm performances are summarized in Tab. 1. For GLUE tasks with short sequence length, we set R_K to 4 and R_V to 2. For SQUAD and IMDb tasks, which involve longer sequences, we choose R_K/R_V values of 6/6 for SQUAD-1.1 and SQUAD-2.0, and 4/4 for IMDb. Under these configurations, VQT-CiM achieves comparable performance to the baseline, with an average accuracy drop of 0.8%. The quantization in feature transformation, inner-product, and weighted-sum results in a negligible average accuracy drop of 0.3%, compared to the floating-point version.

5.3 Hardware Evaluation

Hardware Performance Comparison. Fig. 8 illustrates the energy efficiency and throughput of VQT-CiM across various datasets, which include feature transformation, inner-product, weighted-sum, and softmax in multi-head self-attention. For fair comparisons, the performance metrics of ReBERT [18], ReTransformer [17], and CPSAA [19] are obtained from their corresponding NeuroSim implementations [39]. As listed in Tab. 2, VQT-CiM achieves an average energy efficiency of 3.19 TOPs/W and a throughput of 611.1 GOPs across five datasets, demonstrating improvements of 3.54×, 13.86×, 14.49× in energy efficiency, and 4.53×, 5.64×, 5.01× in throughput, compared to ReBERT, ReTransformer and CPSAA, respectively. VQT-CiM outperforms state-of-the-art NVM-based CiM designs primarily due to the elimination of runtime write operations. By removing the high overhead of write operations and

Table 1: Model performance of VQT-CiM and comparison with the baseline

Dataset	QNLI	MRPC	SQUAD-1.1	SQUAD-2.0	IMDb
Average Length	50.1	52.2	174.9	178.7	271.2
Baseline	90.7	88.9	87.8	77.0	93.0
VQT-CiM	90.4	88.3	86.4	76.0	92.4
Quantized VQT-CiM	89.8	86.8	86.2	76.6	92.5
R_K/R_V	4/2	4/2	6/6	6/6	4/4
Average Codes	96	96	192	192	128

Table 2: Energy efficiency and throughput comparison with state-of-the-art NVM-based CiM transformer designs

	ReTransformer	ReBERT	CPSAA	VQT-CiM
Technology	32nm	32nm	32nm	32nm
Implementation	ReRAM-CiM	ReRAM-CiM	ReRAM-CiM	FeFET-CiM
Energy Efficiency (TOPS/W)	0.23	0.90	0.22	3.19
Throughput (GOPs)	108.16	134.93	121.92	611.1

alleviating the complicated CWC dependencies, VQT-CiM reduces both energy efficiency and latency.

Energy Breakdown. Fig. 9(a) demonstrates the energy breakdown of multi-head self-attention in VQT-CiM, with energy consumption divided into query, key, value and attention. Different from the quadratic complexity in vanilla transformer, VQT-CiM achieves linear complexity through VQ, making the energy consumption of the attention part proportional to codebook size. Align with the number of VQ layers in Tab. 1, the attention part accounts for 15.4%, 27.2%, 21.2% of the total energy consumption in GLUE, SQUAD, IMDb, respectively. We further analyze the attention part in Fig. 9(b), breaking it down into inner-product, weighted-sum, softmax and VQ. Since the total number of codes does not scale proportionally with the sequence length in SQUAD and IMDb, the energy consumption ratio for inner-product and weighted-sum is reduced. Consequently, softmax part occupies a larger proportion of overall energy consumption, ranging from 4.7% in GLUE to 13.6% in IMDb. The VQ part, which primarily involves simple arithmetic operations such as addition and subtraction, consumes a relatively small portion of the total energy, ranging from 2.1% to 3.8%.

6 CONCLUSION

In this paper, we introduce VQT-CiM, a FeFET-based CiM design for transformer, leveraging VQ to address the challenges associated with dynamic VMM in self-attention mechanisms. We utilize Key-VQ and Value-VQ to convert the dynamic VMM in inner-product and weighted-sum into static VMM with pretrained codebooks, while incurring negligible hardware overheads. To maintain the model performance, we integrate RVQ and PVQ to significantly enhance the representation capacity of the codebooks and minimize the discrepancy between the input vector and its quantized result. The dataflow in RVQ is optimized for parallelization. Experiment results show that VQT-CiM achieves 3.54× and 4.53× improvements in energy efficiency and throughput, respectively, compared to state-of-the-art NVM-based CiM transformer accelerators.

ACKNOWLEDGMENT

This work was partially supported by Zhejiang Provincial Key R&D Program (2025C01071), NSFC (92164203) and SGC Cooperation Project (Grant No. M-0612).

REFERENCES

[1] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[2] A. Radford *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[3] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[4] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[5] S. Yin *et al.*, "High-throughput in-memory computing for binary deep neural networks with monolithically integrated rram and 90-nm cmos," *IEEE Transactions on Electron Devices*, vol. 67, no. 10, pp. 4185–4192, 2020.

[6] X. S. Hu *et al.*, "In-memory computing with associative memories: a cross-layer perspective," in *2021 IEEE IEDM*, 2021, pp. 25–2.

[7] A. Biswas *et al.*, "Conv-ram: An energy-efficient sram with embedded convolution computation for low-power cnn-based machine learning applications," in *2018 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2018, pp. 488–490.

[8] X. Si *et al.*, "A twin-8t sram computation-in-memory unit-macro for multibit cnn-based ai edge processors," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, 2019.

[9] F. Tu *et al.*, "Trancim: Full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 6, pp. 1798–1809, 2022.

[10] F. Tu *et al.*, "Multcim: Digital computing-in-memory-based multimodal transformer accelerator with attention-token-bit hybrid sparsity," *IEEE Journal of Solid-State Circuits*, 2023.

[11] S. Liu *et al.*, "Hardsea: Hybrid analog-reram clustering and digital-sram in-memory computing accelerator for dynamic sparse self-attention in transformer," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[12] R. Gray, "Vector quantization," *IEEE Assp Magazine*, vol. 1, no. 2, pp. 4–29, 1984.

[13] A. Van Den Oord *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[14] L. D. Lingle, "Transformer-vq: Linear-time transformers via vector quantization," *arXiv preprint arXiv:2309.16354*, 2023.

[15] N. Zeghidour *et al.*, "Soundstream: An end-to-end neural audio codec," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 495–507, 2021.

[16] P. Stock *et al.*, "And the bit goes down: Revisiting the quantization of neural networks," *arXiv preprint arXiv:1907.05686*, 2019.

[17] X. Yang *et al.*, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[18] M. Kang *et al.*, "A framework for accelerating transformer-based language model on reram-based architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 3026–3039, 2021.

[19] H. Li *et al.*, "Cpsaa: Accelerating sparse attention using crossbar-based processing-in-memory architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[20] Y. Bengio *et al.*, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[21] A. Razavi *et al.*, "Generating diverse high-fidelity images with vq-vae-2," *Advances in neural information processing systems*, vol. 32, 2019.

[22] X. Yin *et al.*, "A homogeneous fefet-based time-domain compute-in-memory fabric for matrix-vector multiplication and associative search," *IEEE TCAD*, 2024.

[23] K. Ni *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[24] X. Yin *et al.*, "Deep random forest with ferroelectric analog content addressable memory," *Science Advances*, vol. 10, p. eadk8471, 2024.

[25] H. Xu *et al.*, "On the challenges and design mitigations of single transistor ferroelectric content addressable memory," *IEEE EDL*, 2023.

[26] T. Soliman *et al.*, "Ultra-low power flexible precision fefet based analog in-memory computing," in *2020 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 2020, pp. 29–2.

[27] X. Yin *et al.*, "An ultracompact single-ferroelectric field-effect transistor binary and multibit associative search engine," *Advanced Intelligent Systems*, vol. 5, no. 7, p. 2200428, 2023.

[28] K. Ni *et al.*, "A circuit compatible accurate compact model for ferroelectric-fets," in *2018 IEEE symposium on VLSI technology*. IEEE, 2018, pp. 131–132.

[29] Z. Yang *et al.*, "Energy efficient dual designs of fefet-based analog in-memory computing with inherent shift-add capability," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[30] X. Yin *et al.*, "Ferroelectric compute-in-memory annealer for combinatorial optimization problems," *Nature Communications*, vol. 15, no. 1, p. 2419, 2024.

[31] A. Wang, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[32] P. Rajpurkar, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[33] P. Rajpurkar *et al.*, "Know what you don't know: Unanswerable questions for squad," *arXiv preprint arXiv:1806.03822*, 2018.

[34] A. Maas *et al.*, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, 2011, pp. 142–150.

[35] T. Wolf *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.

[36] X. Peng *et al.*, "Dnn+ neurosim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *2019 IEEE international electron devices meeting (IEDM)*. IEEE, 2019, pp. 32–5.

[37] Z. Li *et al.*, "I-vit: Integer-only quantization for efficient vision transformer inference," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 17 065–17 075.

[38] W. Wang *et al.*, "Sole: Hardware-software co-design of softmax and layernorm for efficient transformer inference," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[39] X. Yu *et al.*, "Aesha: Accelerating eigen-decomposition-based sparse transformer with hybrid rram-sram architecture," in *2024 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2024, pp. 1–9.

A Memory-Efficient LLM Accelerator with Q-K Correlation Prediction using Cluster-Based Associative Array for Selective KV Accessing

Zikang Zhou, Kaiqi Chen, Xuyang Duan, Jun Han*
State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China
Email: *junhan@fudan.edu.cn

***Abstract*—Attention-based LLMs excel in text generation but face redundant computations in autoregressive token generation. While KV cache mitigates this, it introduces increased memory access overhead as sequences grow. We propose Sella, a hardware-software co-design using cluster-based associative arrays to predict Q-K correlations, enabling selective KV cache access and reducing memory access without retraining. Sella includes a specialized accelerator featuring a prediction engine to improve performance and energy efficiency. Experiments show Sella achieves 2.1×, 93.8×, 31.4×, and 53.5× speedup over SpAtten, Sanger, TITAN RTX GPU, and Xeon CPU, respectively, reducing off-chip memory access by up to 66% with negligible accuracy loss.**

***Index Terms*—Large Language Models, KV Cache, Accelerator**

I. Introduction

Large language models (LLMs) represent a significant technological breakthrough in the new era of artificial intelligence (AI) development. Models such as GPT-4 [1], Llama [2], and OPT [3] have become outstanding examples of transformer architecture in the field of natural language processing (NLP). LLMs have enabled numerous innovative applications, including virtual assistants, text generation, and intelligent search.

The prevalent LLMs are built on stacked decoder architectures and characterized by autoregressive inference shown in Fig.1(a) [4]. However, autoregressive inference introduces a problem: during each attention calculation, the keys and values (KVs) of previous tokens must be recomputed to retrieve contextual information, resulting in quadratic computational complexity relative to the sequence length.

KV cache is proposed to eliminate KVs' recomputation by caching all previous KVs. However, it introduces memory access proportional to the sequence length. Fig.1(b) shows that as token generation progresses, access to the KV cache gradually increases, eventually surpassing that of weights and becoming dominant. The *baseline* in Fig.1(c) illustrates that the bandwidth demand for accessing the KV cache also gradually increases to ensure inference latency. Although HBM (high-bandwidth memory) is available, the KV cache bandwidth demand will eventually exceed what HBM can provide, becoming unacceptable and leading to a decline in token generation speed in practical applications.

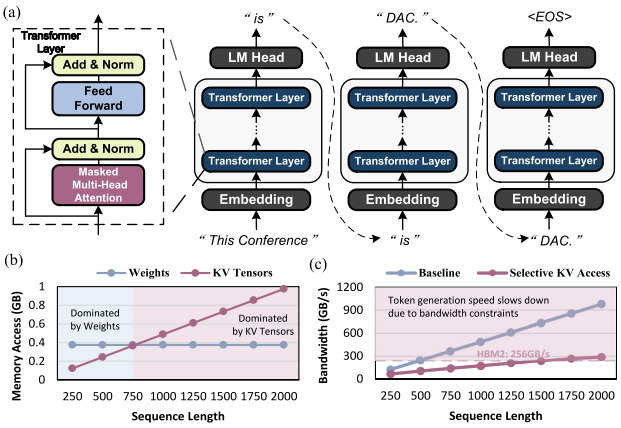


Fig. 1. Figure (a) shows the inference process of LLMs. Figures (b) and (c) show how memory access for Weights and KVs, as well as the bandwidth requirement for accessing the KV cache, change as the sequence length increases during inference for a single-layer decoder in OPT-6.7b with a batch size of 32. In Figure (c), the baseline represents the original bandwidth requirement, while selective KV access indicates the bandwidth requirement after applying the proposed technique.

A straightforward idea to reduce KV cache memory access is to load only the historical KVs strongly correlated with the current query for computation to alleviate the performance bottleneck under long context conditions. This is viable because prior research has demonstrated that not all preceding tokens contribute to generating the current token [5]–[10]. However, when these methods are applied to LLMs with KV cache, they face the following issues: high prediction overhead, inability to identify important KVs during token generation, and expensive retraining.

To address these issues, in this work, we propose a hardware-software co-designed solution for efficient KV accessing during token generation for LLMs, called Sella. Sella can reduce memory access and bandwidth requirements, as shown by the *selective KV access* in Fig.1(c). Note that although this paper primarily focuses on memory access, this method also reduces the attention computation. The contribu-

*Corresponding author.
This work was supported by the National Natural Science Foundation of China under Grant 61934002 and 62234008.

tions of this work are summarized as follows:

- We introduce a low-overhead, high-accuracy mechanism for predicting Q-K correlations using cluster-based associative arrays to selectively access the KV cache. This approach maintains tables that record correlation scores for K and approximated Q and determines important KVs dynamically during token generation through table lookups. Weights used to calculate the correlation scores are calibrated offline using clustering. This method ensures accuracy without costly retraining.
- We propose a high-performance, energy-efficient specialized LLM accelerator for text generation. This accelerator consists of a prediction engine and a computation engine. The prediction engine looks up and updates the associative array to predict significant KVs. The computation engine performs self-attention and is fully pipelined, achieving high throughput.
- We evaluated Sella across a wide range of LLMs and tasks. Experiments demonstrate that Sella significantly reduces memory access and inference time without significant accuracy loss, and decreases energy consumption.

II. Background and Motivations

A. Large Language Models

Autoregressive inference. Autoregressive inference means generating each token sequentially, using each newly generated token with previously generated tokens as input to predict the next token. This process repeats until it reaches the model's max length or encounters an end-of-sequence token (EOS). It guarantees that each newly produced token relies on the existing context.

Attention and KV cache. The attention mechanism is the key component of LLMs, it computes the dependencies of each token on other tokens in the input sequence. It involves three tensors: Q (Query), K (Key), and V (Value). First, the Q and K tensors undergo an inner product, scaled by the square root of hidden dimension, and passed through a *softmax* to get the attention weights. Next, the attention weight is multiplied by tensor V to complete the attention computation [11].

Due to the autoregressive inference of LLMs, it is necessary to recompute all previous KVs when generating tokens for attention calculation. KV cache avoids recomputation by caching the previously computed K and V vectors. The computation flow is as follows: during the generation of each new token x_t, the corresponding k_t and v_t are calculated and stored in K_{cache} and V_{cache}, where K_{cache} is $\{k_1, ..., k_t\}$ and V_{cache} is $\{v_1, ..., v_t\}$. When generating a new token, the current token's q_{t+1}, k_{t+1}, and v_{t+1} are computed, and the attention computation is performed using the K_{all} and V_{all}, where $K_{all} = \{K_{cache}, k_{t+1}\}, V_{all} = \{V_{cache}, v_{t+1}\}$.

B. Motivations

Attention weights are sparse, and skipping KVs with weak dependencies has little impact on accuracy, as shown in Table I. However, predicting the important KVs before the actual attention computation is a challenge. Many previous works try to predict the sparsity of attention weights to reduce attention computation [5]–[9]. However, these methods have the following limitations when applied to LLMs with KV cache. A^3 [5] only reduces computational complexity but still requires all K vectors to participate in the calculation, thus cannot reduce memory access complexity. SpAtten [6] requires storing the entire attention weights for KV pruning decisions. It is only able to prune prompt tokens and cannot dynamically decide important KVs during the token generation. The predictions of Sanger [7] and DOTA [9] still require the inner product between each simplified Q and each simplified K to determine important KVs, leading to significant computational and memory access overhead during prediction. Moreover, both SpAtten [6] and DOTA [9] require retraining the networks, which is very expensive and not scalable for LLMs.

TABLE I
The accuracy and sparsity achieved by pruning KVs associated with attention weights under 0.001

Network	Dataset	Dense Accuracy	Skipping Accuracy	KV Skipping Rate
Llama2-7b	COPA-16shots	0.920	0.920	0.781
	PIQA-4shots	0.781	0.786	0.769
Opt-6.7b	COPA-16shots	0.810	0.810	0.625
	PIQA-4shots	0.768	0.762	0.616

We expect to find a method to predict the important KVs before accessing the KV cache, with low overhead, dynamic prediction while token generation, and no need for costly retraining. We are inspired by clustering approximation. By using cluster centroids to approximate nearby vectors, calculating the inner product for every Q and K can be avoided. The next section will introduce the implementation using clustering approximation.

III. Cluster-Based Associative Array for Selective KV Accessing

A. Clustering Approximation for Queries

Fig. 2 shows the process of clustering approximation. There are three query vectors: Q_1, Q_2, and Q_3, and two cluster centroid vectors, H_1 and H_2 (both are unit vectors). Among them, Q_1 and Q_2 are close to H_1 (γ_{11} is small), while Q_3 is close to H_2 (γ_{32} is small). When performing dot product, the cosine values of the angle between other K vectors and Q_1 or Q_2 can be approximated as the cosine values of other K vectors and H_1; similarly, the cosine values between other K vectors and Q_3 can be approximated as the cosine values of K vectors and H_2. During inference, the projection results of K vectors can be reused for different Q vectors to reduce computational costs (e.g., Q_1 and Q_2 in the figure reuse the $h_1(K_n)$ results), thus reducing the computation and relevant memory access.

B. Cluster-Based Associative Array

Based on clustering approximation, we implemented the associative array structure to reduce KV memory access during the LLM token generation process and accelerate inference.

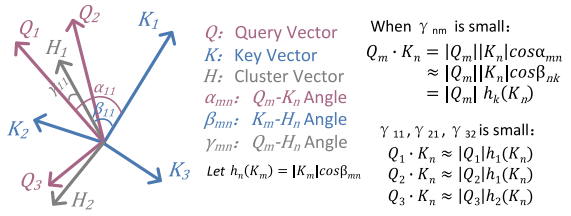


Fig. 2. The example of clustering approximation for KV dot-product.

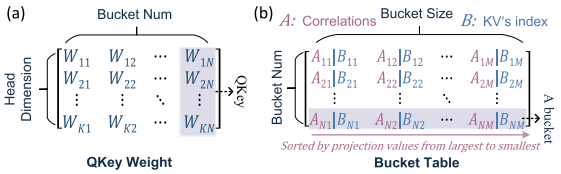


Fig. 3. The QKey weight and bucket table structure.

The associative array's keys (denoted as **QKeys**) are a set of centroids obtained by offline clustering of Q vectors. The associative array's values are finite-sized lists (denoted as **buckets**), which contain the several largest inner product values between QKeys and K, along with the corresponding indexes of K. The matrix formed by QKeys is denoted as **QKey weight**, and the array composed of buckets is denoted as **bucket table**. During inference, the Q vector is compared with the QKeys to find the best-matching bucket, and the KVs referenced in the bucket are used for computation. Next, we will introduce the components of the cluster-based associative array and its overall workflow.

1) Bucket Table Structure: Fig. 3(b) illustrates the bucket table structure used in this method. Each attention head in each layer maintains a bucket table. An element in the table takes 4 bytes and consists of two parts: the first few large projection values of the cached K onto the corresponding QKey (predicted Q-K correlations), and the corresponding token index. Larger predicted Q-K correlations indicate that the corresponding KVs are more important for this QKey. The number of rows in the table is denoted as BN (Bucket Num), representing the number of QKeys. Each row in the table is arranged in descending order of correlations. The number of columns in the table is denoted as BS (Bucket Size), which indicates the number of recorded elements in each bucket. BS grows proportionally as sequence length increases, which is $BS = \alpha * sequence_length$. Although the bucket table needs to be read and updated during each inference, this overhead is negligible compared to the saved memory access.

2) QKey Weight Calibration: Fig. 3(a) shows the QKey weight structure. Unlike the bucket table, all attention heads in each layer share a single QKey weight. The QKey weight is calibrated by clustering the Q vectors offline using the K-Means algorithm instead of retraining. First, we randomly sample data from the dataset for inference and record the Q vectors generated at each layer during the inference. Next, we set BN and perform clustering on the Q vectors generated by each layer, getting BN cluster centroids, thereby forming a QKey weight of size $BN*HD$.

3) Accuracy Compensation and Exception Handling: For a Q vector, once the best-matching bucket is found, a simple and effective method is retrieving all the KVs recorded in that bucket. However, two situations can lead to accuracy issues. 1. Even if the Q vector is close to the nearest QKey, due to the relatively small BS, it may not be able to capture all important KVs, leading to a drop in accuracy. 2. Due to the small BN, the Q may be far from the nearest QKey, making it difficult to obtain the KVs that are important for that Q vector, resulting in excessive error and a decrease in accuracy.

For problem 1, although increasing BS is effective, it would greatly increase memory access overhead. Instead, we use elements from other buckets to perform accuracy compensation. After calculating the cosine values between Q and all the QKeys, the bucket with the highest cosine value retrieves all the cached KVs recorded in the bucket. For the other buckets, the top-N elements based on their cosine values are selected in the subsequent attention calculation, providing accuracy compensation. The N is determined by Eq.1.

$$N = round(max(0, cos\gamma) \cdot BS) \tag{1}$$

For problem 2, we set a threshold δ. When the cosine value between Q and the nearest QKey is less than δ, it is considered an exception. In this case, the bucket table is considered unreliable for that Q. At this time, no selection of KVs is performed. Instead, all cached KVs are directly accessed for attention computation.

4) Overall workflow: Fig.4 illustrates the workflow using associative arrays for selective KV accessing.

Bucket Table Update. ❶ At the beginning, the bucket table is loaded from the DRAM. ❷ Meanwhile, the K vector is multiplied by the QKey weights to obtain the predicted Q-K correlations for each bucket. ❸ Next, the predicted Q-K correlations obtained in step ❷ are used to update the bucket table. During the update, the predicted correlations and token index are inserted into the corresponding position in the table based on the relative magnitude of the correlations, and the element with the smallest correlation in each bucket is removed. There are no insertions if the correlation is smaller than all existing values in the bucket. ❹ After the update is complete, the bucket table is stored back into the DRAM for use in the inference of the next token.

Estimation of significant KVs. ❺ computes the cosine values between Q and all QKeys, which is $\frac{Q \cdot QKey}{||Q||}$, obtaining a list of all cosine values for Q on QKeys. ❻ Next, use this list of cosine values to look up the bucket table obtained in step ❸. If the largest value in the list is greater than δ, all cached KVs referenced in the nearest bucket are used for attention calculation, and the KVs referenced in other buckets are used for accuracy compensation as described above. Otherwise, the previously mentioned exception handling is performed. ❼ After obtaining all important KVs' indexes, these indexes are merged and deduplicated because indexes from different

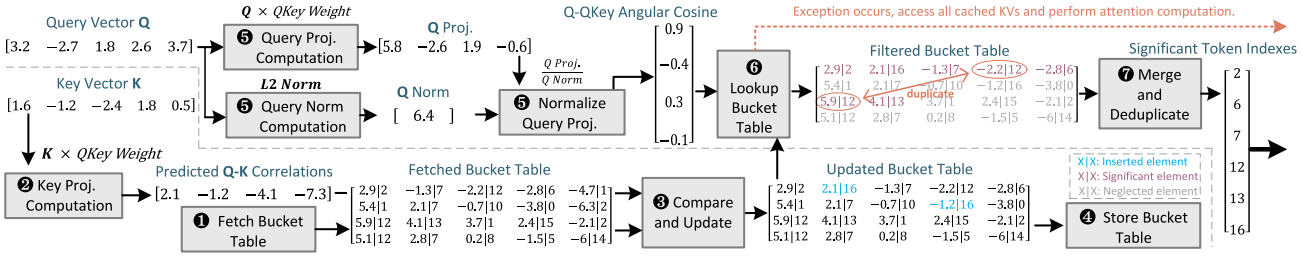


Fig. 4. Workflow using associative arrays for selective KV access. Assume the current token index is 16.

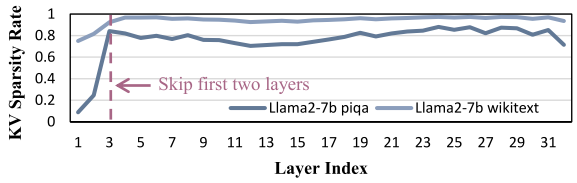


Fig. 5. KV skipping potential of Llama2-7b in Wikitext-2 and PIQA tasks.

buckets may be repeated. Then, a list of KVs' indexes to be accessed is obtained for subsequent attention computation.

5) Further improvement measures: As shown in Fig.5, we analyze the proportion of attention weights below 0.001 to evaluate the KV sparsity potential of each layer for PIQA and Wikitext-2 in Llama2-7b. It was found that the KV sparsity potential is lower in the initial layers. This situation also exists in other LLMs and tasks. If our proposed technique is applied to the initial layers, it may lead to a drop in accuracy, or require a large table to meet accuracy requirements, resulting in negative gains. Therefore, in this paper, we skip the first 2 layers and only apply the proposed technique to the subsequent layers for all LLMs to achieve high overall gains. Another possible approach is to apply bucket tables of different sizes for different layers, which we leave for future work.

IV. Hardware Design

A. Overview

Fig. 6 depicts the overview architecture of our proposed hardware accelerator design. The accelerator consists of two major components: (a) the prediction engine, and (b) the computation engine. The prediction engine implements the proposed algorithm, which updates the bucket table and predicts the KVs required for each attention head. The load/store unit (LSU) then loads these KVs from off-chip memory. Finally, the computation engine performs the attention computation.

B. Prediction Engine

The prediction engine performs projection computation and determines important KVs to be fetched from off-chip memory. It includes the following modules: the processing elements (PEs), a bucket table, and an index merging module. A PE is implemented as a multiply-accumulate (MAC) tree, as shown in Fig. 6(c). The L2-norm required by normalization is calculated by a PE and a square root module. An array of

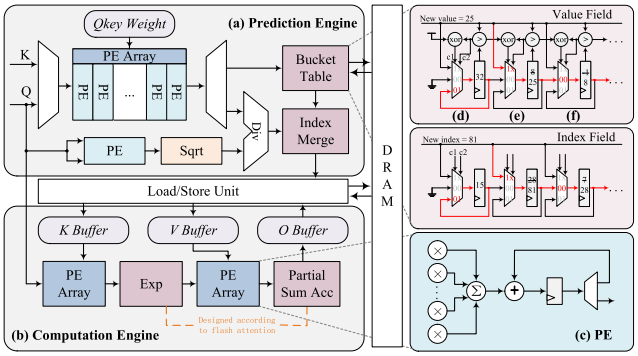


Fig. 6. Overview of Sella accelerator.

stacked PE performs the vector-matrix multiplication operation and is responsible for the computation of predicted Q-K correlation.

Each row of the bucket table has a correlation value field and an index field, and is designed based on shift registers. The value field always maintains a descending order from left to right, this is achieved by a compare-and-insert updating strategy. When a new value-index pair arrives, the new value is first compared with each register, if the new value is smaller, then the register keeps its value (Fig. 6(d)). Otherwise, the registers with smaller values need to be right-shifted (Fig. 6(f)), and the left-most register among these registers (Fig. 6(e)) is updated by the new value. The position of this left-most register is found by the xor gates. Therefore, the value field remains ordered after inserting a new value, and the smallest value will be discarded. The index field records the KVs' indexes for the value field, so the control signals ($c1$ and $c2$) for multiplexers are the same as their corresponding values. An example of this updating process is presented in Fig. 6, where the activated data paths are highlighted in red.

The index merging module selects and deduplicates the top-N indexes required for computation in each bucket, as mentioned in section III-B3. We adopt a bit-map to achieve the deduplication of indexes. The bit-map consists of M 1-bit registers, where M is the maximum index possible. In this paper, we set M to 2048, which is the maximum context length of the tested networks. When a new index i arrives, we first look up the i-th register in the bit-map, as in Eq. 2, the index is $Valid$ when it has not occurred until cycle t. Then, the bit-

map is updated by setting the i-th register to 1, as in Eq. 3, where BM stands for bit-map.

$$Valid_t = !((1 << i) \& BM_t) \quad (2)$$

$$BM_{t+1} = (1 << i) | BM_t \quad (3)$$

The index merging module's output is serial and is pipelined with LSU. Once a new valid token index is generated, the LSU will fetch the corresponding KVs from off-chip memory.

C. Computation Engine

The computation engine performs the attention calculation. It is designed to be fully pipelined to maximize throughput. The computation engine includes two PE arrays, exponent modules, partial sum accumulators, and on-chip buffers. The two PE arrays are homogeneous to the PE array mentioned in the prediction engine, responsible for $Q \times K^T$ and $Attn \times V$ computation, respectively. The exponent module and partial sum accumulator are designed according to the flash attention algorithm [12], in which the *softmax* operation can be completed in a single traverse on the input, avoiding the stalling problem of *softmax* with a long context. The K and V buffers hold only the selected KVs loaded by LSU which are determined by the prediction engine.

V. Evaluation

A. Evaluation Methodology

For algorithm-related evaluations, we use the lm-evaluation-harness library [13] and evaluate Sella on three different LLMs: Llama2-7b [2], Opt-6.7b [3] and Pythia-6.9b [14]. Our evaluation benchmarks include PIQA [15], OpenBookQA [16] (OBQA), COPA [17], Winogrande [18] (WG), and Wikitext-2 [19] (Wiki). Specifically, we run PIQA as a 4-shot task, whereas OpenBookQA, COPA, and Winogrande are 16-shot tasks for longer sequences. We report task-specific metrics, e.g, perplexity for Wikitext-2 and accuracy for the rest tasks. We randomly picked 100 samples from Wikitext-2 for clustering to calibrate the QKey weights.

We implemented the hardware in Chisel [20] and synthesized using Synopsis Design Compiler under 22nm CMOS technology to get the chip area and total power at 500MHz working frequency. Table II provides the design parameters and area/power breakdown of Sella. We use HBM2 as the main memory and set the bandwidth to 256 GB/s. The DRAMsim3 [21] is used to evaluate the HBM's access energy. For performance evaluation, we developed a cycle-accurate simulator based on Sanger's open-source codes. During the performance evaluation, we set the first 20% of the sequence as prompts and the remaining 80% as generated tokens for each dataset.

We compare Sella with a server GPU (NVIDIA TITAN RTX), a server CPU(Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz), and the state-of-the-art attention accelerators SpAtten [6] and Sanger [7]. For SpAtten, we reproduced the results based on its paper and open-source code. We assumed it first performs 70% sparsity on the prompt KVs, then started token generation. For Sanger, we obtained its performance from its open-source simulator. For a fair comparison, we scale the bandwidth of SpAtten and Sanger to 256GB/s.

TABLE II
Design Parameters of Sella.

Module	Parameters	Area(mm^2)	Power(mW)
Prediction Engine	8×8 MACs 8×128 bucket table 8KB QKey weight buffer	0.3	537.7
Computation Engine	Two 32×16 MACs 16 Exp. modules 32 partial sum accumulators 256KB KV buffer 16KB output buffer	6.4	1106.5
Total	-	6.7	1644.2

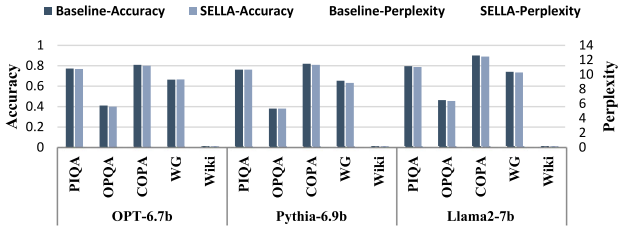


Fig. 7. Accuracy of baseline and Sella.

In the following experiments, unless stated otherwise, for Sella, BN is set to 64, α is set to 0.0625, δ is set to 0.6, and this technique is not employed for the first two layers of all the networks.

B. Evaluation Results

1) Accuracy Results: We compared Sella's inference accuracy with the baseline (obtained with the prediction engine off). The results are shown in Fig. 7. Sella demonstrates excel-

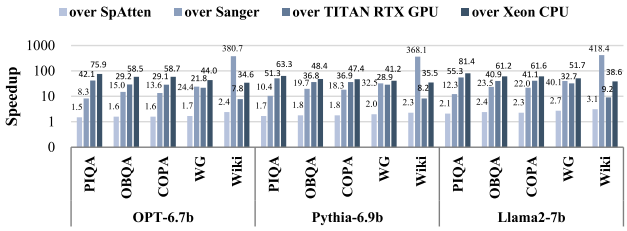


Fig. 8. Speedup over other accelerators.

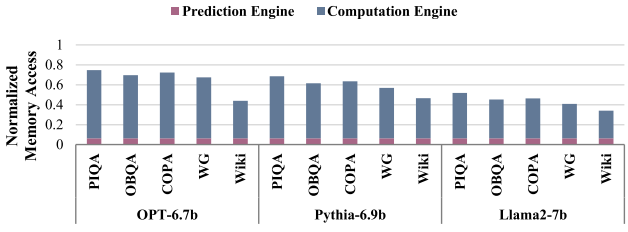


Fig. 9. Normalized memory access breakdown of Sella.

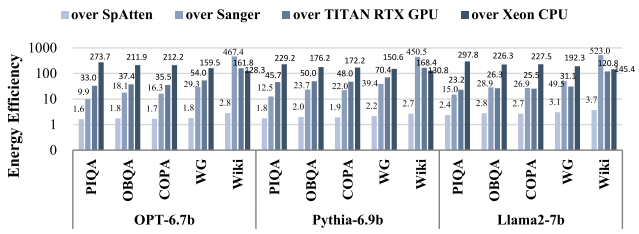


Fig. 10. Energy efficiency over other accelerators.

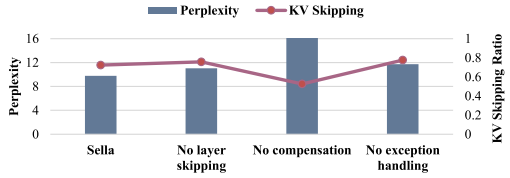


Fig. 11. Comparison of the effects of enabling and disabling layer skipping, accuracy compensation, and exception handling.

lent algorithmic performance. Across four question-answering tasks on three networks, the accuracy drops by 1.4% at most. For the Wikitext-2 dataset, The perplexity increase was less than 7%.

2) Comparison with other accelerators: Fig.8 shows the speedup of Sella compared to SpAtten, Sanger, TITAN RTX GPU, and Xeon CPU. Sella demonstrates superior performance over other accelerators across five datasets on three networks. On average, Sella achieves a speedup of 2.1×, 93.8×, 31.4×, and 53.5× over SpAtten, Sanger, TITAN RTX GPU, and Xeon Gold 5218 CPU. The speedup mainly comes from Sella's ability to skip unnecessary KVs during each token generation at a very low cost, reducing substantial off-chip KV cache access and attention computation. Sanger performs poorly on Wikitext-2 because the Wikitext-2 sequence length is long, and Sanger does not support KV cache, requiring KV recomputation with each token generation.

Fig.9 shows the ratio of Sella's memory access breakdown with prediction engine enabled versus disabled. It can be seen that it reduces a large amount of off-chip memory access, reaching up to 66% (Wikitext-2 in Llama2-7b). The rest of the observations from the figure are as follows: (1) the overhead introduced using associative arrays is relatively small, accounting for at most 6.3% of the memory access before optimization. (2) Gains are greater on Wikitext-2. This is due to the long sequence length of the Wikitext-2 task. The speedup improves with longer sequence lengths, as it skips more KV access in these cases.

Fig.10 shows the energy efficiency results. On average, Sella achieves an energy efficiency of 2.3×, 115.5×, 62.1×, and 195.6× over SpAtten, Sanger, TITAN RTX GPU, and Xeon Gold 5218 CPU. The significant reduction in KVs' off-chip access contributes to the superior energy efficiency.

3) Effectiveness of Layer Skipping, Accuracy Compensation, and Exception Handling: Fig.11 demonstrates the

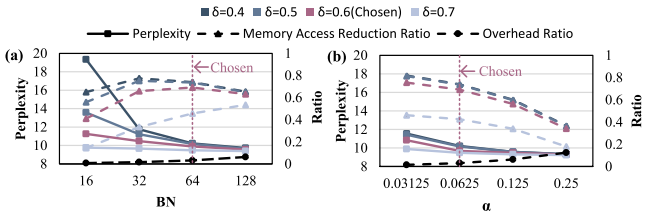


Fig. 12. Design space exploration of BN, α and δ. Overhead means the memory access introduced by the prediction engine.

effectiveness of the several proposed techniques. The KV skipping ratio represents the percentage of KVs skipped by the proposed technique, with higher values indicating better inference performance. Experiments were performed on the Wikitext-2 with the Llama2-7b. The perplexity significantly increased after disabling the layer-skipping and exception-handling techniques. Despite the KV skipping ratio also increasing slightly, the model's accuracy was no longer as expected. When disabling accuracy compensation, there was a sharp increase in perplexity, but the KV skipping ratio decreased. This is because disabling accuracy compensation led to significant inference errors, causing the Q vectors across layers to deviate greatly from cluster centroids, triggering many exception-handling events.

4) Design Space Exploration: In this section, we explore the design space of BN, α, and δ in Sella. Fig.12 shows the results of the design space exploration on the Wikitext-2 using the Llama2-7b model. The figure indicates that a larger δ corresponds to lower perplexity, but also a lower memory access reduction ratio. This is because a higher δ implies stricter exception-handling criteria, leading to more exceptions. As BN increases, perplexity decreases, while the reduction ratio rises at δ values of 0.4, 0.5, and 0.6 before slightly decreasing, and steadily rising at δ = 0.7. This is because, at lower δ values, when BN increases to a certain extent, the memory access from accuracy compensation exceeds the reduction from exception handling, leading to a slight decrease in the reduction ratio; the opposite occurs at higher δ values. For α, increasing its value leads to lower perplexity and reduced reduction ratio. This is because a larger α introduces more important KVs. Furthermore, as BN and α increase, the memory access overhead introduced by the associative arrays also steadily grows. By balancing the trade-offs between accuracy and memory access, we selected BN=64, α=0.0625, and δ=0.6 as the default settings.

VI. Conclusion

This paper introduces Sella, a software-hardware co-design solution to speed up LLM token generation by reducing cached KV access and computations via cluster-based associative array. Experiments show that Sella can deliver significant performance gains and reduce substantial memory access without significant accuracy loss.

REFERENCES

[1] OpenAI, "Gpt-4 technical report," 2024. [Online]. Available: https://arxiv.org/abs/2303.08774

[2] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, and Others, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[3] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022. [Online]. Available: https://arxiv.org/abs/2205.01068

[4] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2402.06196

[5] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh *et al.*, "A3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 328–341.

[6] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021, pp. 97–110.

[7] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," ser. MICRO '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 977–991. [Online]. Available: https://doi.org/10.1145/3466752.3480125

[8] T. J. Ham, Y. Lee, S. H. Seo, S. Kim, H. Choi, S. J. Jung, and J. W. Lee, "Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 692–705.

[9] Z. Qu, L. Liu, F. Tu, Z. Chen, Y. Ding, and Y. Xie, "Dota: detect and omit weak attentions for scalable transformer acceleration," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 14–26. [Online]. Available: https://doi.org/10.1145/3503222.3507738

[10] Y. Zhao, D. Wu, and J. Wang, "Alisa: Accelerating large language model inference via sparsity-aware kv caching," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 1005–1017.

[11] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[12] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[13] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h *et al.*, "A framework for few-shot language model evaluation, 12 2023," *URL https://zenodo.org/records/10256836*, vol. 7.

[14] S. Biderman, H. Schoelkopf, Q. G. Anthony, H. Bradley, K. O'Brien, E. Hallahan, M. A. Khan, S. Purohit, U. S. Prashanth, E. Raff *et al.*, "Pythia: A suite for analyzing large language models across training and scaling," in *International Conference on Machine Learning*. PMLR, 2023, pp. 2397–2430.

[15] Y. Bisk, R. Zellers, J. Gao, Y. Choi *et al.*, "Piqa: Reasoning about physical commonsense in natural language," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 7432–7439.

[16] T. Mihaylov, P. Clark, T. Khot, and A. Sabharwal, "Can a suit of armor conduct electricity? a new dataset for open book question answering," 2018. [Online]. Available: https://arxiv.org/abs/1809.02789

[17] A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "Superglue: A stickier benchmark for general-purpose language understanding systems," *Advances in neural information processing systems*, vol. 32, 2019.

[18] K. Sakaguchi, R. L. Bras, C. Bhagavatula, and Y. Choi, "Winogrande: An adversarial winograd schema challenge at scale," *Communications of the ACM*, vol. 64, no. 9, pp. 99–106, 2021.

[19] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016. [Online]. Available: https://arxiv.org/abs/1609.07843

[20] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[21] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

LLMShare: Optimizing LLM Inference Serving with Hardware Architecture Exploration

Hongduo Liu[1], Chen Bai[2], Peng Xu[1], Lihao Yin[3], Xianzhi Yu[3],
Hui-Ling Zhen[3], Mingxuan Yuan[3], Tsung-Yi Ho[1], Bei Yu[1]
[1]CUHK [2]HKUST [3]Huawei Technologies

***Abstract*—Large Language Models (LLMs) have revolutionized language tasks but pose significant deployment challenges due to their substantial computational demands during inference. The hardware configurations of existing LLM serving systems do not optimize for the different computational and bandwidth needs of the prefill and decoding phases in LLM inference, leading to inefficient resource use and increased costs. In this paper, we systematically investigate promising hardware configurations for LLM inference serving. We develop a simulator that models the performance and cost across different hardware solutions and introduce a customized design space exploration framework to identify optimal setups efficiently. By aligning hardware capabilities with the specific demands of the prefill and decoding phases, we achieve 13% cost savings and over 4× throughput improvements compared to conventional serving system setups.**

I. Introduction

Large Language Models (LLMs), such as GPTs [1], [2] and Llamas [3], [4], have achieved unprecedented performance in language understanding, generation, and complex reasoning tasks. These models have become integral to numerous applications, including chatbots [5], code generation [6], [7], and intelligent assistants [8]. As the scale of these models continues to grow, the scaling law has emerged: larger models can deliver better performance across a broad spectrum of tasks [9], [10].

However, deploying these large-scale models for inference-serving systems in real-world applications poses significant challenges. The computational demands of LLM inference require substantial GPU resources to meet stringent service-level objectives (SLOs) regarding latency and throughput. Efficiently providing LLM inference systems with performance guarantees and reducing deployment costs has become a critical problem for researchers and industry practitioners.

To address these challenges, previous research has explored various optimization strategies for LLM inference serving. Scheduling-level approaches [11]–[13] aim to maximize GPU utilization through efficient scheduling algorithms. For instance, Orca [11] introduces continuous batching at the token level, dynamically integrating new requests into the batch to improve throughput. Memory management techniques [14]–[16] focus on optimizing the use of GPU memory resources. For example, PagedAttention [14] manages the key-value (KV) cache using non-contiguous memory blocks to reduce memory fragmentation and waste.

While these approaches have made significant strides in optimizing LLM inference serving, they largely focus on software-level optimizations but merely consider the hardware configurations underlying the serving system. Recognizing this gap, a recent work Splitwise [17] proposes to split the prefill and decoding phases of LLM inference across machines with different hardware capabilities. They observe that the prefill phase is significantly more compute-intensive than the decoding phase, suggesting that using high-end GPUs like the NVIDIA H100 for both phases can lead to underutilization of hardware resources during decoding. Fig. 1 lists comparison between NVIDIA A100 cluster and H100 cluster with 8 GPUs on Llama-70B [3] without batching, where the statistics is from [17]. As we can see, although the H100 offers substantial throughput improvements (1.85×) during the prefill phase due to its increased computational power, the performance gains during the decoding phase are relatively marginal (1.43×). The discrepancy arises because the H100 exhibits limited improvements in memory size and bandwidth over the A100. This observation prompts a critical question: If we can adjust the computation capacity or memory throughput of the H100, can we achieve a better performance-cost tradeoff?

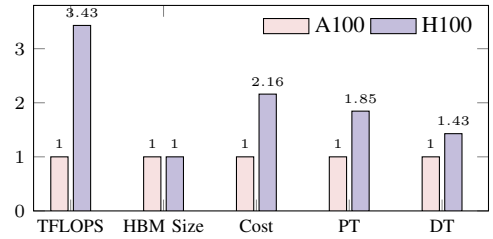


Fig. 1 Comparison of NVIDIA A100 and H100 cluster with 8 GPUs on Llama-70b without batching. 'PT' denotes prefill phase throughput, and 'DT' denotes decoding phase throughput.

Existing hardware solutions with fixed configurations, such as A100 and H100 clusters, may not perfectly match the specific computational and memory requirements of the different phases of LLM inference. This mismatch can lead to suboptimal hardware utilization and increased costs. Therefore, we are motivated to explore whether customizing hardware configurations to better align with the distinct computation and memory demands of the prefill and decoding phases can optimize both performance and cost.

In this work, we introduce **LLMShare**, a framework that optimizes large language model (LLM) inference serving through hardware architecture exploration. Our approach involves developing a simulator that models the performance and cost of different hardware configurations within an LLM serving system. Given the vast design space of LLM serving systems and the time-consuming nature of performance evaluations, we also introduce a customized design space exploration algorithm to efficiently identify optimal hardware configurations. By aligning hardware capabilities with the specific requirements of each inference phase, we achieve a 13% cost reduction and a 4× throughput improvement compared to traditional GPU setups.

The main contributions of this paper are as follows:

- We develop a simulator that models the behavior of an LLM serving system, enabling the assessment of performance and cost across various hardware configurations.
- We propose an effective design space exploration framework to identify Pareto-optimal hardware configurations within a large design space.
- We demonstrate that customized hardware configurations can achieve significant cost savings and throughput improvements over conventional GPU configurations in LLM serving.

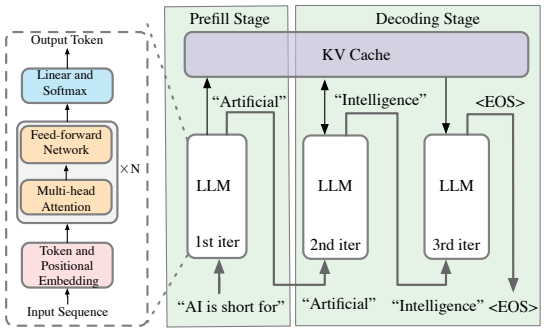


Fig. 2 The inference process of LLMs.

II. Preliminaries

A. Overview of LLM inference

Fig. 2 illustrates the inference process of decoder-only LLMs. These mainstream LLMs are predominantly constructed from multiple Transformer blocks [18], each comprising a self-attention-based multi-head attention (MHA) mechanism followed by a feed-forward network (FFN).

In the Transformer architecture, the initial step involves applying three different weight matrices to encoded embeddings of the input text sequence, which computes the queries $\boldsymbol{Q}$, keys $\boldsymbol{K}$, and values $\boldsymbol{V}$ matrices for each token. The self-attention mechanism then utilizes these matrices to calculate attention scores, allowing each token to focus on different parts of the sequence, which is given by

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}, \tag{1}$$

where d is the dimension of the token embeddings. By employing multiple attention heads, the model can capture different types of relationships in parallel. The outputs from these multiple attention heads are concatenated and passed through a feed-forward network, which refines the information and enables the model to learn complex patterns and dependencies.

The inference process of LLMs can be systematically divided into two primary stages: the prefill phase and the decoding phase. During the prefill phase, the model processes the entire input prompt, which establishes the initial context for the LLM and generates the first token. As depicted in Fig. 2, when the LLM processes the input sequence "AI is short for", it computes the internal representations necessary to generate the next token, "Artificial". A crucial optimization in this process is the caching of the key-value (KV) pairs generated by each transformer block for each token. By storing the KV cache in memory, the model enhances computational efficiency by avoiding recomputation of these representations during subsequent decoding steps.

Following the prefill phase, the decoding phase begins, during which the model generates output tokens one at a time in an autoregressive manner. In each decoding step, the LLM predicts the next token based on both the original prompt and the sequence of tokens generated thus far. Simultaneously, it generates new KV pairs for use in subsequent iterations. For example, at the beginning of the decoding phase, the model uses the cached KV pairs to predict the next token "Intelligence". This autoregressive process ends until the LLM generates an end-of-sequence token <EOS>, signaling the completion of the generated output. By utilizing the cached KV pairs and processing tokens autoregressively, the model efficiently generates sequences without redundant computations, which is essential for handling long prompts and making real-time interactions feasible.

Parameter	Notation	Value Range	#
Server Count	sc	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	10
Device Count	dc	4, 8, 12, 16	4
Link Count Per Device	lc	6, 12, 18, 24	4
Main Memory (GB)	mm	40, 64, 80, 96, 112, 128	6
Global Buffer (MB)	gb	20, 30, 40, 50, 60, 70, 80, 90, 100, 110	10
Core Count	cc	72, 96, 108, 132, 156, 180	6
Local Buffer (KB)	lb	64, 128, 192, 256, 320, 384, 448, 512	8
Lane Count	lc	1, 2, 4, 8	4
Array Height	ah	16, 32, 64, 128	4
Vector Width	vw	16, 32, 64, 128	4

TABLE I Design space of the prefill and decoding pools. Each pool has a design space size of approximately 3×10^7. Consequently, the entire design space of an LLM serving system is nearly 9×10^{14}.

B. LLM Serving Framework

Fig. 3 illustrates the framework of a general LLM serving system. The system includes a prefill pool and a decoding pool, each consisting of multiple servers for parallel processing. The prefill pool manages the prefill phase of a request, while the decoding pool handles the completion of the decoding phase, with KV cache transfer bridging the two stages. Separating prefill and decoding servers can optimize performance [17], [19], [20], as these stages have different computational and memory communication characteristics. For instance, the prefill stage is computation-intensive and requires limited batch sizes to maintain performance, whereas the decoding phase can handle larger batch sizes, enabling higher throughput [17].

The serving system also includes a request scheduler that assigns tasks to the prefill and decoding pools. Additionally, the scheduler controls a router that dynamically reallocates server instances. If there are pending tasks in the prefill pool and no tasks in the decoding pool, instances can be moved from the decoding pool to the prefill pool. Conversely, if the prefill pool has available servers and there are blocked tasks in the decoding pool, instances can be transferred from the prefill pool to the decoding pool. Within each pool, additional scheduling handles the batching and execution of prefill or decoding tasks.

Each server (e.g., an NVIDIA DGX node) in the prefill and decoding pool comprises multiple devices connected via an inter-device interconnect (e.g., NVLink). Each device (e.g., a GPU) contains multiple cores, a shared global buffer, and off-chip main memory. The global buffer (e.g., L2 cache in NVIDIA GPUs) connects to both the main memory and the cores. Each core (e.g., a Streaming Multiprocessor in NVIDIA GPUs) includes multiple lanes that share a local buffer (e.g., L1 cache in NVIDIA GPUs). The local buffer connects to the global buffer through an on-chip interconnect. Each lane includes its vector unit, systolic or multiply-accumulation array, registers, and control logic. Here we follow the idea of LLMCompass [21] to use aforementioned general device template to describe mainstream accelerators used for LLM inference, such as NVIDIA H100 [22], AMD MI210 [23], and Google TPUv3 [24]. This description is universal because these accelerators share a similar architectural hierarchy.

This hardware template enables various LLM serving system configurations. As shown in TABLE I, we have a total design space of nearly 9×10^{14} configurations for the complete serving system. Here we only consider the server count range from 1 to 10, it can be scaled proportionally for large serving systems.

C. Problem Formulation

Definition 1 (LLM Serving System Design Space)**.** *An LLM serving system design combines a prefill pool and a decoding pool, each with design spaces detailed in TABLE I. We denote a system configuration as $\boldsymbol{x} \in \mathcal{X}$, where $\mathcal{X}$ represents the complete design space.*

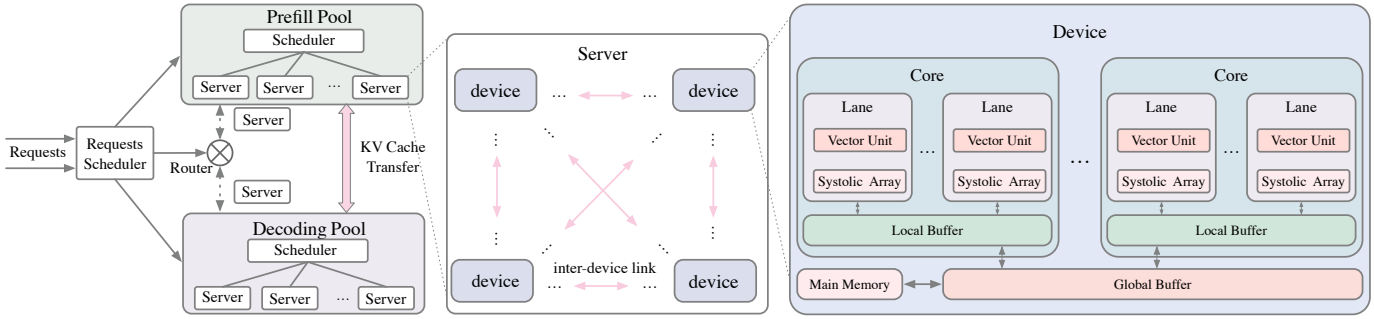


Fig. 3 Overview of a LLM serving system.

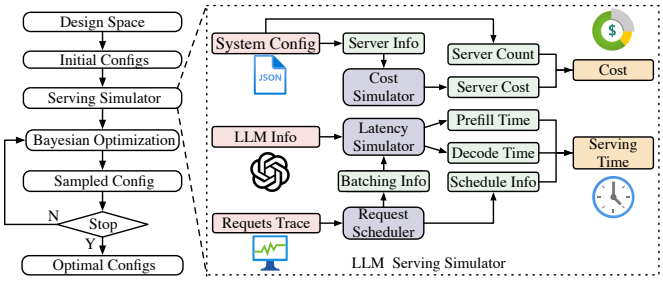


Fig. 4 The overview of LLMShare.

Definition 2 (Pareto Optimality)**.** *Given a stream of requests and an LLM, each configuration $\boldsymbol{x}$ has associated objectives: serving time $f_t(\boldsymbol{x})$ and cost $f_c(\boldsymbol{x})$. A design $\boldsymbol{x}^* \in \mathcal{X}$ is Pareto optimal if there is no other design $\boldsymbol{x} \in \mathcal{X}$ such that:*

$$f_t(\boldsymbol{x}) \leq f_t(\boldsymbol{x}^*) \quad \textit{and} \quad f_c(\boldsymbol{x}) \leq f_c(\boldsymbol{x}^*),$$

and at least one of the inequalities is strict: $f_t(\boldsymbol{x}) < f_t(\boldsymbol{x}^*)$ or $f_c(\boldsymbol{x}) < f_c(\boldsymbol{x}^*)$.

The design of LLM serving systems involves optimizing both serving time and cost. These objectives are inherently conflicting as reducing serving time typically requires additional servers, thereby increasing costs. Consequently, we focus on identifying Pareto-optimal designs.

Problem 1 (LLM Serving System Design Space Exploration)**.** *Given a design space $\mathcal{X}$, the goal of design space exploration is to identify a subset $\mathcal{X}^* \subset \mathcal{X}$ containing all Pareto-optimal configurations.*

III. LLMShare

A. Overview of LLMShare

Fig. 4 illustrates the overall workflow of LLMShare, which comprises an LLM serving simulator and a design space exploration framework. The simulator outputs the cost and serving time for a given LLM serving system configuration, providing essential feedback to the design space exploration process.

The simulator takes three main inputs: the serving system configuration, LLM information, and the request trace. The system configuration includes data on the number of servers in the prefill and decoding pools, as well as their specific configurations, as detailed in TABLE I. The LLM information covers attributes like the number of layers, the attention head count of multi-head self-attention, and hidden dimension size. The request trace details the arrival time, input token size, and output token size for each request. The cost simulator calculates the cost of a server based on its configuration, including the die costs of chips and main memory costs. By aggregating the costs of the servers in the prefill and decoding pools, the cost simulator determines the total cost of the serving system. Using the LLM information and request trace, the request scheduler decides on request batching and assigns tasks to the prefill and decoding pools. The latency simulator then calculates the prefill and decode times for each request. By considering additional scheduler details like KV cache transfer time and request pending time in the batching info reported by the request scheduler, the simulator ultimately determines the system's overall serving time. We use LLMCompass [21] for cost and latency simulation, which achieves less than a 5% error rate compared to real-world hardware. For request scheduling, we use the simulator proposed in Splitwise [17].

We adopt Bayesian optimization for design space exploration. It begins with a predefined design space, where Memory-Centric Initialization (Section III-B) is employed to generate initial configurations. Specifically, the memory size of the serving system guides the sampling of these initial designs. These configurations are then evaluated using our LLM serving simulator to obtain serving time and cost metrics based on various system configurations. To identify the optimal configurations, a deep tree kernel-based (Section III-C) Bayesian optimization algorithm is utilized to find configurations that optimize a specified acquisition function. Subsequently, the selected designs are re-evaluated through the simulator to obtain updated serving time and cost data, which are used to refine the surrogate model. Ultimately, LLMshare outputs a set of Pareto optimal designs based on the explored configurations, including initial configurations and sampled configurations during Bayesian optimization.

B. Memory-Centric Initialization (MCI)

Initial sampling involves selecting a small subset of configurations, denoted as $\mathcal{D}_x \subset \mathcal{X}$, to evaluate before commencing the iterative Bayesian optimization process. These initial points lay the groundwork for constructing the surrogate model. Evaluating each LLM serving system configuration requires processing a request trace comprising thousands of requests. Even with request batching, the simulation time remains substantial due to the frequent invocation of a computationally intensive latency simulator. Consequently, only a limited number of points are feasible for initialization.

For effective optimization, it is imperative that the initial designs are diverse and representative to ensure comprehensive coverage of the entire design space. Traditional approaches have employed random sampling [25] or orthogonal designs [26] due to their simplicity. More recently, transductive experimental design (TED) [27] has gained popularity for selecting the most representative and challenging-to-predict designs as initial points in high-level synthesis [28] and CPU microarchitecture design space exploration [29].

However, these methods do not account for the unique characteristics of LLM serving systems, leading to suboptimal initial sets.

To overcome the limitations of existing initialization methods in the context of LLM serving systems, we propose a memory-centric initialization approach. This approach is motivated by the critical role that memory configuration plays in both serving time and cost. Firstly, modern accelerators designed for LLM inference often employ high-bandwidth memory (HBM) to facilitate rapid data communication. While HBM offers substantial performance benefits, it also significantly increases the total cost of servers due to its high expense. Secondly, the memory capacity directly impacts the number of requests that can be batched and processed simultaneously. Larger memory allows for higher batching capability, which can dramatically increase throughput. Therefore, our memory-centric initialization method selects initial configurations that are more representative of the performance and cost trade-offs inherent to the LLM serving system.

Algorithm 1 Memory-Centric Initialization

Input: • $\mathcal{U}$: unsampled design space with n configurations;
• t: total number of initial configurations to select;
• u: number of groups used during sampling.
Output: $\mathcal{D}_x$ with $|\mathcal{D}_x| = t$ ▷ Selected initial designs

Compute the total main memory size for each design:
for $i = 1$ to n **do**

$$c_i = \boldsymbol{x}^p_{i,\text{sc}} \cdot \boldsymbol{x}^p_{i,\text{dc}} \cdot \boldsymbol{x}^p_{i,\text{mm}} + \boldsymbol{x}^d_{i,\text{sc}} \cdot \boldsymbol{x}^d_{i,\text{dc}} \cdot \boldsymbol{x}^d_{i,\text{mm}};$$

Determine percentiles: $P = \left\{ \frac{100 \times j}{u} \,\middle|\, j = 0, 1, \ldots, u \right\}$;
Compute bin edges for the percentiles of $\{c_i\}_{i=1}^n$:

$$B = \left\{ b_j = \text{Percentile}(\{c_i\}_{i=1}^n, p_j) \,\middle|\, j = 0, 1, \ldots, u \right\};$$

Compute base sample count per group: $q \leftarrow \left\lfloor \frac{t}{u} \right\rfloor$;
Compute remainder: $r \leftarrow t \bmod u$;
Initialize $\mathcal{D}_x \leftarrow \emptyset$;
for $j = 1$ to u **do**
  $\mathcal{G}_j = \left\{ i \,\middle|\, b_{j-1} \le m_i < b_j \right\}$;
  if $j \le r$ **then**
    $s_j \leftarrow q + 1$;
  else
    $s_j \leftarrow q$;
  Select s_j samples from $\mathcal{G}_j$: $\mathcal{S}_j = \textbf{TED}(\mathcal{G}_j, s_j)$;
  $\mathcal{D}_x \leftarrow \mathcal{D}_x \cup \mathcal{S}_j$;
return $\mathcal{D}_x$

Algorithm 1 presents our memory-centric initialization method. For each configuration, $\boldsymbol{x}_i$, a composite metric c_i is computed using main memory-related features. Specifically, c_i represents the total main memory size of configuration i,

$$c_i = \boldsymbol{x}^p_{i,sc} \cdot \boldsymbol{x}^p_{i,dc} \cdot \boldsymbol{x}^p_{i,mm} + \boldsymbol{x}^d_{i,sc} \cdot \boldsymbol{x}^d_{i,dc} \cdot \boldsymbol{x}^d_{i,mm}, \tag{2}$$

where $\boldsymbol{x}^p_{i,sc}$ denotes the server count of the prefill pool of i-th configuration while $\boldsymbol{x}^d_{i,sc}$ indicates the server count of the decoding pool of i-th configuration. The abbreviation of other features can be referred to TABLE I. Once these total memory sizes are computed, the algorithm determines percentiles to divide the range of memory sizes into equal intervals. For each interval defined by these percentiles, the algorithm identifies all designs whose total memory sizes fall within that interval. It then samples configurations using a basic sampling method like TED [27] from each interval. Finally, t samples with representative memory sizes can be obtained.

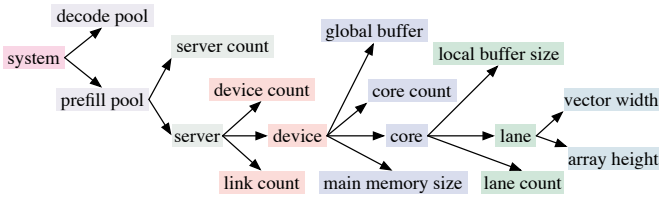


Fig. 5 Tree structure of the design space, where each leaf node represents a design parameter. Design parameters at the same hierarchical level are marked with the same color. The subtree for the decode pool is omitted as it mirrors the prefill pool's subtree.

C. Deep Tree Kernel (DKL)

After sampling the initial serving system configurations $\mathcal{D}_x$, we can utilize the simulator as shown in Fig. 4 to get serving time and cost, denoted as $\mathcal{D}_y$. Due to the time-consuming nature of metric evaluation, we aim to construct a surrogate model $\mathcal{M}$ that describes the relationship between the design space $\mathcal{X}$ and the target space $\mathcal{Y}$ based on the currently selected samples $(\mathcal{D}_x, \mathcal{D}_y)$. This surrogate model guides the selection of the next design by predicting function values at unobserved points and estimating the associated uncertainties. It is updated as the labels of newly selected samples are obtained. In this paper, we use a Gaussian process (GP) [30] as our surrogate model due to its flexibility and modeling for both predictions and uncertainty estimates, which has wide application in various design space exploration tasks [31]–[33]. A Gaussian process defines a prior over value function $f(\boldsymbol{x})$ such that any finite set of function values follows a multivariate Gaussian distribution. Formally, a GP is specified by its mean function $m(\boldsymbol{x})$ and kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$:

$$f(\boldsymbol{x}) \sim \mathcal{GP}\left(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')\right). \tag{3}$$

Given observed data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $y_i = f(\boldsymbol{x}_i) + \epsilon_i$ and ϵ_i is Gaussian noise with variance σ_e^2, we consider the joint distribution of the function values at the observed points $\boldsymbol{f} = [f(\boldsymbol{x}_1), \ldots, f(\boldsymbol{x}_n)]^\top$ and the function value at a new point $\boldsymbol{x}_*$, denoted f_*. This joint distribution is given by:

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mathbf{m} \\ m_* \end{bmatrix}, \begin{bmatrix} \mathbf{K_{XX}} + \sigma_e^2 \mathbf{I} & \mathbf{k_{Xx_*}} \\ \mathbf{k_{x_*X}} & k_{\mathbf{x_*x_*}} \end{bmatrix} \right), \tag{4}$$

where $\boldsymbol{m} = [m(\boldsymbol{x}_1), \ldots, m(\boldsymbol{x}_n)]^\top$, $m_* = m(\boldsymbol{x}_*)$, $\mathbf{K_{XX}}$ is the covariance matrix between the observed inputs with entries $(\mathbf{K_{XX}})_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $\boldsymbol{k}_{\mathbf{Xx}_*} = [k(\boldsymbol{x}_1, \boldsymbol{x}_*), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}_*)]^\top$, $k_{\mathbf{x_*x_*}} = k(\boldsymbol{x}_*, \boldsymbol{x}_*)$, $\mathbf{I}$ is the identity matrix, and σ_e^2 represents the variance of the observation noise.

Choosing an appropriate kernel is crucial for the accuracy of the Gaussian process. As our design space exhibits a hierarchical tree structure as illustrated in Fig. 5, we propose a deep tree kernel tailored to capture the hierarchical structure of the hardware configuration design space for LLM serving systems. Let $T = (\mathcal{V}, \mathcal{E})$ denote a tree-structured configuration, where $\mathcal{V}$ is the set of nodes and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is a set of directed edges that establish a parent-child relationship. Each node $v \in \mathcal{V}$ can be either a leaf node, having no children, or an internal node, with one or more children. The tree also includes a designated root node $v_r \in \mathcal{V}$, which has no parent. The root node of the tree represents the whole serving system, while each leaf node indicates a design parameter of the system. Our objective is to compute an embedding $\boldsymbol{h}_T$ that leverages the features of the

leaf nodes $\boldsymbol{x}$ and the hierarchical structure of the tree. Let $\mathcal{L} \subseteq \mathcal{V}$ denote the set of leaf nodes of T. For an internal node $v \in \mathcal{V} \setminus \mathcal{L}$, the embedding $\boldsymbol{h}_v$ is computed based on the embeddings of its child nodes. Let $\mathcal{C}(v) = \{u_1, u_2, \ldots, u_{k_v}\}$ denote the set of child nodes of node v, where k_v is the number of children of v. The embedding $\boldsymbol{h}_v$ is computed as

$$\boldsymbol{h}_v = \phi_v \left(\text{concat} \left(\boldsymbol{h}_{u_1}, \boldsymbol{h}_{u_2}, \ldots, \boldsymbol{h}_{u_{k_v}} \right); \theta_v \right), \tag{5}$$

where ϕ_v is an embedding function (e.g., an MLP) for internal node v, θ_v denotes its parameters, and concat$(\cdot)$ denotes the concatenation operation. The recursive computation proceeds in a bottom-up manner, starting from the lowest level and moving upwards through the root. At each step, we compute the embeddings of internal nodes only after computing the embeddings of all their child nodes. Finally, the embedding of the tree is given by the embedding of the root node:

$$\boldsymbol{h}_T = \boldsymbol{h}_{v_r}. \tag{6}$$

For example, the lowest-level features such as array height $\boldsymbol{x}_{\text{ah}}$ and vector width $\boldsymbol{x}_{\text{vw}}$, associated with a lane, are first transformed into lane embeddings $\boldsymbol{h}_{\text{lane}}$ using an embedding function ϕ_{lane} parameterized by θ_{lane}:

$$\boldsymbol{h}_{\text{lane}} = \phi_{\text{lane}} \left(\text{concat} \left(\boldsymbol{x}_{\text{ah}}, \boldsymbol{x}_{\text{vw}} \right); \theta_{\text{lane}} \right). \tag{7}$$

This recursive embedding culminates in the system-level embedding $\boldsymbol{h}_{\text{system}}$, which is obtained by combining the embeddings of the prefill pool $\boldsymbol{h}_{\text{prefill}}$ and decoding pool $\boldsymbol{h}_{\text{decode}}$ using the embedding function ϕ_{system} parameterized by θ_{system}:

$$\boldsymbol{h}_{\text{system}} = \phi_{\text{system}}(\boldsymbol{h}_{\text{prefill}}, \boldsymbol{h}_{\text{decode}}; \theta_{\text{system}}). \tag{8}$$

The deep tree kernel k_t between two system configurations $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is then defined as the kernel function applied to their respective system-level embeddings:

$$k_t(\boldsymbol{x}_i, \boldsymbol{x}_j) = k \left(\boldsymbol{h}_{\text{system}}^{(i)}, \boldsymbol{h}_{\text{system}}^{(j)} \right), \tag{9}$$

where k is a traditional kernel function, such as the radial basis function. $\boldsymbol{h}_{\text{system}}^{(i)}$ and $\boldsymbol{h}_{\text{system}}^{(j)}$ denotes the system-level embedding of two configuration $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ respectively. This customized deep kernel effectively measures the similarity between different hardware configurations by leveraging the hierarchical relationships and dependencies inherent in the tree-structured design space.

D. Multi-objective Bayesian Optimization

For the design space exploration of the LLM serving system, we aim to find solutions that balance trade-offs between serving time and cost, which are two conflicting objectives. The set of these optimal trade-off solutions is known as the Pareto front. A solution is considered Pareto-optimal if improving any one objective would lead to the degradation of at least one other objective. The Pareto Hypervolume is a quantitative metric that measures the volume (or area in two dimensions) of the objective space dominated by the Pareto front, bounded by a predefined reference point. Mathematically, consider a set of n objective vectors $\mathcal{P} = \{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(n)}\}$ in an d-dimensional objective space, where each $\mathbf{y}^{(i)} = [y_1^{(i)}, y_2^{(i)}, \ldots, y_d^{(i)}]$ represents the objective values of configuration i. Let $\mathbf{r} = [r_1, r_2, \ldots, r_d]$ be the reference point, which is chosen to be worse than any objective values in $\mathcal{P}$ (for minimization problems). The Pareto hypervolume

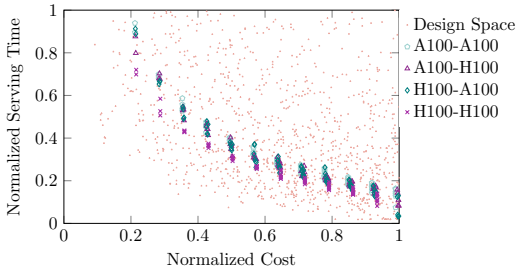


Fig. 6 The effectiveness of LLM serving system design space exploration

HV$(\mathcal{P})$ is defined as:

$$\text{HV}(\mathcal{P}) = \lambda \left(\bigcup_{i=1}^{n} [\mathbf{y}^{(i)}, \mathbf{r}] \right), \tag{10}$$

where $\lambda(\cdot)$ denotes the Lebesgue measure (length, area, volume, etc., depending on d) and $[\mathbf{y}^{(i)}, \mathbf{r}]$ represents the hyper-rectangle (in d-dimensional space) bounded by $\mathbf{y}^{(i)}$ and $\mathbf{r}$.

With the Pareto hypervolume defined, we can now discuss how it guides the selection of new design points in our multi-objective Bayesian optimization framework. We employ the Expected Hypervolume Improvement (EHVI) [34], [35] as our acquisition function to strategically explore the design space. The EIPV quantifies the expected increase in the hypervolume that would result from sampling a new design point, integrating over the uncertainty in the predictions provided by the Gaussian process (GP) surrogate model. Mathematically, for a candidate design $\mathbf{x}_*$, the expected hypervolume improvement EHV$(\mathbf{x}_*)$ is computed as:

$$\text{EHVI}(\mathbf{x}_*) = \int_{\mathbf{y}} \max\left(\text{HV}(\mathcal{P} \cup \{\mathbf{y}\}) - \text{HV}(\mathcal{P}),\, 0\right) p(\mathbf{y} \mid \mathbf{x}_*, \mathcal{D})\, d\mathbf{y}, \tag{11}$$

where $\mathcal{P}$ is the current set of Pareto-optimal objective vectors observed so far, HV$(\mathcal{P})$ is the current Pareto Hypervolume, and $p(\mathbf{y} \mid \mathbf{x}_*, \mathcal{D})$ is the predictive posterior distribution of the objective vector $\mathbf{y}$ at $\mathbf{x}_*$ given the surrogate model and observed data $\mathcal{D}$.

During the search process of Bayesian optimization, we identify the candidate design $\mathbf{x}_*$ that maximizes the EHVI:

$$\boldsymbol{x}_* = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \text{EHVI}(\boldsymbol{x}). \tag{12}$$

IV. Experiments

A. Experiemtal Settings

We validate our methods using a 2-minute serving trace containing 2454 requests, with GPT3-175B [1] as the underlying LLM of the serving system. The distribution of input and output token sizes is derived from a Microsoft Azure production trace [36], reflecting real-world LLM serving demands.

As shown in TABLE I, the complete design space of an LLM serving system approaches 9×10^{14} configurations. Given the time-consuming nature of the simulation pipeline, exhaustively evaluating the entire design space is impractical. Therefore, we construct an offline dataset comprising 1,055 designs through random sampling from the complete design space. For design space exploration (DSE), we initialize with 10 sampled designs and perform 20 exploration steps. u is set to 5 for Algorithm 1.

We compare LLMShare to state-of-the-art DSE methods. DAC'16 [26] uses an AdaBoost-based strategy to selectively simulate informative designs, reducing simulation costs. ASPDAC'20 [37] employs an XGBoost model for flow parameter tuning. ICCAD'21 [29]

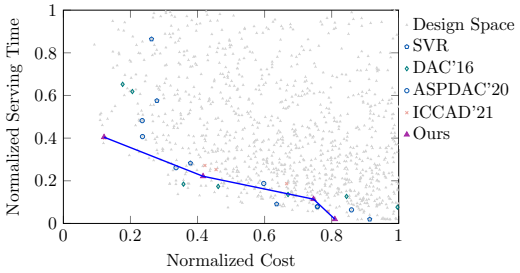


Fig. 7 Learned Pareto optimal set of LLMShare and other baseline methods.

TABLE II Comparison of different algorithms.

Algorithms	Normalized ADRS	Hypervolume (10^8)
SVR [38]	0.1811	4.9593
DAC'16 [26]	0.1718	4.9714
ASPDAC'20 [37]	0.1805	4.9513
ICCAD'21 [29]	0.2059	4.8134
LLMShare	**0.1589**	**5.0552**

integrates active learning for initial sample generation with deep-kernel learning in a Gaussian process to explore optimal RISC-V CPU designs. Additionally, a Support Vector Regression (SVR) [38] based greedy search is used as another baseline. For a fair comparison, we conduct experiments on each baseline algorithm 10 times and report the averages.

We evaluate our design space exploration method using two metrics: hypervolume, as defined in Equation (10), and Average Distance to Reference Set (ADRS). The hypervolume metric measures the size of the objective space dominated by the obtained Pareto front and bounded by a reference point. ADRS quantifies the proximity between a learned Pareto optimal set and the ground truth Pareto optimal set. The ADRS is formally defined as:

$$\text{ADRS}(\mathcal{S}, \mathcal{R}) = \frac{1}{|\mathcal{S}|} \sum_{i=1}^{|\mathcal{S}|} \min_{\boldsymbol{r} \in \mathcal{R}} d(\boldsymbol{s}_i, \boldsymbol{r}), \tag{13}$$

where $\mathcal{S}$ denotes the learned Pareto optimal set, $\mathcal{R}$ represents the groundtruth Pareto optimal set, $\boldsymbol{s}_i$ refers to each solution in $\mathcal{S}$, and $d(\boldsymbol{s}_i, \boldsymbol{r})$ measures the distance between solution $\boldsymbol{s}_i$ and true pareto optimal solution $\boldsymbol{r}$. Higher hypervolume and lower ADRS indicate better DSE algorithm performance.

B. Experiemental Results

1) Necessity of Design Space Exploration: Fig. 6 illustrates the performance comparison between randomly sampled designs from the complete design space and solutions using existing widely adopted commercial GPUs. In our notation, `A100-A100` represents configurations using NVIDIA A100 GPUs for both prefill and decoding phases, while `H100-A100` denotes configurations using NVIDIA H100 GPUs for prefill and A100 GPUs for decoding. The number of prefill and decoding servers varies from 1 to 10. Therefore, 100 total possible configurations of GPU clusters are investigated. Our analysis reveals that certain design configurations consistently outperform commercial solutions, achieving lower costs while reducing serving time. This observation highlights the benefits of exploring diverse hardware configurations to better accommodate the distinct computational and memory patterns of LLM prefill and decoding phases.

2) Performance of Proposed DSE Algorithm: Fig. 7 plots the learned Pareto optimal sets of LLMShare and other baseline algorithms. We observe that LLMShare obtains a Pareto optimal set closer to the real Pareto frontier. We also provide a quantitative comparison, as shown in TABLE II. LLMShare consistently outperforms the baseline algorithms across both ADRS and hypervolume metrics. Specifically, LLMShare achieves 12%, 7%, 11%, and 23% ADRS reduction compared with SVR [38], DAC'16 [26], ASPDAC'20 [37], and ICCAD'21 [29], respectively. The superior ADRS achieved by LLMShare demonstrates its effectiveness in providing a set of solutions that better approximate the true Pareto frontier. Additionally, LLMShare achieves the highest hypervolume, with a 5% improvement compared to ICCAD'21 [29] and a 2% improvement compared to other baselines.

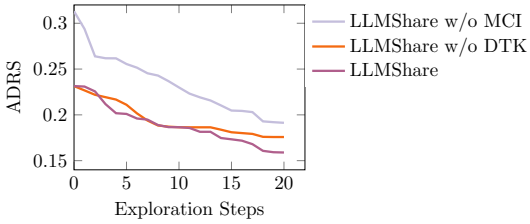


Fig. 8 Ablation study on the effectiveness of DTK and MCI.

TABLE III Normalized cost and request per second (RPS) of a Pareto optimal H100 cluster and a Pareto optimal configuration found by LLMShare. The design parameters are in the same order as TABLE I.

Hardware Config			Cost	RPS
H100-cluster	Prefill Decode	7,8,18,80,50,132,256,4,16,32 6,8,18,80,50,132,256,4,16,32	1.00	1.00
LLMShare	Prefill Decode	4,4,24,112,100,156,512,1,128,32 6,12,18,80,70,72,448,2,32,16	0.87	4.11

Fig. 8 presents an ablation study comparing different configurations of the LLMShare model over exploration steps, with ADRS on the y-axis. For `LLMShare w/o MCI`, we use the TED method implemented in ICCAD'21 [29] to obtain the initial designs instead of our Memory-Centric Initialization (MCI) algorithm. Fig. 8 reveals that the inclusion of MCI allows LLMShare to achieve a set of representative initial designs, resulting in a notably lower ADRS as exploration progresses. We also substitute the Deep Tree Kernel (DTK) with a naive deep kernel that does not consider the hierarchical design space of the LLM serving system, denoted as `LLMShare w/o DTK`. As we can see, DTK enhances the modeling of the design space, enabling more effective exploration compared to the naive deep kernel.

TABLE III lists the cost and throughput comparison between a Pareto optimal H100 cluster among the 100 possible server count combinations and a Pareto optimal configuration found by LLMShare. We can see that the learned configuration can achieve $4\times$ throughput improvement with a 13% reduction in cost.

V. Conclusion

In conclusion, this paper explores optimizing LLM inference serving through hardware design exploration. Our evaluation framework assesses the performance and cost of various configurations, showing that customized solutions tailored for prefill and decoding phases significantly enhance throughput and reduce costs compared to traditional GPUs.

Acknowledgements

The project is supported in part by Research Grants Council of Hong Kong SAR (No. RFS2425-4S02 and No. CUHK14211824), and the MIND project (MINDXZ202404).

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901.

[2] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[4] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[5] Chatgpt overview. [Online; accessed 30-October-2024]. [Online]. Available: https://openai.com/chatgpt/overview/

[6] Github copilot. [Online; accessed 30-October-2024]. [Online]. Available: https://github.com/features/copilot

[7] P. Zehua, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," in *Forty-first International Conference on Machine Learning*, 2024.

[8] Google assistant with bard. [Online]. Available: https://blog.google/products/assistant/google-assistant-bard-generative-ai/

[9] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[10] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling Laws for Neural Language Models," *arXiv preprint arXiv:2001.08361*, 2020.

[11] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[12] C. Holmes, M. Tanaka, M. Wyatt, A. A. Awan, J. Rasley, S. Rajbhandari, R. Y. Aminabadi, H. Qin, A. Bakhtiari, L. Kurilenko *et al.*, "Deepspeed-fastgen: High-throughput text generation for llms via mii and deepspeed-inference," *arXiv preprint arXiv:2401.08671*, 2024.

[13] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee, "Taming throughput-latency tradeoff in llm inference with sarathi-serve," *arXiv preprint arXiv:2403.02310*, 2024.

[14] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[15] R. Prabhu, A. Nayak, J. Mohan, R. Ramjee, and A. Panwar, "vattention: Dynamic memory management for serving llms without pagedattention," *arXiv preprint arXiv:2405.04437*, 2024.

[16] C. Hu, H. Huang, J. Hu, J. Xu, X. Chen, T. Xie, C. Wang, S. Wang, Y. Bao, N. Sun *et al.*, "Memserve: Context caching for disaggregated llm serving with elastic memory pool," *arXiv preprint arXiv:2406.17565*, 2024.

[17] P. Patel, E. Choukse, C. Zhang, A. Shah, Í. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 118–132.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[19] Y. Zhong, S. Liu, J. Chen, J. Hu, Y. Zhu, X. Liu, X. Jin, and H. Zhang, "Distserve: Disaggregating prefill and decoding for goodput-optimized large language model serving," *arXiv preprint arXiv:2401.09670*, 2024.

[20] R. Qin, Z. Li, W. He, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: Kimi's kvcache-centric architecture for llm serving," *arXiv preprint arXiv:2407.00079*, 2024.

[21] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1080–1096.

[22] Nvidia h100. [Online]. Available: https://resources.nvidia.com/en-us-tensor-core/gtc22-whitepaper-hopper

[23] Advanced Micro Devices, Inc, "Amd cdna™ 2 architecture," AMD, Tech. Rep., 2021. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/instinct-business-docs/white-papers/amd-cdna2-white-paper.pdf

[24] T. Norrie, N. Patil, D. H. Yoon, G. Kurian, S. Li, J. Laudon, C. Young, N. Jouppi, and D. Patterson, "The design process for google's training chips: Tpuv2 and tpuv3," *IEEE Micro*, vol. 41, no. 2, pp. 56–63, 2021.

[25] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. IEEE, 2007, pp. 340–351.

[26] D. Li, S. Yao, Y.-H. Liu, S. Wang, and X.-H. Sun, "Efficient design space exploration via statistical sampling and adaboost learning," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[27] K. Yu, J. Bi, and V. Tresp, "Active learning via transductive experimental design," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 1081–1088.

[28] H.-Y. Liu and L. P. Carloni, "On learning-based methods for design-space exploration with high-level synthesis," in *Proceedings of the 50th annual design automation conference*, 2013, pp. 1–7.

[29] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[30] D. J. MacKay *et al.*, "Introduction to gaussian processes," *NATO ASI series F computer and systems sciences*, vol. 168, pp. 133–166, 1998.

[31] Q. Sun, T. Chen, S. Liu, J. Chen, H. Yu, and B. Yu, "Correlated multi-objective multi-fidelity optimization for hls directives design," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 4, pp. 1–27, 2022.

[32] H. Geng, T. Chen, Y. Ma, B. Zhu, and B. Yu, "Ptpt: Physical design tool parameter tuning via multi-objective bayesian optimization," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 42, no. 1, pp. 178–189, 2022.

[33] P. Xu, S. Zheng, Y. Ye, C. Bai, S. Xu, H. Geng, T.-Y. Ho, and B. Yu, "Ranktuner: When design tool parameter tuning meets preference bayesian optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE*, 2024, pp. 1–7.

[34] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9851–9864, 2020.

[35] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: An analysis and review," *IEEE Transactions on evolutionary computation*, vol. 7, no. 2, pp. 117–132, 2003.

[36] Azure public dataset: Azure llm inference trace 2023. [Online]. Available: https://github.com/Azure/AzurePublicDataset/blob/master/AzureLLMInferenceDataset2023.md

[37] Z. Xie, G.-Q. Fang, Y.-H. Huang, H. Ren, Y. Zhang, B. Khailany, S.-Y. Fang, J. Hu, Y. Chen, and E. C. Barboza, "Fist: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 19–25.

[38] M. Awad, R. Khanna, M. Awad, and R. Khanna, "Support vector regression," *Efficient learning machines: Theories, concepts, and applications for engineers and system designers*, pp. 67–80, 2015.

P-DAC: Power-Efficient Photonic Accelerators for LLM Inference

Wen-Tse Chang*, Chun-Feng Wu*, Yun-Chen Lo†
*Department of Computer Science, National Yang Ming Chiao Tung University, Taiwan
†Department of Computer Science, Harvard University, USA
Corresponding Author: Chun-Feng Wu
E-mail: stupidrara123.cs12@nycu.edu.tw, cfwu417@cs.nycu.edu.tw, yunchenlo@seas.harvard.edu

***Abstract*—As traditional electronic hardware encounters the limitations of Moore's Law, optical computing is emerging as a promising alternative, delivering high data transmission rates, especially beneficial for big data and AI applications. Photonic accelerators, such as the Lightening-Transformer, utilize optical analog signals to accelerate Transformer-based models, achieving exceptional speed and low energy consumption. However, controlling modern optical intensity modulators (e.g., Mach-Zehnder Modulators) requires using electrical analog signals (e.g., voltage values) to adjust the optical signal intensity for realizing optical-based vector inner product calculations. Managing this modulation consumes significant power, as it involves selecting optimal electrical values through an electrical controller and converting digital signals to analog using digital-to-analog converters (DACs). In this work, we introduce P-DAC, a solution designed to reduce DAC power consumption, significantly enhancing the energy efficiency of optical accelerators for Transformer models.**

***Index Terms*—Photonic Accelerators, LLM Inference, Mach-Zehnder Modulators, DAC, Photonic DAC, Energy Efficient**

I. Introduction

Transformer-based large language models (LLMs) have gained significant attention in recent years, requiring substantial computing power and high data transmission bandwidth due to their scale and computational complexity [39], [14], [11], [29]. While modern systems primarily rely on electrical-based computation and interconnection, these devices are approaching the physical and economic limits of Moore's Law [35] and Dennard scaling [8], leading to increased power dissipation and slower performance gains. To address these challenges, integrated photonic accelerators have emerged as a promising alternative, offering ultra-high speed, parallelism, and energy efficiency [31], [20]. Unlike transistor-based chips, photonic devices leverage optical signals to achieve superior theoretical performance [9], [32], with the potential to significantly surpass electronic accelerators as technology advances. Photonic computing and interconnection could revolutionize LLM inference, making them a critical candidate for replacing traditional electronic systems [30], [45].

Photonic interconnections outperform electrical counterparts by offering higher bandwidth, lower latency, and faster, long-distance data transmission without signal degradation [36]. This technology has been widely applied to interconnect processing elements (PEs) in computing systems [16]. SPRINT [22] integrates photonic interconnections with Convolutional neural network (CNNs), leveraging broadcasting to share data across waveguides. PEs perform computations, write results to other waveguides, and send processed data to the global buffer. Similarly, SPACX [23] supports DNN inference by broadcasting data from a global waveguide to local waveguides, facilitating computation in PEs. Processed data is then transferred back to the global waveguide and buffer. Additionally, CAMON [41], a silicon photonic chiplet for manycore processors, reduces communication bottlenecks and enhances energy efficiency by optimizing cache and memory management in optical communication systems, especially in large-scale architectures.

In addition to interconnection, several research teams [21], [44] focus on accelerating CNN tasks, utilizing various photonic tensor core (PTC) architectures, such as the Mach-Zehnder interferometer (MZI) array [33]. The MZI requires singular value decomposition (SVD) and phase decomposition for operand mapping and is capable of performing arbitrary 2-D unitary matrix operations. However, it requires CPU to conduct task mapping, which is time-consuming. For example, mapping a 12×12 matrix takes approximately 1.5 ms for conducting SVD and phase decomposition. To substantially improve computation parallelism, recent researchers have focused on realizing the photonic dot productions in the analog space. For example, Albireo [34] and Lightening-Transformer [45] work on integrating analog-based photonic dot productions with CNN and Transformer, respectively. The key idea is to convert electrical digital signals to photonic analog signals by modulating the intensity and frequency and run dot productions via interleaving photonic signals.

Based on our investigation, analog photonic accelerators, such as Albireo and Lightening-Transformer, rely heavily on Mach-Zehnder Modulators (MZMs) for optical signal modulation. This process requires controllers to compute input voltage values and Digital-to-Analog Converters (DACs) to generate the corresponding voltages for driving the MZMs. However, the power consumption of DACs is significant and increases with higher bit precision, creating a bottleneck for energy-efficient optical computing [10], [43], [12], [25], [24], [3], [40], [38]. To address this, we propose the Photonic-DAC (P-DAC), which eliminates the need for traditional DACs by using approximation to convert optical digital signals into analog signals. Since the P-DAC operates with optical digital signals, it can leverage Wavelength Division Multiplexing (WDM) [13] to enhance the data rate by combining multiple wavelengths into a single waveguide. The P-DAC is particularly well-suited for Large Language Models (LLMs), whose inherent tolerance for numerical errors aligns with the P-DAC's design, often improving performance in such applications.

The key contributions of this work include:

- Proposing the P-DAC, a power-efficient alternative to traditional DAC-based systems, designed for optical accelerators.
- Demonstrating the feasibility of the P-DAC through mathematical derivation and experimental validation.
- Achieving significant energy savings in practical workloads, such as BERT [6] and DeiT [37], with up to 35.4% reduction in power consumption for 8-bit data sizes.

II. Background, Observation, and Motivation

A. Background

1) Large Language Models (LLM) Inference: Large language models (LLMs) are advanced neural networks built on the transformer architecture, which integrates self-attention mechanisms with fully-connected layers to process and understand complex relationships in text. Self-attention enables the model to grasp global context by dynamically incorporating prior information when generating outputs. To produce the next output token, the model computes Query (Q), Key (K), and Value (V) vectors by multiplying the concatenated input and current output tokens with pre-trained weight matrices. Attention scores, derived by comparing Q with K, are used to weight

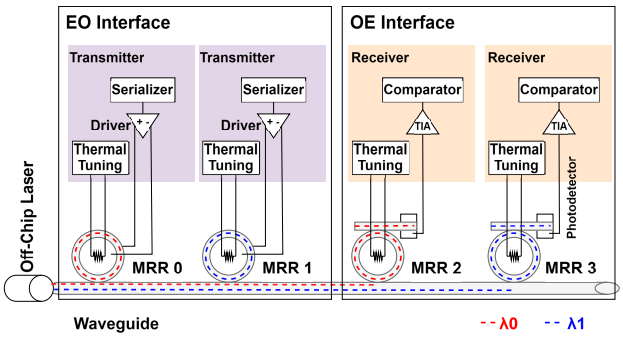


Fig. 1. WDM Technique

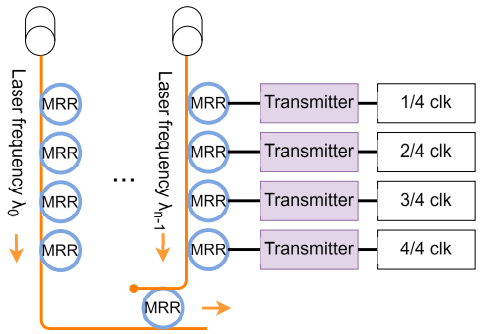


Fig. 2. A 4-bit EO Interface

the V vectors, capturing contextual relationships between tokens. During inference, the KV cache stores precomputed K and V vectors, allowing the model to reuse them efficiently for subsequent tokens without redundant calculations [18], [17], [11].

Running LLM inference demands substantial computing power and high data transmission bandwidth due to the vast scale of the model and the complexity of its computational processes. Large language models consist of billions (e.g., GPT-3), sometimes trillions (e.g., GPT-4 [28]), of parameters, necessitating extensive matrix multiplications for each token processed, particularly during the self-attention and fully-connected layers. The self-attention mechanism involves calculating Query, Key, and Value vectors for every token, performing dot products, and weighting operations, which are computationally expensive, especially for long sequences. Additionally, accessing the KV cache requires high data transmission bandwidth to efficiently retrieve and update cached values as new tokens are processed [17], [11]. Combined, these factors make LLM inference resource-intensive in terms of both computing and data transmission, posing challenges for optimization on modern hardware architectures.

2) Photonic Transmissions: Photonic (or optical) transmissions refer to data transmission using optical signals, with optical waveguides serving as the medium. They effectively overcome the limitations of electrical transmissions, offering high transmission rates and low delays, especially over long distances [42]. However, photonic systems require more complex components and incur additional costs due to the need for Electrical-to-Optical (E/O) and Optical-to-Electrical (O/E) conversions [19], [26], [5]. To achieve high transmission bandwidth, the Wavelength Division Multiplexing (WDM) technique is essential. WDM allows multiple data streams to be transmitted simultaneously over the same optical waveguide by using different wavelengths of light, thereby enabling photonic transmissions to offer significantly higher bandwidth — often one or even two orders of magnitude more than electrical transmissions [7], [1].

To better explain the functionality behind WDM technique, we illustrate a toy example in Figure 1. An MRR [4], [27] functions as a multiplexer (mux) or demux for WDM technique. An MRR filters and selects specific wavelengths by resonating at frequencies influenced by its structure, with precise tuning achieved through temperature adjustments. When the harmonic wavelength of the MRR matches an integer multiple of the wavelength in the waveguide, the light can be captured by the MRR. For example, as shown in figure 1, given the same laser source, MRR 0 and MRR 1 program optical signals to λ_1 and λ_2, while MRR 2 and MRR 3 receive optical signals as receivers to receive λ_1 and λ_2, respectively.

An EO interface typically includes transmitters that control the resonance frequency of each MRR through thermal tuning. CAMON [41] introduces a multi-bit EO interface that encodes n bits of data per laser frequency within a single clock cycle. Figure 2 illustrates a 4-bit EO interface, converting 4-bit electrical data to optical data. The clock cycle is divided into four intervals, with the transmitter modulating the MRR to write data at specific intervals (from $1/4$ to $4/4$ clk). This approach allows each laser frequency to store 4 bits of data within one cycle. On the OE interface side, each receiver includes a photodetector (PD) to convert incoming optical signals into electrical signals by generating current when photons interact with its sensitive material, typically via the photoelectric effect. A transimpedance amplifier (TIA) then amplifies the weak current from the PD into a usable voltage signal. Its output voltage can be expressed as follows:

$$V_{out} = R_f \times I_{in} \tag{1}$$

where R_f is the feedback resistor of the TIA.

3) Photonic Accelerators – Lightening Transformer: Among photonic accelerators, Lightening-Transformer [45] demonstrates the highest known performance in accelerating transformer-based applications, where we illustrate its architecture in Figure 3. Lightening-Transformer leverages the properties of optical analog signals and introduces two main optimizations specifically for the Transformer model: *1) it supports full-range inputs and outputs.* Since Transformer activations are not restricted to non-negative integers, Lightening-Transformer leverages the wave properties of light by treating optical field intensity as a numerical value and using the phase of light to determine positive and negative values, thus achieving full-range support. *2) It enables dynamic matrix multiplication.* "Dynamic" means that operands are generated in real time. Since the Transformer's Q, K, and V vectors are dynamically generated, mapping and device programming would cause significant system stalls on the photonic tensor core if they are much slower than the computation speed. Lightening-Transformer can rapidly compute dynamic input matrix multiplications, addressing this issue.

Lightening-Transformer works on the optical field which indicates the distribution and propagation of light waves in space, encompassing both amplitude and phase information. The amplitude determines the intensity of the light, while the phase determines the positions of the wave's peaks and troughs. Phase differences can cause interference, resulting in the reinforcement or cancellation of light waves. The architecture relies on three critical components: the Mach-Zehnder Modulator (MZM), Phase Shifter (PS), and Directional Coupler (DC). We illustrate all three structures in Figure 3.

- **Mach-Zehnder Modulator (MZM):** By adjusting the path length difference in the split light paths, the recombined light can achieve either constructive or destructive interference [15]. Leveraging these principles, the MZM can modulate both the phase and intensity of the optical field. In analog signal processing, phase modulation enables the MZM to achieve full-range encoding (including positive and negative values), as shown in the following equation:

$$E_{out} = E_{in} cos\phi \tag{2}$$

where ϕ is the phase shift of light. Practically, the MZM is regulated by two input voltages, V_1 and V_2, to adjust the output optical field, the equation is as follows:

$$E_{out} = \frac{E_{in}}{2}\left((1+k)e^{j\frac{\pi V_1}{2V_\pi}} + (1-k)e^{j\frac{\pi V_2}{2V_\pi}}\right) \tag{3}$$

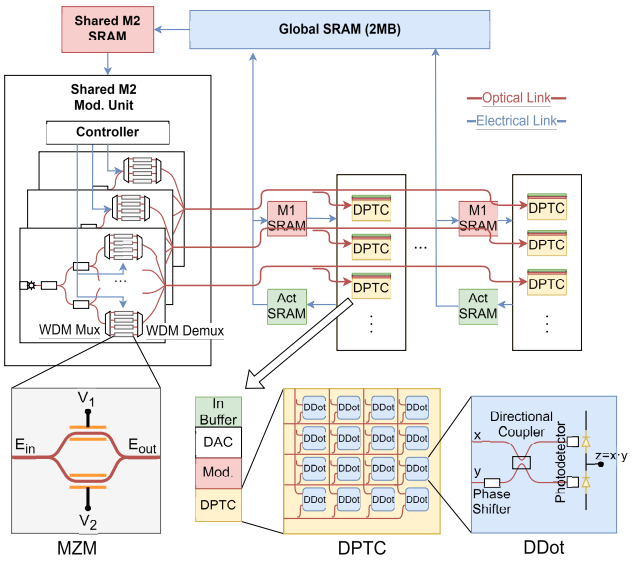


Fig. 3. Lightening-Transformer

where factor k represents the imbalance of splitting. Since the MZM can freely modulate light intensity, it serves as a key component for integrating computation into optical analog signals.

- **Phase Shifter (PS):** PS can change the phase of the light. The following equation represents its transformation formula:

$$x' = e^{j\phi}x \tag{4}$$

where ϕ is the phase shift of light, and x is the light signal.

- **Directional Coupler (DC):** DC can couples light of the same wavelength between two waveguides, transferring energy between them. The device comprises two closely positioned waveguides that enable energy transfer between them. Its transfer matrix of a 2-by-2 DC can be expressed as follows:

$$\begin{pmatrix} t & \sqrt{1-t^2}j \\ \sqrt{1-t^2}j & t \end{pmatrix} \tag{5}$$

where t is the transmission coefficient.

One of the key innovations of the Lightening-Transformer is its ability to compute dot products in the analog domain using a Dynamically-Operated Full-range Dot-Product Unit (DDot). DDot exploits the physical properties of light to efficiently compute the dot product $x \cdot y$. The operation follows the equation:

$$x \cdot y = \sum^{i}(x_i + y_i)^2 - \sum^{i}(x_i - y_i)^2 \tag{6}$$

In other words, by leveraging the physical properties of light to obtain

$$\sum^{i}(x_i + y_i)^2 \quad \text{and} \quad \sum^{i}(x_i - y_i)^2$$

and then taking their difference, we can derive $x \cdot y$. However, before attaining $\sum^{i}(x_i + y_i)^2$ and $\sum^{i}(x_i - y_i)^2$ we first need to find

$$x_i + y_i \quad \text{and} \quad x_i - y_i$$

DDot applies *WDM* technique and assigns each pairs x_i and y_i to the same wavelength. Thus, the light field intensities corresponding to wavelength i are x_i and y_i. The values $x_i + y_i$ and $x_i - y_i$ can be obtained by using a *phase shifter (PS)* and *directional coupler (DC)*:

$$\frac{1}{\sqrt{2}}\begin{pmatrix} x_i + y_i \\ j(x_i - y_i) \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & j \\ j & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & e^{-j\frac{\pi}{2}} \end{pmatrix}\begin{pmatrix} x_i \\ y_i \end{pmatrix}$$

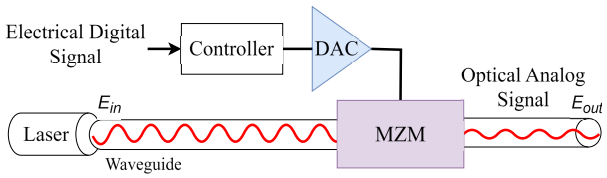


Fig. 4. MZM and DAC

where the leftmost matrix is the transformation matrix of the 50:50 DC, and the second matrix is the transformation matrix of the $-90°$ PS applied to the y-signal. After obtaining the two physical quantities $x_i + y_i$ and $x_i - y_i$, we apply the definition of light intensity:

$$I \propto \frac{1}{2}|E|^2$$

where I is the light intensity, E is the electric field amplitude. Since the photodetector can detect light intensity resulting from the superposition of multiple optical frequencies, we obtain the current by detecting both waveguides:

$$\sum^{i}(x_i + y_i)^2 \quad \text{and} \quad \sum^{i}(x_i - y_i)^2.$$

Finally, using Equation (6) enables us to attain the current intensity $x \cdot y$, and thus finish an inner product by DDot.

Combining DDot with WDM allows multiple wavelengths to utilize the same DDot unit, thereby enhancing computing density and parallelism. The PS and DC components of DDot are fully passive and fixed, which means there is no extra energy consumption because no need for external control and no issues with thermal crosstalk.

B. Observation on Lightening-Transformer's High Power Consumption & Motivation

To achieve high computational parallelism, Lightening-Transformer employs numerous MZMs to convert large volumes of data into optical analog signals simultaneously. Using WDM, these signals are transmitted to DDots for large-scale matrix multiplications. Activating many MZMs requires an equally large number of DACs to regulate their input voltages, as illustrated in Figure 4. However, driving these DACs can lead to significant power consumption. To validate this, we conducted an experimental analysis profiling the Lightening-Transformer's power usage. Results in Figure 5(a) and Figure 5(b) reveal that DACs contribute substantially to overall power consumption: 4-bit DACs in LT-B account for 21.8% (Figure 5(a)), while 8-bit DACs account for 50.5% (Figure 5(b)). These findings highlight that as bit precision increases to enhance model accuracy, DAC power consumption becomes a critical factor.

To achieve power-efficient large-scale matrix multiplication with photonic accelerators, this paper introduces a new design to convert data directly into optical analog signals, bypassing electrical signal processing. We propose replacing traditional DACs with pure-photonic DAC components to regulate MZMs. The primary challenge is configuring MZMs using optical digital signals exclusively through photonic components, eliminating the need for electrical DACs.

III. Photonic Digital-to-Analog Converter (P-DAC)

A. Overview

We present our design in Figure 6, where we propose the P-DAC to control the Mach-Zehnder Modulator (MZM) without using electronic devices, integrating it with Lightening-Transformer. In this approach, we eliminate both the controller and DAC, which are typically required for regulating the MZM in Lightening-Transformer. Additionally, we leverage the high data rate of optical interconnections to efficiently propagate data from the shared M2 SRAM. The design concept behind P-DAC is detailed in Section III-B, with

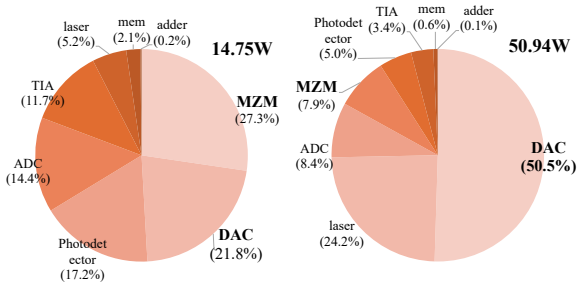


Fig. 5. Power breakdown of LT-B for (a) 4-bit (b) 8-bit precision.

mathematical derivations provided in Section III-B to demonstrate that P-DAC can fully replace the MZM regulation.

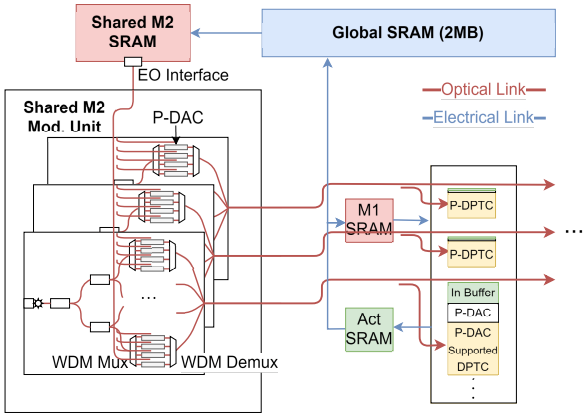


Fig. 6. Integration between Lightening-Transformer and P-DACs.

B. Design Concept behind P-DAC

In this section, we will explain how the Photonic-DAC (P-DAC) replaces traditional DACs, as well as its operational principles and advantages. Notably, in optical analog accelerators, Mach-Zehnder Modulators (MZMs) are typically used to convert electrical signals into optical analog signals. However, this process requires a controller to set the traditional DAC's output as the external voltage for the MZM, which, according to the experiments with the Lightening Transformer, results in significant power consumption from the DAC. In our design, we first convert the electrical digital signals into optical digital signals and then approximate them into optical analog signals, where this approximation process is key to the P-DAC's energy efficiency.

For the optical analog signal, Lightening-Transformer uses a controller to drive the DAC, which then applies the DAC's output voltage to the MZM. However, additional calculations are required to obtain the desired output voltage due to the non-linear nature of Equation (3). In our approach, we eliminate the need for both these calculations and the DAC. Instead, we directly use the P-DAC to convert the optical digital signal into an optical analog signal without prior computation.

It is worth noting that, since we initially convert the electrical digital signals into optical digital signals, we can also utilize the Wavelength Division Multiplexing (WDM) technique to pre-convert data from the memory side into optical digital signals and then transfer it to the P-DAC, thereby saving some energy.

We integrate the MZM into the P-DAC as the critical component for converting signals into optical analog format. Unlike traditional methods, we use only a few optical components to supply the external voltage to the MZM, significantly reducing the energy consumption

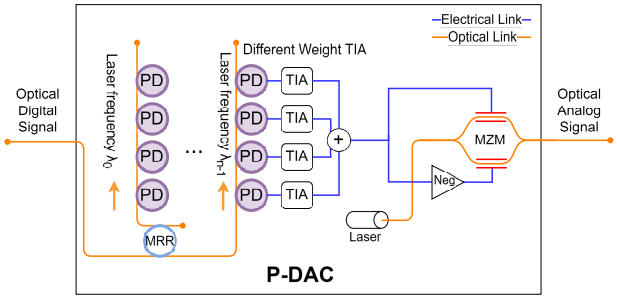


Fig. 7. P-DAC Structure: Integrated with OE interface and MZM

associated with traditional DACs. First, as shown in Figure 2, in the electrical-to-optical (EO) transformation, we adopt the multi-bit EO interface to convert signals into optical digital format. Then, as illustrated in Figure 7, once the optical digital data reaches the P-DAC, we refer to the multi-bit optical-to-electrical (OE) interface to apply different weights to each bit through a TIA and superimpose the voltages of each bit to serve as the MZM's input voltage.

Our P-DAC offers several advantages. It provides a simple implementation with high speed, functioning similarly to a Binary-Weighted DAC but ours is converting digital optical signals directly into analog optical signals. This design theoretically results in low power consumption, as the MZM operates based on relative voltage rather than absolute voltage, making its power usage dependent on the reference voltage. Note that the P-DAC exhibits slight numerical errors. However, since our target application is LLMs, which are inherently tolerant to minor inaccuracies, the P-DAC is perfectly suited for such use cases. By directly controlling the voltage weight of each bit, the input voltage can be calculated without the need for pre-calculations, thereby reducing controller (or CPU) overhead, streamlining the overall system design.

C. Mathematical Derivation for Validating P-DAC

Determining the weight values for the TIA is complex because Equation (3) is nonlinear, so weights cannot be assigned proportionally. Fortunately, given the stochastic nature of LLMs, exact numerical precision is not as critical, as long as the output falls within an acceptable range for human perception. Thus, we determine TIA weights through approximation. Our goal is to control the MZM equation's V_1 and V_2 so that the value of E_{out} becomes the desired analog signal, so we focus on the Equation (3) first.

We first simplify Equation (3) to make equation more readable,

$$E_{out} = \frac{E_{in}}{2}\left(e^{jV_1'} + e^{jV_2'}\right) \tag{7}$$

where we define $V_1' = \frac{\pi V_1}{2V_\pi}$ and $V_2' = \frac{\pi V_2}{2V_\pi}$, assuming no imbalance in splitting so that the k factor can be ignored. Then, applying Euler's formula ($e^{jx} = cosx + jsinx$) to Equation (7):

$$E_{out} = \frac{E_{in}}{2}\left(\left(cosV_1' + jsinV_1'\right) + \left(cosV_2' + jsinV_2'\right)\right) \tag{8}$$

Then we let $V_2' = -V_1'$, allowing the expression to be further simplified:

$$\begin{aligned} E_{out} &= \frac{E_{in}}{2}\left(cosV_1' + jsinV_1' + cos\left(-V_1'\right) + jsin\left(-V_1'\right)\right) \\ &= \frac{E_{in}}{2}\left(cosV_1' + jsinV_1' + cosV_1' - jsinV_1'\right) \\ &= E_{in}cosV_1' \end{aligned} \tag{9}$$

As we can see, Equation (9) is equivalent to Equation (2) when $V_1' = \phi$. Although we successfully simplified Equation (3) to Equation (9), we still need a way to map the data to the desired input voltage after knowing the relationship between input voltage and phase shift. As

the goal of P-DAC is to convert optical digital data to optical analog data, our goal now is to map the analog data to V_1' such that

$$r \propto E_{out} = E_{in} cos V_1' \tag{10}$$

where r can seen as the analog data within the standard interval (-1, 1), which in turn corresponds to the desired analog value E_{out}. After normalizing E_{in} in Equation (10), we obtain:

$$r = cos V_1' \tag{11}$$

To proceed with the approximation, we first need to write down the trivial equation:

$$x = cos\left(cos^{-1}(x)\right) \quad \text{when} \quad -1 \leq x \leq 1 \tag{12}$$

After comparing Equation (11) and Equation (12), we obtain the equation as follows:

$$V_1' = cos^{-1}(r) \quad \text{when} \quad -1 \leq r \leq 1 \tag{13}$$

Thus, when MZM's input voltage V_1' is $cos^{-1}(r)$, from Equation (10), we can ensure that $E_{out} = rE_{in}$. In summary, by adjusting TIA weights of each bit to transform the optical digital data into $cos^{-1}(r)$ and using it as the input voltage to the MZM, we can ensure that $E_{out} = rE_{in}$. For example, if digital value is $0x40$ in 8-bit system, which analog value can be calculated as $\frac{0x40}{2^7-1} = 0.5$, you can set the variable $V_1' = cos^{-1}(0.5)$ and apply Equation (10) to achieve $E_{out} = 0.5E_{in}$, where 0.5 is the analog data of $0x40$ within the standard interval (-1, 1). The only problem now is how to assign TIA weights for $0x40$ to convert it to $V_1' = cos^{-1}(0.5)$. Note that assigning TIA weights is straightforward when the conversion function is linear (e.g., $f(r) = ar + b$), as the mapping is direct. To obtain an approximation, we expand $cos^{-1}(r)$ using a Taylor series:

$$cos^{-1}(r) = \frac{\pi}{2} - \left(r + \frac{1}{6}r^3 + ...\right) \tag{14}$$

by taking first-order approximation from Equation (14), we can get the approximation as follows:

$$cos^{-1}(r) \approx \frac{\pi}{2} - r = f(r) \quad \text{when} \quad 0 \leq r \leq 1 \tag{15}$$

the equation simplifies to a linear form; however, the greatest error occurs at $r = 1$ and $r = -1$, as demonstrated in the following approximation:

$$\left|\frac{1 - cos(f(1))}{1}\right| \approx 15.9\%$$
$$\left|\frac{(-1) - cos(f(-1))}{-1}\right| \approx 15.9\%$$

To further improve the accuracy of the approximation, we introduce an additional linear equation to combine with the result of Equation (15). Given that the highest error occurs at $r = 1$, we identify a linear function that passes through $(1, cos^{-1}(1))$ or $(1, 0)$ to minimize the total error. Note that we focus on the positive domain, as the function is symmetric. Combining this with the result from Equation (15), we derive the following function:

$$f(r) = \begin{cases} \frac{\pi}{2} - r & \text{when} \quad 0 \leq r \leq k, \\ \frac{k-\pi/2}{k-1}(1-r) & \text{when} \quad k < r \leq 1. \end{cases} \tag{16}$$

where $\frac{k-\pi/2}{k-1}(1-r)$ is the expression of linear equation that passes $(0, \pi/2)$ and $(1, 0)$, and k is the intersection of two expressions.

To find the smallest total error, we have to find the minimum of following expression:

$$\left(\int_{r=0}^{k} \left|\frac{cos(\frac{\pi}{2} - r) - r}{r}\right|\right) + \left(\int_{r=k}^{1} \left|\frac{cos\left(\frac{k-\pi/2}{k-1}(1-r)\right) - r}{r}\right|\right) \tag{17}$$

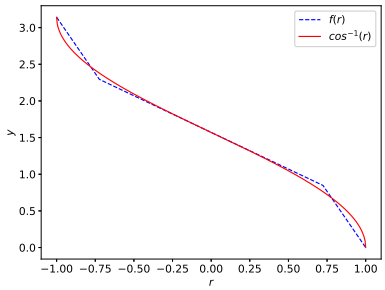


Fig. 8. The Plot of $f(r)$ and $cos^{-1}(r)$. When $r = \pm 0.7236$ has maximum error 8.5%

where $0 \leq k \leq 1$. After running the program to find the optimal k value, we determined that the smallest result occurs when $k \approx 0.7236$ has smallest value. To extend this analysis, we performed the same calculation for negative values of r, and the resulting function is as follows:

$$f(r) = \begin{cases} -3.0651r + 0.07648 & \text{when} \quad -1 \leq r \leq -0.7236, \\ \frac{\pi}{2} - r & \text{when} \quad -0.7236 < r \leq 0.7236, \\ -3.0651(r-1) & \text{when} \quad 0.7236 < r \leq 1. \end{cases} \tag{18}$$

the function plot is shown in Figure 8. The maximum error is at $r \pm 0.7236$, as demonstrated in the following calculation:

$$\left|\frac{-0.7236 - cos(f(-0.7236))}{-0.7236}\right| \approx 8.5\%$$
$$\left|\frac{0.7236 - cos(f(0.7236))}{0.7236}\right| \approx 8.5\%$$

Finally, the function in (18) is now linear, allowing us to easily assign the TIAs' weights using the above function. Note that the function in the P-DAC hardware can be easily decomposed into three parts by adding logic gates in the circuit (e.g., leq).

IV. Performance Evaluation

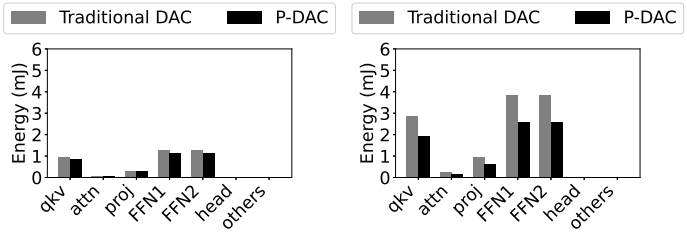
A. Performance Metrics and Evaluation Setup

In this section, we evaluate the performance of the proposed P-DAC design using power consumption and specific workload energy consumption as key metrics. The total power consumption is calculated by accounting for contributions from different modules, measured under typical operating conditions. The proposed design is compared with Lightening-Transformer with using traditional DAC [2], to highlight the advantages of the P-DAC. The workloads used in this evaluation are BERT [6] and DeiT [37]. Note that the power of the P-DAC is calculated based solely on its components and does not include circuit power. Also, since the MZM's power usage depends on the reference voltage, this suggests that circuit power can be further reduced, so we assume that circuit power can be neglected in this analysis.

This work utilizes the source code of Lightening-Transformer (written in python), which is distributed under the GNU General Public License (GPL). Modifications were made to adapt the DAC implementation for our proposed P-DAC.

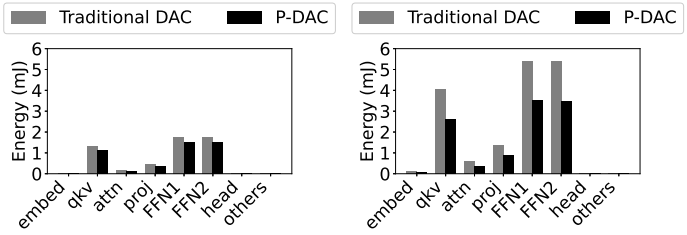
B. Evaluation Results

1) Model Workloads Energy Evaluation: Figure 9 presents the energy breakdown results for BERT-base with a sequence length of 128, while Figure 10 shows the energy breakdown for DeiT with ImageNet1K-224×224 and 197 tokens. Both figures compare the performance of the Lightening-Transformer using a traditional DAC versus our P-DAC. The x-axis represents different operations during inference, while the y-axis indicates the energy consumption



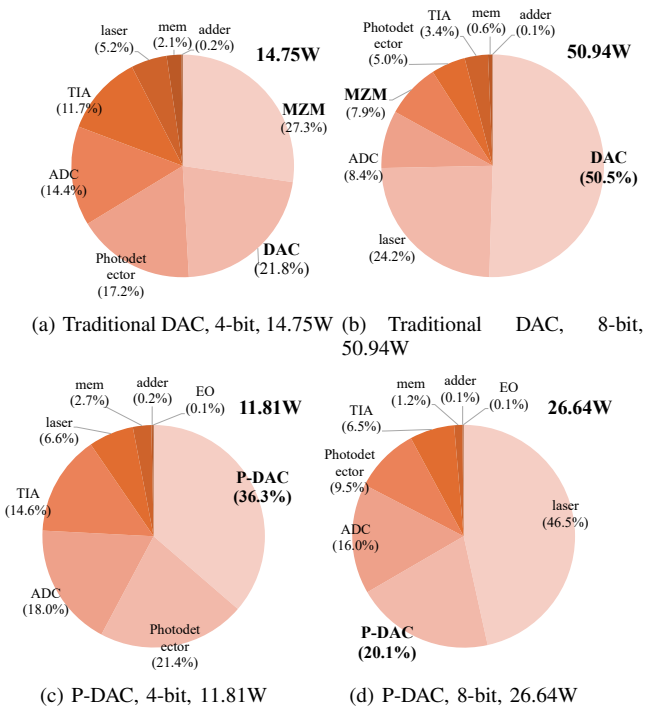
(a) 4-bit, Total: 3.951mJ, 3.510mJ (b) 8-bit, Total: 11.767mJ, 7.961mJ

Fig. 9. Energy breakdown of BERT-base with 128 sequence length



(a) 4-bit, Total: 5.437mJ, 4.740mJ (b) 8-bit, Total: 16.982mJ, 10.974mJ

Fig. 10. Energy breakdown of DeiT with ImageNet1K-224 × 224, 197 tokens



(a) Traditional DAC, 4-bit, 14.75W (b) Traditional DAC, 8-bit, 50.94W

(c) P-DAC, 4-bit, 11.81W (d) P-DAC, 8-bit, 26.64W

Fig. 11. Power breakdown of LT-B

for a single inference run. For BERT in 4-bit data size (Figure 9(a)), replacing the original architecture with the P-DAC design achieves an energy reduction of 11.2%, while for an 8-bit data size(Figure 9(b)), the reduction increases to 32.3%. For DeiT in 4-bit data size (Figure 10(a)), replacing the original architecture with the P-DAC design achieves an energy reduction of 11.2%, while for an 8-bit data size(Figure 10(b)), the reduction increases to 32.3%. The experimental results demonstrate that the proposed P-DAC design offers substantial energy efficiency improvements over traditional DAC-based architectures. It is worth noting that the data movement in the attention accounts for a smaller proportion compared to the FFN, resulting in a larger energy reduction in attention. This is because P-DAC does not affect the energy consumption associated with data movement. For instance, in BERT, the attention achieves an energy savings of 18.3% for 4-bit data and 42.1% for 8-bit data, while the FFN achieves savings of 11.0% for 4-bit and 32.1% for 8-bit. Similarly, in DeiT, the attention saves 19.0% for 4-bit and 42.3% for 8-bit, while the FFN saves 12.6% for 4-bit and 35.1% for 8-bit.

2) Full Compute-bound Scenario Power Evaluation: Figure 11 illustrates the power breakdown of each hardware component, with the P-DAC's power including the integrated MZM. This metric reflects a fully compute-bound scenario where hardware performance is not limited by memory bandwidth. As shown in Figure 11(a) and Figure 11(c), the P-DAC reduces power consumption by 19.9% compared to traditional DAC-based systems for a 4-bit data size. For an 8-bit data size, as depicted in Figure 11(b) and Figure 11(d), the power savings increase substantially to 47.7%.

This setup effectively highlights the capabilities of the P-DAC and provides a projection of its energy consumption under scenarios with sufficient memory bandwidth in the future. While we expect that higher bit sizes will enable more accurate responses from LLMs, the P-DAC significantly reduces energy consumption, yet the majority of the energy consumption remains constrained by the laser. This is because the energy savings provided by the P-DAC are so significant that the relative power consumption of the laser becomes more prominent. Therefore, we anticipate that with future advancements in laser technology, overall power consumption can be further reduced.

V. Conclusion

In this paper, we proposed the P-DAC (Photonic Digital-to-Analog Converter) design as an energy-efficient alternative to traditional DAC-based systems for optical signal processing. Our evaluation demonstrated that the P-DAC significantly reduces power consumption compared to conventional DAC systems, especially as the data size increases. Under a fully compute-bound scenario, the P-DAC achieved an impressive 47.7% reduction for 8-bit data. We also demonstrated the feasibility of the P-DAC design through mathematical inference. For both BERT and DeiT workloads, we observed considerable energy savings in the attention mechanism, with reductions as high as 35.4% for 8-bit data. These improvements highlight the potential of the P-DAC to enable more energy-efficient optical computing systems.

VI. Acknowledgement

We would like to thank Prof. David Brooks and Prof. Gu-Yeon Wei from Harvard University for their thoughtful comments and insightful suggestions. This work was supported in part by the National Science and Technology Council under grant nos. 113-2628-E-A49-021, 113-2640-E-A49-012 and Ministry of Education under Yushan Young Fellow Program.

References

[1] A. M. Alatwi, A. N. Zaki Rashed, and E. M. El-Gammal. Wavelength division multiplexing techniques based on multi transceiver in low earth orbit intersatellite systems. *Journal of Optical Communications*, 45(1):125–135, 2024.

[2] P. Caragiulo, O. E. Mattia, A. Arbabian, and B. Murmann. A compact 14 gs/s 8-bit switched-capacitor dac in 16 nm finfet cmos. In *2020 IEEE Symposium on VLSI Circuits*, pages 1–2. IEEE, 2020.

[3] H.-Y. Cheng, C. Hakert, K.-H. Chen, Y.-H. Chang, J.-J. Chen, C.-L. Yang, T.-W. Kuo, et al. Future computing platform design: A cross-layer design approach. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 312–317. IEEE, 2021.

[4] I. Chremmos, O. Schwelb, and N. Uzunoglu. *Photonic microresonator research and applications*, volume 156. Springer, 2010.

[5] C. Demirkiran, F. Eris, G. Wang, J. Elmhurst, N. Moore, N. C. Harris, A. Basumallik, V. J. Reddi, A. Joshi, and D. Bunandar. An electro-photonic system for accelerating deep neural networks. *ACM Journal on Emerging Technologies in Computing Systems*, 19(4):1–31, 2023.

[6] J. Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] M. M. Eid, A. N. Z. Rashed, S. El-Meadawy, and M. A. Habib. Best selected optical fibers with wavelength multiplexing techniques for minimum bit error rates. *Journal of Optical Communications*, (0):000010151520200239, 2020.

[8] D. J. Frank, R. H. Dennard, E. Nowak, P. M. Solomon, Y. Taur, and H.-S. P. Wong. Device scaling limits of si mosfets and their application dependencies. *Proceedings of the IEEE*, 89(3):259–288, 2001.

[9] J. Gu, C. Feng, H. Zhu, R. T. Chen, and D. Z. Pan. Light in ai: toward efficient neurocomputing with optical neural networks—a tutorial. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6):2581–2585, 2022.

[10] S. Jain, A. Sengupta, K. Roy, and A. Raghunathan. Rxnn: A framework for evaluating deep neural networks on resistive crossbars. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(2):326–338, 2020.

[11] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei. s^3: Increasing gpu utilization during generative inference for higher throughput. *Advances in Neural Information Processing Systems*, 36:18015–18027, 2023.

[12] Y.-W. Kang, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, and S.-Y. Ho. On minimizing analog variation errors to resolve the scalability issue of reram-based crossbar accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(11):3856–3867, 2020.

[13] S. V. Kartalopoulos. Introduction to dwdm technology. *(No Title)*, 1999.

[14] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[15] F. Koyama and K. Iga. Frequency chirping in external modulators. *Journal of Lightwave Technology*, 6(1):87–93, 1988.

[16] A. V. Krishnamoorthy, R. Ho, X. Zheng, H. Schwetman, J. Lexau, P. Koka, G. Li, I. Shubin, and J. E. Cunningham. Computer systems based on silicon photonic interconnects. *Proceedings of the IEEE*, 97(7):1337–1361, 2009.

[17] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626, 2023.

[18] W. Lee, J. Lee, J. Seo, and J. Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 155–172, 2024.

[19] C. Li, M. Browning, P. V. Gratz, and S. Palermo. Luminoc: A power-efficient, high-performance, photonic network-on-chip. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(6):826–838, 2014.

[20] C. Li, X. Zhang, J. Li, T. Fang, and X. Dong. The challenges of modern computing and new opportunities for optics. *PhotoniX*, 2:1–31, 2021.

[21] S. Li, H. Yang, C. W. Wong, V. J. Sorger, and P. Gupta. Photofourier: A photonic joint transform correlator-based neural network accelerator. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 15–28. IEEE, 2023.

[22] Y. Li, A. Louri, and A. Karanth. Sprint: A high-performance, energy-efficient, and scalable chiplet-based accelerator with photonic interconnects for cnn inference. *IEEE Transactions on Parallel and Distributed Systems*, 33(10):2332–2345, 2021.

[23] Y. Li, A. Louri, and A. Karanth. Spacx: Silicon photonics-based scalable chiplet accelerator for dnn inference. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 831–845. IEEE, 2022.

[24] T.-S. Lo, C.-F. Wu, Y.-H. Chang, T.-W. Kuo, and W.-C. Wang. Space-efficient graph data placement to save energy of reram crossbar. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6. IEEE, 2021.

[25] S. Ma, D. Brooks, and G.-Y. Wei. A binary-activation, multi-level weight rnn and training algorithm for adc-/dac-free and noise-resilient processing-in-memory inference with envm. *IEEE Transactions on Emerging Topics in Computing*, 11(2):292–302, 2023.

[26] A. Narayan, Y. Thonnart, P. Vivet, A. Coskun, and A. Joshi. Architecting optically controlled phase change memory. *ACM Transactions on Architecture and Code Optimization*, 19(4):1–26, 2022.

[27] A. Narayan, Y. Thonnart, P. Vivet, and A. K. Coskun. Prowaves: Proactive runtime wavelength selection for energy-efficient photonic nocs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(10):2156–2169, 2020.

[28] OpenAI. Gpt-4 technical report, 2023.

[29] R. Pope, S. Douglas, A. Chowdhery, J. Devlin, J. Bradbury, J. Heek, K. Xiao, S. Agrawal, and J. Dean. Efficiently scaling transformer inference. *Proceedings of Machine Learning and Systems*, 5:606–624, 2023.

[30] S. Rumley, M. Bahadori, R. Polster, S. D. Hammond, D. M. Calhoun, K. Wen, A. Rodrigues, and K. Bergman. Optical interconnects for extreme scale computing systems. *Parallel Computing*, 64:65–80, 2017.

[31] J. Shalf. The future of computing beyond moore's law. *Philosophical Transactions of the Royal Society A*, 378(2166):20190061, 2020.

[32] B. J. Shastri, A. N. Tait, T. Ferreira de Lima, W. H. Pernice, H. Bhaskaran, C. D. Wright, and P. R. Prucnal. Photonics for artificial intelligence and neuromorphic computing. *Nature Photonics*, 15(2):102–114, 2021.

[33] Y. Shen, N. C. Harris, S. Skirlo, M. Prabhu, T. Baehr-Jones, M. Hochberg, X. Sun, S. Zhao, H. Larochelle, D. Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature photonics*, 11(7):441–446, 2017.

[34] K. Shiflett, A. Karanth, R. Bunescu, and A. Louri. Albireo: Energy-efficient acceleration of convolutional neural networks via silicon photonics. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 860–873. IEEE, 2021.

[35] T. N. Theis and H.-S. P. Wong. The end of moore's law: A new beginning for information technology. *Computing in science & engineering*, 19(2):41–50, 2017.

[36] C. A. Thraskias, E. N. Lallas, N. Neumann, L. Schares, B. J. Offrein, R. Henker, D. Plettemeier, F. Ellinger, J. Leuthold, and I. Tomkos. Survey of photonic and plasmonic interconnect technologies for intra-datacenter and high-performance computing communications. *IEEE Communications Surveys & Tutorials*, 20(4):2758–2783, 2018.

[37] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[38] C.-L. Tsai, C.-F. Wu, Y.-H. Chang, H.-W. Hu, Y.-C. Lee, H.-P. Li, and T.-W. Kuo. A digital 3d tcam accelerator for the inference phase of random forest. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[39] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[40] Z. Wang, T. Luo, C. Liu, W. Liu, R. S. M. Goh, and W.-F. Wong. Enabling energy-efficient deployment of large language models on memristor crossbar: A synergy of large and small. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[41] Z. Wang, Z. Wang, J. Xu, Y.-S. Chang, J. Feng, X. Chen, S. Chen, and J. Zhang. Camon: Low-cost silicon photonic chiplet for manycore processors. *IEEE transactions on computer-aided design of integrated circuits and systems*, 39(9):1820–1833, 2019.

[42] S. Werner, J. Navaridas, and M. Luján. Designing low-power, low-latency networks-on-chip by optimally combining electrical and optical links. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 265–276. IEEE, 2017.

[43] Q. Zheng, Z. Wang, Z. Feng, B. Yan, Y. Cai, R. Huang, Y. Chen, C.-L. Yang, and H. H. Li. Lattice: An adc/dac-less reram-based processing-in-memory architecture for accelerating deep convolution neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[44] S. Zheng, J. Zhang, and W. Zhang. High-throughput optical neural networks based on temporal computing. *arXiv preprint arXiv:2303.01287*, 2023.

[45] H. Zhu, J. Gu, H. Wang, Z. Jiang, Z. Zhang, R. Tang, C. Feng, S. Han, R. T. Chen, and D. Z. Pan. Lightening-transformer: A dynamically-operated optically-interconnected photonic transformer accelerator. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 686–703. IEEE, 2024.

FineRR-ZNS: Enabling Fine-Granularity Read Refreshing for ZNS SSDs

Jun Li[1], Zhibing Sha[2], Fan Yang[2], Xiaofei Xu[3], Xiaobai Chen[1], Jieming Yin[1*], Jianwei Liao[2*]
[1]School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China
[2]College of Computer and Information Science, Southwest University, Chongqing 400715, China
[3]School of Computing Technologies, RMIT University, Melbourne 3001, Australia
Emails: {junli, chenxb86, jieming.yin}@njupt.edu.cn, {zhibing.sha, fanyang0329, liaotoad}@gmail.com, xiaofei.xu@ieee.org

***Abstract*—Zoned namespace (ZNS) SSDs are emerging storage devices offering low cost, high performance, and software definability. By adopting host-managed zone-based sequential programming, ZNS SSDs effectively eliminate the space overhead associated with on-board DRAM memory and garbage collection. However, while background read refreshing serves as a data protection mechanism in conventional block-interface SSDs, state-of-the-art ZNS SSDs lack read refreshing functionality to guarantee data reliability. Moreover, implementing zone-level read refreshing in ZNS SSDs incurs significant overhead due to the large volume of valid data movements in a zone, leading to degraded I/O performance.**

To efficiently enable read refreshing for ZNS SSDs, this paper proposes FineRR-ZNS, a fine-granularity read refreshing mechanism for ZNS SSDs. FineRR-ZNS employs a host-controlled fine-granularity read refreshing scheme that selectively determines block-level read refreshing via metadata remapping. A zone reconstruction method is also designed to retrieve remapped data forming complete data during zone-level RR. Specifically, the remapped data after zone reconstruction are still available and prioritized for read access until their respective blocks need the next RR. Evaluation results show that FineRR-ZNS significantly enhances read refreshing efficiency and I/O throughput compared to zone-level read refreshing implemented in the state-of-the-art ZenFS file system.

***Index Terms*—ZNS SSDs, Fine-granularity read refreshing, Remap, Reconstruct, Replica.**

I. Introduction

NAND flash-based solid-state drives (SSDs) have become the mainstream storage media thanks to their advantages of high access throughput [1]. Due to the nature of out-of-place updates, SSDs require a DRAM cache to expedite translations between logical and physical addresses [2]. Then, garbage collection (GC) is employed to reclaim the available space through valid data movements and block erasures, heavily postponing I/O processing [3]. Additionally, the user-invisible flash capacity, known as over-provisioning (OP) space is reserved, to enhance GC efficiency and availability [4].

To eliminate the DRAM and flash costs and mitigate the GC impacts on I/O processing, a new storage interface, zoned namespace (ZNS) is introduced for flash-based SSDs [5]. Since ZNS interface only supports sequential writes, coarse metadata are maintained, with a zone-level mapping table in file system and a zone-to-block one in ZNS SSDs. This indicates the requirement of DRAM cache inside SSDs can significantly be reduced. Since the host takes charge of the basic operations of read, write and GC, also called zone reset in ZNS SSDs, OP spaces can be removed that are needed by the in-device garbage collection of conventional SSDs [6]. Besides, as application data with similar lifetime are gathered together in the same zone by the host, some fully-invalid zones can be reclaimed directly without heavy data movements [7].

*Corresponding authors: Jieming Yin and Jianwei Liao.

Different from garbage collection, read refreshing, also called read reclaim (RR) is a data rewriting method fighting against read disturb that is a physical circuit noise phenomenon [8]. In other words, while the primitive purpose of garbage collection is to reclaim the space, read refreshing is employed to ensure the data reliability. Therefore, the target of read refreshing is generally full of valid data [9]. To the best of our knowledge, state-of-the-art ZNS-compatible file systems, such as ZenFS [10] and F2FS [11], lack support for host-controlled background data refreshing mechanisms. However, several inherent circuit-level noises, such as retention disturb and read disturb still exist [12].

To ensure the data reliability, the background read refreshing method remains necessary in the context of ZNS SSDs. This paper first explores introducing read refreshing functionality into ZNS SSDs to ensure data reliability. Since zone-granularity management in ZNS SSDs induces a large number of valid data movements during RR operations and thus aggravates I/O delays, FineRR-ZNS is proposed to enable a fine-granularity read refreshing method for ZNS SSDs. It can selectively adopt block-level RR by *Zone Remapping* module, and later combine remapped data and original data by *Zone Reconstruction* module. Furthermore, frequently-read remapped data can be kept as replicated data until its block reaches the next RR threshold. Consequently, FineRR-ZNS can eliminate data movements during RR and result in better I/O performance. This paper offers the following contributions:

- We introduce read refreshing method for ZNS SSDs, to ensure data reliability induced by read disturb.
- We develop FineRR-ZNS, a fine-granularity read refreshing management at the host side for ZNS SSDs, which can eliminate the total number of valid data migrations during read refreshing, thus improving I/O performance.
- We design two functionalities of zone remapping and zone reconstruction to implement FineRR-ZNS without noticeable overhead.
- We conduct experiments on FEMU SSD emulator using RocksDB applications. Compared to zone-management read refreshing method implemented in state-of-the-art Zenfs, FineRR-ZNS can improve RR efficiency and I/O throughput by 36.4% and 28.2% on average, respectively.

II. Background Knowledge

Due to the nature of out-of-place updates, SSDs utilize flash translation layer (FTL) within their internal controller to allocate data to expected location in flash arrays, and maintain a mapping table for the relationship between physical and logical addresses [13]. To reclaim the space occupied by outdated data marked as invalid pages, garbage collections (GCs) are triggered when available space becomes insufficient [14]. Specifically, all the valid data in the

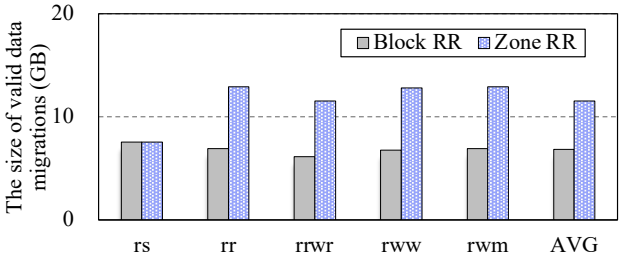


Fig. 1. Comparison on valid data migrations between zone-level and block-level read refreshing.

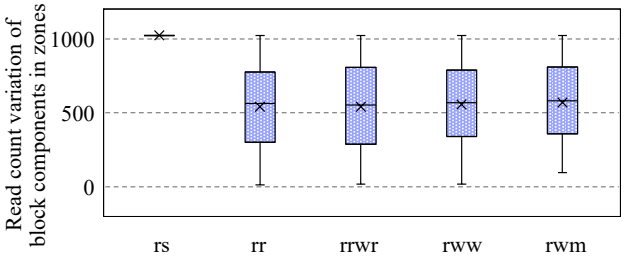


Fig. 2. Box plots of block read count variations during zone-level read refreshing in ZoneRR-ZNS.

selected block are copied to new locations, and then erases and returns the block to an available free state [15]. However, garbage collection remains a persistent challenge in block-interface SSDs, as it is handled by SSDs and transparent to the host [16]. This makes it difficult for the host to predict SSD performance. To mitigate the negative impact of garbage collection, ZNS has been introduced into SSDs, shifting the main software components to the host [17]. Consequently, the host provides software-defined capabilities that handles data placement and I/O scheduling, while SSDs are only responsible for basic read, write, and erase operations [18].

Since ZNS SSDs only support strict sequential writes, a coarse-grained zone-to-block mapping table is maintained inside ZNS SSD device that eliminates DRAM space over than 10X [19], rather than a page-level mapping table. Additionally, valid data migrations with multiple reads and writes during garbage collections are controlled by host, ensuring performance predictability of SSDs. Consequently, the reserved over-provisioning space can also be removed, which comprising 7% to 28% of the advertised storage capacity, further decreasing the SSD cost [5].

In ZNS SSDs, the host manages the data placement at the granularity of zone, comprising a large number of parallel unit with flash blocks. Considering the files with the same lifetime are preferred to be placed in the same zone, the valid data movements can be minimized in ZNS SSDs during garbage collections [20]. However, hot read data in the specific zone accumulate over time, leading to read disturbance issues. Additionally, a high concentration of valid data in a zone significantly impacts I/O efficiency during data refreshing. To address these challenges, we then conduct a preliminary study on read refreshing in ZNS SSDs and analyze the observations.

III. Preliminary Study and Motivation

The background read refreshing method is commonly employed in conventional SSDs, with its threshold typically set based on the worst-case error scenario [21], [22], [23]. For example, the SLC cell has strong reliability, while QLC cell is prone to several disturb-induced errors. Consequently, the block read threshold of read refreshing is correspondingly reduced from 100K to 1K [9], [24]. Since the QLC is widely used in commercial products [25], this paper considers QLC as the default flash media. Considering that the current framework of ZNS SSD system does not support read refreshing module in both file system and SSD controller, we first do a preliminary study on how to implement the read refreshing method for ZNS SSDs.

A. Preliminary Study on Read Refreshing in ZNS SSDs

SSD-side block-level refreshing. An intuitive approach is to implement a read refreshing module directly within SSDs, similar to traditional block-based SSDs. Specifically, it requires both the over-provisioning and DRAM cache spaces for managing the data mapping inside SSDs. However, this approach goes against the foundational intent underlying ZNS design. This indicates that additional over-provisioning and metadata spaces and non-transparent internal operations of SSDs are employed in this design. Considering this approach in turn increases both DRAM and flash cost of ZNS SSDs and degrades performance predictability, it is excluded from consideration in this paper.

Host-side zone-level refreshing. Following the basic principle of ZNS SSDs, the host takes charge of most of software modules. Consequently, the baseline refreshing method is employed as a part of ZNS-aware file system at zone granularity, called *ZoneRR-ZNS*, and we use this method as baseline read refreshing. Specifically, it recalls the functionality of garbage collection, awaking a separated thread termed as *RRWorker*. Note that, since the host does not know the information below zone granularity, a block-level read counter for triggering read refreshing must be kept within SSD controller, requiring around tens of kilobytes that will be described in Section V-C. When the SSD responds with data to the host, it also includes a refreshing flag. If the flag is on, RRWorker is activated to perform valid data migrations and zone resets.

B. Motivational Experiments

We implemented ZoneRR-ZNS in FEMU [26] by running applications of RocksDB [27] with the ZNS-compatible file system plugin of ZenFS [10]. The read refreshing threshold of block read count is set as 1,024, 40 times of the number of pages per block, referred to [24]. The detailed specifications will be described in Section V-A. Once the read refreshing is triggered, we collected the statistic for all blocks within the zone (labeled as Zone RR), and tracked the block-level read refreshing regardless of ZNS write constraints (Block RR). As presented in Figure 1, the benchmark of *readseq (rs)* has entirely sequential reads, so that the zone-level RR does not amplify valid data movements in this scenario. More importantly, we can observe that zone-level read refreshing results in an average of 70% more valid data migrations compared to block-level read refreshing. To clearly show the root cause of increased data movements, we further collect block read count variations across zones in ZoneRR-ZNS.

We utilize box plots to illustrate the variations in block read counts across zones. As shown in Figure 2, the large range of block read counts can indicate that while the current RR block has reached the read count threshold, most blocks are still far from it. In other words, a major part of blocks within the zone do not require immediate data refreshing at that time.

Motivated by aforementioned observations from preliminary studies on ZoneRR-ZNS, this paper proposes a novel read refreshing method for ZNS SSDs, aiming at eliminating the number of concen-

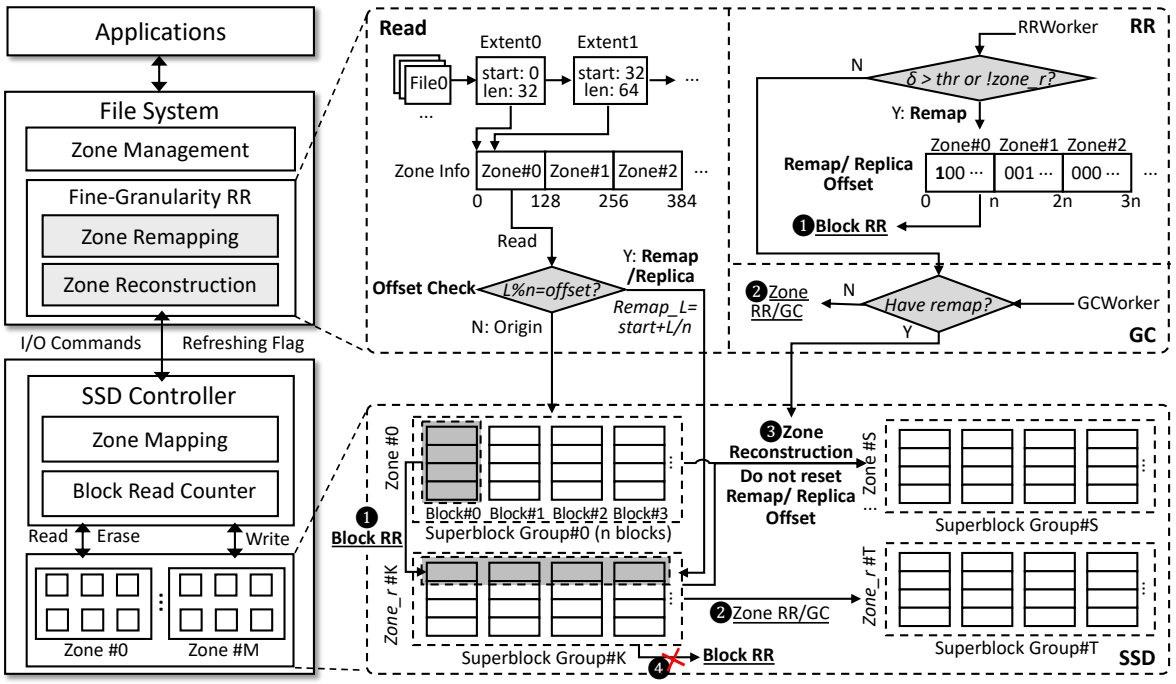


Fig. 3. The overview of FineRR-ZNS, designed for supporting a fine-granularity read refreshing (RR). During read refreshing is required, it is checked by *Zone Remapping* module for employing ❶*Block RR*. When data are required for read requests, they are also checked for fetching data from original zone or remapped zone. Once RR or garbage collection (GC) targets at original zone whose data were previously remapped, ❸*Zone Reconstruction* module is triggered. Note that default ❷zone-level RR and GC is employed if remapping is not enabled, and the second remapping on zone_r is not permitted (❹).

trated data movements during RR operations, eventually enhancing overall I/O performance.

IV. FineRR-ZNS Design

A. Overview

We propose FineRR-ZNS, a fine-granularity read refreshing management for ZNS SSDs. Figure 3 presents the overview of FineRR-ZNS, highlighting its design architecture and illustrating its functionalities. Since blocks within a zone do not necessarily reach the read refreshing threshold simultaneously, FineRR-ZNS selectively refreshes only the required blocks by *zone remapping* module, thereby avoiding unnecessary data movements for other blocks within the zone. It can minimize data movements during RR, thus mitigating I/O congestion and improving I/O throughput. In order to maintain the metadata of remapped data without complex data structures, *zone reconstruction* module is introduced to combine remapped data with original data into the normal zone when the original zone needs refreshing. Additionally, the remapped data are retained as *replicated data*, that are prioritized for read accesses. These data will eventually be directly invalidated without migrations until their block reaches the RR threshold. Consequently, both data movements during RR and I/O congestion caused by RR can be minimized.

B. Benefit Analysis of FineRR-ZNS

The basic intent of FineRR-ZNS is to minimize unnecessary read refreshing operations and mitigate the postpone in I/O processing. Consequently, a block-level read refreshing is supported by zone remapping module that will be detailedly described in Section IV-C. However, when the original zone needs doing zone-level read refreshing, the remapped data by previous block-level RR should also be recovered into sequential data by zone reconstruction module. That is to say, the **cost** of FineRR-ZNS is the data movement number of remapped data in RR block. Fortunately, many unnecessary data movements in zone-level read refreshing that were shown in Figure 1 can be eliminated. Furthermore, since the flash blocks with remapped data can endure additional read accesses without requiring immediate refreshing until reaching their RR threshold, the **benefits** include that RR of other blocks with low read counts can be delayed, and the remapped data can endure further read accesses. Therefore, we can define the expectation of cost-benefit model for data movements by RR operations as follows.

$$\begin{aligned} E &= (\sum_{i=0}^{n-1} \frac{thr - RC_i}{thr} \times N_{page} + \frac{thr}{thr} \times N_{page}) - N_{page} \\ &= \sum_{i=0}^{n-1} \frac{thr - RC_i}{thr} \times N_{page} \end{aligned} \tag{1}$$

where thr is the RR threshold, N_{page} is the page number per block, and RC_i is the read count of ith block in a zone. Note that, $\frac{thr}{thr} \times N_{page}$ is one of the benefit by the remapped data, since it can endure the read count of thr as replica data, though the zone reconstruction is performed. Thus, E is always a non-negative value, and can be considered as a decreasing function of $\sum_{i=0}^{n-1} RC_i$. In other words, FineRR-ZNS enables block-granularity RR when a certain number of blocks in a zone do not reach the refreshing threshold. To support FineRR-ZNS, zone remapping management and zone reconstruction management are designed in §IV-C and §IV-D, respectively.

C. Zone Remapping

Similar to the baseline method of ZoneRR-ZNS described in Section III-A, a separated thread RRWorker in file system and a block read counter in SSD controller are utilized for implementing

RR. Apart from a zone-level read refreshing (❷zone RR) flag in ZoneRR-ZNS, FineRR-ZNS introduces individual block flags within a zone for block-level read refreshing (❶block RR). This section will describe **when** to determine block RR and **how** to conduct it in ZNS SSDs.

Determining block RR. Once read refreshing is required, FineRR-ZNS determines whether to perform zone-level or block-level read refreshing. Motivated by the observation in Figure 2 and the benefit analysis in Equation 1, FineRR-ZNS can improve the space wastage induced by zone RR when the other blocks are far from the RR threshold. Thus, we use $\sum_{i=0}^{n-1}(thr - RC_i)$ as an indicator δ to enable the block RR. Considering that a small value of δ may not notably benefit RR operations, the threshold for δ defined as equal to the RR threshold, set to 1,024 in the experimental setting.

Handling block RR. Once block RR is required, zone remapping module in FineRR-ZNS migrates the data to the specific zone, such as *Zone #K* as shown in Figure 3, and modified the flag of refreshed flash block in its zone in *remap offset*. Importantly, zone remapping module maintains the metadata of remapped data. It holds offset of remapped block as a bit map for checking $L\%n = offset$, as well as the start address of remapped zone, denoted as $start$. Consequently, the remapping address can be calculated by $Remap_L = start + L/n$, where L is the original logical address of the request, and $start$ is the starting address of the remapped data. The unrefreshed data can be accessed in the original zone as default. That is to say, a read request requires checking the remapping information. If the remapping offset is matched, the data should be fetched from the remapped zone, termed as *zone_r*, based on their addresses and offsets. Otherwise, the default read mode is employed. Note that the frequently read data with block granularity have been distributed in different flash blocks in zone_r, so that we argue that the read accesses on zone_r are uniformly distributed across blocks.

To facilitate the management of remapped data, the data should be relocated to the dedicated zone. When the zone remapping is enabled, it first checks whether the left space in the current open zone_r is enough. If not, it applies for a new zone from free zone list and assigns it as zone_r.

D. Zone Reconstruction

When the original zones need to carry out zone-level RR or GC, the complete data, including the remapped data must be integrated. Otherwise, several discontinuous data left in the original zone cannot be migrated to another zone due to inconsistency with ZNS writes. In our design of FineRR-ZNS, ❸zone reconstruction is enabled when the original zones having remapped data are selected as targets of zone-level RR or GC. If activated, zone reconstruction is employed by retrieve all the associated remapped data and then merge them with the original data.

Although zone remapping module can reduce data movements during a given RR operation compared to zone-level RR, FineRR-ZNS utilizes the more space to additionally occupy the zone_r regions, as analyzed in Equation 1. To eliminate the space occupation by remapped data, zone reconstruction module not only reorganize the remapped and original data, but also keeps the remapped data in a valid state for further accesses. Thus, the remap offset also serves as the *replica offset* without resetting to 0 until the replicated data blocks reach the RR threshold.

Additionally, in order to simplify the remapping metadata management, the second remapping on the same data is not permitted, as shown in ❹ in Figure 3. We argue that the frequently read data is relatively uniform within zone_r, as the remapped data have been distributed from individual blocks to multiple blocks, eliminating the need for another block RR in remapped data zones. Consequently, only the data in normal zone can be remapped by FineRR-ZNS, while those in zone_r can only be reconstructed with original data (❸) or be employed in zone-level RR and GC (❷). Note that the zone-level RR or GC on zone_r migrates data to another zone_r, and the start address *start* should be updated correspondingly.

TABLE I
EXPERIMENTAL SETUP

Hardware Platform	
(FEMU, Kernel Version)	*(8.0, 5.10.0)*
(*Channel, Chip, Die Plane, Block, Page)*	*(8, 2, 2, 4, 256, 256)*
(Page Size, Cell Density)	*(4KB, QLC)*
(Read, Program, Erase Latency)	*(40μs, 200μs, 2ms)*
Software Platform	
(RocksDB, ZenFS Version)	*(7.7.3, 2.0.1)*
(Zone size, Number of Zones)	*(128MB, 256)*
(GC, RR Threshold)	*(20%, 1,024)*

V. EXPERIMENT AND EVALUATION

A. Experiment Setup

We evaluate our proposed method by using FEMU [26], a QEMU-based SSD emulator that supports ZNS SSD functionality and full system stack, which has been widely utilized for recent ZNS SSD researches [28], [7], [29], [30]. It runs Rocksdb applications [31] with Zenfs in-application file system [10]. The detailed specifications of hardware and software platforms are shown in Table I. Specifically, the flash parameters are based on [30] and zone settings are followed by [28]. Each superblock comprises 128 blocks having same offset across all flash planes, with both the zone size and superblock size configured to 128MB. The thresholds of garbage collection and read refreshing are referred to [10] and [24], respectively.

Additionally, a built-in benchmark tool *db_bench* of RocksDB is used to generate different workloads [27]. We evaluate 5 kinds of workloads, each with 60 million key-value pairs of 1KB size, referred to [19]. Specifically, *readseq (rs)*, *readrandom (rr)*, *readrandomwriterandom (rrwr)*, *readwhilewriting (rww)*, *readwhilemerging (rwm)* are used for evaluating effectiveness of the proposed method.

We employ the following comparison counterparts to validate the effectiveness of our proposal.

- *NoRR-ZNS*, which is a pure ZNS SSD with Zenfs, without enabling read refreshing method. We set this method as a comparison to show the upper limit of I/O performance, though it cannot ensure the data reliability.
- *ZoneRR-ZNS*, which has been described in Section III. A host-controlled zone-level read refreshing method is implemented via calling the functionality of garbage collection.
- *FineRR-ZNS*, which is the new proposed method. It supports fine-granularity read refreshing for ZNS SSDs, implemented by two modules of remapping and reconstruction.

B. Evaluation and Result

To measure the validity of the proposed mechanism that aims to improve RR efficiency and boost I/O performance for ZNS SSDs, we use the following metrics in our tests: (a) *RR efficiency* including valid data movements and erase counts, and (b) *I/O performance* to measure the quality of service.

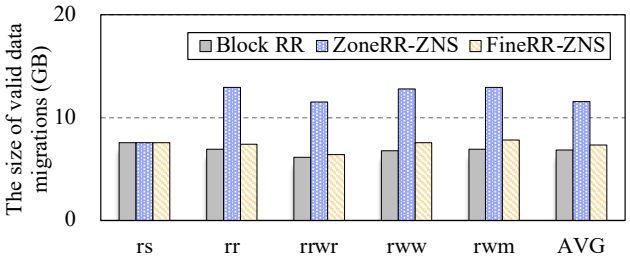


Fig. 4. The comparison of valid data movement during read refreshing after running selected RocksDB workloads.

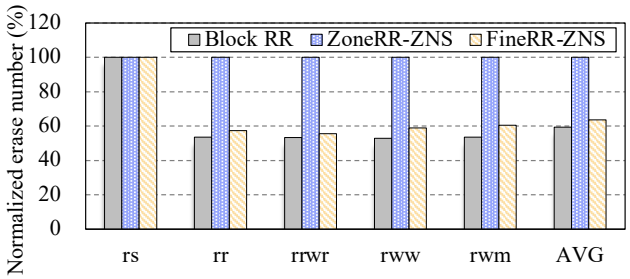


Fig. 5. The comparison of block erasure number after running selected RocksDB workloads.

1) RR efficiency: The basic idea of FineRR-ZNS is to avoid unnecessary valid data movement in RRs, and the statistic on it is shown in Figure 4. Except for similar performance in benchmark of *rs* with full sequential reads, FineRR-ZNS significantly reduce the amount of valid pages by 41.8% on average, compared with ZoneRR-ZNS. This is because FineRR-ZNS offers a fine-granularity read refreshing, eliminating the most of data migrations induced by default zone-level read refreshing, as analyzed in Equation 1.

Additionally, we also include *Block RR* that was presented in Section III-B, to show the lower limit of RR statistics, though it does not obey the sequential write constraint of ZNS interface. As seen, the performance of FineRR-ZNS in valid data movements is approaching the lower limit presented in Block RR. The remaining 7.1% of data movements result from two factors: (a) the threshold set for enabling block-level RRs, which avoids excessive and performance-inefficient block-level operations but inevitably induces a few more data movements during zone-level RR, and (b) the replicated data retained in the remapping zone, which not yet endure enough read accesses at the end of evaluated benchmarks, leading to additional data migrations.

The erasure count is a key metric reflecting SSD lifetime, due to the erase limit of flash memory. Figure 5 presents the results of erasure count induced by read refreshing operations. As unveiled, FineRR-ZNS can notably decrease the number of block erasure by an average of 36.4% compared with ZoneRR-ZNS. This is because FineRR-ZNS can effectively eliminate the valid data movements, resulting efficient block erasure.

2) I/O Performance: FineRR-ZNS can improve RR efficiency and thus mitigate delays in I/O processing caused by excessive RRs. I/O throughput is an indicator to reflect the efficiency of FineRR-ZNS, as seen in Figure 6. As shown, FineRR-ZNS can improve data access throughput by 28.2% on average, compared with ZoneRR-ZNS. We also include the NoRR-ZNS method that is not enabled read refreshing method, to show the upper limit of the scenarios without

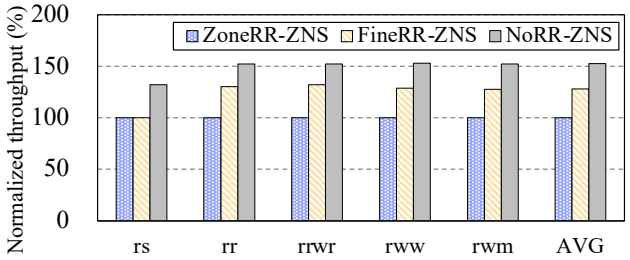


Fig. 6. The comparison of I/O throughput after running selected RocksDB workloads.

TABLE II
ADDITIONAL SPACE OVERHEAD FOR 32GB ZNS SSD REQUIRED IN ZONERR-ZNS AND FINERR-ZNS.

Methods	*ZoneRR-ZNS*	*FineRR-ZNS*
SSD overhead	40KB	40KB
Host overhead	0.032KB	68.07KB

delays by RR operations. But it is important to note that NoRR-ZNS cannot guarantee data reliability, since read disturb may lead to data loss. Even considering the NoRR-ZNS method, FineRR-ZNS unveils attractive I/O performance with the data protection mechanism of RR enabled.

C. Overhead Analysis

Metadata overhead. The space overhead mainly consists of remapping metadata to enable fine-granularity read refreshing in ZNS SSDs. The detailed space overhead is presented in Table II. To deploy block-level counters in SSD, both ZoneRR-ZNS and FineRR-ZNS need 40 kilobytes. Additionally, they requires a refreshing flag to trigger RR operation in host, and FineRR-ZNS needs an additional flag to direct whether block- or zone-level refreshing based on δ. As for remapping metadata, remap offset needs flags of each block, and the start address of remapped zone *start* is recorded by zone ID and zone offset, involving 4+4 bits for respectively representing 256 zones and 256 pages. In total, ZoneRR-ZNS needs 1-bit flag for each zone, while FineRR-ZNS requires 2 bits for determining RR, 128 bits for remap/replica offset table, and additional 128×16 bits for remapped start address. Fortunately, these overheads are negligible when managed by the host.

Time overhead. FineRR-ZNS only adds a few judgments to determine the original mapping or zone remapping. Additionally, FineRR-ZNS needs computing δ. Because δ is calculated through several addition and subtraction operations, its computing overhead is minimal. In summary, FineRR-ZNS has negligible additional time overhead to enable fine-granularity read refreshing for ZNS SSDs.

VI. RELATED WORK

Garbage collection optimizations in ZNS SSDs. Since the space reclaim granularity of ZNS SSDs is zone, several researches focus on garbage collection optimizations. Choi et al. [32] proposed to separate hot and cold data during garbage collections, and thus the later GC efficiency can be enhanced. Huang et al. [29] introduced smaller zone size to reduce data migration in garbage collections, and proposed several optimization methods for alleviating the degraded performance due to underutilized parallelism. Long et al. [28] proposed WA-Zone to avoid unnecessary erases upon unused blocks and balance inter- and intra-zone wear in ZNS SSDs. However, the

background read refreshing method is not employed for ZNS SSDs for ensuring the reliability of ZNS SSDs. This paper proposes to implement the read refreshing method for ZNS SSDs and optimize it via fine granularity.

Read refreshing optimizations in conventional SSDs. The read refreshing method is a background method in conventional SSDs. Since a large number of data movements block the front-end I/O responses in SSDs, several methods proposed to manage the frequently read data in dedicated blocks. Liu et al. [33] proposed *Read Leveling*, a strategy that allocates specific blocks, known as shadow blocks, to mitigate the effects of read disturb. Since the blocks contain several invalid data pages or free pages, both of which are unaffected by read disturb, Read Leveling only refreshes the frequently read accessed data in shadow blocks. Additionally, Wu et al. [34] proposed to reprogram TLC blocks to MLC blocks, since MLC blocks can enable a larger margin between different states to tolerate read disturb. However, these methods decrease the space utilization, caused by sacrificing available dedicated blocks to fight against read disturb.

For further eliminating the negative impacts of read refreshing towards I/O response and optimizing the read data allocation management, Li et al. [35] introduced reinforcement learning to utilize idle intervals for partial read refreshing, mitigating the postpones on I/O responses. Zhao et al. [36] optimized data allocation by estimating optimal read limits based on initial read counts and P/E cycles of flash blocks, that can eliminate the frequency of read refreshing operations and ECC time overhead. Zhang et al. [37] proposed redistributing hot read data across multiple blocks with cold data to limit read concentration, thereby decreasing RR frequency.

These optimization methods target at the conventional block-interface SSDs. In ZNS SSDs, enlarged zone-level management and sequential programming management prevent enabling efficient read refreshing, and our proposal of fine-granularity read refreshing can significantly improve RR efficiency and I/O processing.

VII. Conclusion

This paper proposes an efficient read refreshing for ZNS SSDs to fight against read disturbs. The primary objective is to eliminate unnecessary valid data movements under zone-level management. To this end, a fine-granularity read refreshing method is designed with two key components of zone remapping and zone reconstruction. Consequently, unnecessary data migrations by zone-level read refreshing can be significantly reduced, thereby mitigating the I/O postpones. Evaluation results measure that the proposed method can yield attractive improvement on I/O throughput by an average of 28.2% and space utilization by up to 44.4% compared to zone-level read refreshing scheme.

Acknowledgment

We sincerely thank the reviewers for their valuable suggestions. This work was partially supported by National Key R&D Program of China Grant No. 2023YFB4404400, National Natural Science Foundation of China Grants No. 62102193, No. 92473205, No. 61872299, State Key Laboratory of Computer Architecture (ICT CAS) Grant No. CARCH202106, and Natural Science Research Start-up Foundation of Recruiting Talents of Nanjing University of Posts and Telecommunications Grant No. NY224062.

References

[1] L. Zuolo, C. Zambelli, R. Micheloni *et al.*, "Solid-state drives: Memory driven design methodologies for optimal performance," *Proceedings of the IEEE*, vol. 105, no. 9, pp. 1589–1608, 2017.

[2] R. Micheloni, "Solid-state drive (ssd): A nonvolatile storage system," *Proceedings of the IEEE*, vol. 105, no. 4, pp. 583–588, 2017.

[3] T. Lange, J. Naor, and G. Yadgar, "Offline and online algorithms for ssd management," *Communications of the ACM*, vol. 66, no. 7, pp. 129–137, 2023.

[4] F. Wu, J. Zhou, S. Wang *et al.*, "Fastgc: Accelerate garbage collection via an efficient copyback-based data migration in ssds," in *Proceedings of the 55th Annual Design Automation Conference (DAC)*, 2018, pp. 1–6.

[5] M. Bjørling, A. Aghayev, H. Holmberg *et al.*, "ZNS: avoiding the block interface tax for flash-based ssds," in *USENIX Annual Technical Conference (ATC)*, 2021.

[6] K. Han, H. Gwak, D. Shin *et al.*, "ZNS+: advanced zoned namespace interface for supporting in-storage zone compaction," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2021.

[7] Y. Wang, Y. Zhou, Z. Lu *et al.*, "Flexzns: Building high-performance zns ssds with size-flexible and parity-protected zones," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. IEEE, 2023, pp. 291–299.

[8] K. Ha, J. Jeong, and J. Kim, "An integrated approach for managing read disturbs in high-density nand flash memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 7, pp. 1079–1091, 2015.

[9] J. Liao, J. Li, M. Zhao *et al.*, "Read refresh scheduling and data reallocation against read disturb in ssds," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 2, pp. 1–27, 2022.

[10] Western Digital Corporation, "Zenfs github repository," https://github.com/westerndigitalcorporation/zenfs, 2024.

[11] C. Lee, D. Sim, J. Hwang *et al.*, "{F2FS}: A new file system for flash storage," in *13th USENIX Conference on File and Storage Technologies (FAST 15)*, 2015, pp. 273–286.

[12] Y. Luo, S. Ghose, Y. Cai *et al.*, "Improving 3d nand flash memory lifetime by tolerating early retention loss and process variation," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 3, pp. 1–48, 2018.

[13] W. Zhang, Q. Cao, H. Jiang *et al.*, "Pa-ssd: A page-type aware tlc ssd for improved write/read performance and storage efficiency," in *Proceedings of the 2018 International Conference on Supercomputing*, 2018, pp. 22–32.

[14] C. Gao, L. Shi, Y. Di *et al.*, "Exploiting chip idleness for minimizing garbage collection—induced chip access conflict on ssds," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 2, pp. 1–29, 2017.

[15] J. Cui, Y. Zhang, J. Huang *et al.*, "Shadowgc: Cooperative garbage collection with multi-level buffer for performance improvement in nand flash-based ssds," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1247–1252.

[16] J. Min, C. Zhao, M. Liu *et al.*, "{eZNS}: An elastic zoned namespace for commodity {ZNS}{SSDs}," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, 2023, pp. 461–477.

[17] D. Seo, P.-X. Chen, H. Li *et al.*, "Is garbage collection overhead gone? case study of f2fs on zns ssds," in *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*, 2023, pp. 102–108.

[18] T. Kim, J. Jeon, N. Arora *et al.*, "Raizn: Redundant array of independent zoned namespaces," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 660–673.

[19] Y. Wang, Z. Sun, Y. Zhou *et al.*, "Balloon-zns: Constructing high-capacity and low-cost zns ssds with built-in compression," in *Design Automation Conference (DAC)*, 2024.

[20] Q. Wang, J. Li, P. P. Lee *et al.*, "Separating data via block invalidation time inference for write amplification reduction in {Log-Structured} storage," in *20th USENIX Conference on File and Storage Technologies (FAST 22)*, 2022, pp. 429–444.

[21] Micron Technical Note (No. TN-29-27): Design and Use Considerations for NAND Flash Memory.

[22] J. Werner, E. T. Cohen, and T. L. Canepa, "Read disturb handling for non-volatile solid state media," May 15 2014, uS Patent App. 13/729,966.

[23] Y. Seo, J. Yun, W. Lee *et al.*, "Memory controller, method of operating the same and memory system including the same," Dec. 13 2016, uS Patent 9,519,576.

[24] C. Liu, Y. Lee, M. Jung *et al.*, "Prolonging 3d NAND SSD lifetime via read latency relaxation," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021.
[25] Q. Li, H. Dang, Z. Wan *et al.*, "Midas touch: Invalid-data assisted reliability and performance boost for 3d high-density flash," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 657–670.
[26] Huaicheng Li, "Femu github repository," https://github.com/MoatSysLab/FEMU, 2024.
[27] Z. Cao, S. Dong, S. Vemuri *et al.*, "Characterizing, modeling, and benchmarking {RocksDB}{Key-Value} workloads at facebook," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*, 2020, pp. 209–223.
[28] L. Long, S. He, J. Shen *et al.*, "Wa-zone: Wear-aware zone management optimization for lsm-tree on ZNS ssds," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 21, no. 1, pp. 16:1–16:23, 2024.
[29] D. Huang, D. Feng, Q. Liu *et al.*, "Splitzns: Towards an efficient lsm-tree on zoned namespace ssds," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 3, pp. 1–26, 2023.
[30] Z. Tan, L. Long, J. Shen *et al.*, "Optimizing garbage collection for zns ssds via in-storage data migration and address remapping," *ACM Transactions on Architecture and Code Optimization*, 2024.
[31] Facebook, "Rocksdb github repository," https://github.com/facebook/rocksdb, 2024.
[32] G. Choi, K. Lee, M. Oh *et al.*, "A new {LSM-style} garbage collection scheme for {ZNS}{SSDs}," in *USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 20)*, 2020.
[33] C.-Y. Liu, Y.-M. Chang, and Y.-H. Chang, "Read leveling for flash storage systems," in *Proceedings of the 8th ACM International Systems and Storage Conference*, 2015, pp. 1–10.
[34] T.-C. Wu, Y.-P. Ma, and L.-P. Chang, "Flash read disturb management using adaptive cell bit-density with in-place reprogramming," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 325–330.
[35] J. Li, B. Huang, Z. Sha *et al.*, "Mitigating negative impacts of read disturb in ssds," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 1, pp. 1–24, 2020.
[36] M. Zhao, J. Li, Z. Cai *et al.*, "Block attribute-aware data reallocation to alleviate read disturb in ssds," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2021, pp. 1096–1099.
[37] G. Zhang, Y. Deng, Y. Zhou *et al.*, "Cocktail: Mixing data with different characteristics to reduce read reclaims for nand flash memory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2336–2349, 2022.

STREAM: Spatiotemporal Similarity-based Efficient Approximate Median with Tunable Granularity

Fenfang Li[1,2], Huizhang Luo[1], Weichen Liu[2], Anthony Theodore Chronopoulos[3], Kenli Li[1], Chubo Liu[1]
[1]Hunan University, China; [2]Nanyang Technological University, Singapore; [3]The University of Texas at San Antonio, USA

***Abstract*—The median (MED) is a crucial statistic for measuring the central tendency. However, exact MED computation remains costly, with even state-of-the-art (SOTA) algorithms failing to meet (near) real-time processing demands. While approximate MED algorithm has arisen as a promising candidate, existing approaches ignore the potential opportunity of spatiotemporal similarity within the application and fail to provide application-specific trade-offs between execution time and accuracy. Our goal is to design an enhanced approximate MED algorithm STREAM, which is capable of exploiting the spatiotemporal similarity to achieve bucket reuse and establish a tunable-grained bucket mechanism to meet the accuracy of application-specific requirements. Experimental results show that while maintaining nearly identical accuracy, STREAM outperforms the SOTA approximate methods DDSketch (up to 10×, 4.7× on average) and KLL (up to 71.2×, 10.1× on average).**

***Index Terms*—approximate computing, approximate median, spatiotemporal similarity, synopsis.**

I. Introduction

The identification of significant elements within large datasets is a fundamental challenge that drives numerous fields, such as embedded system libraries [1]–[4], computer vision [5], and data mining [6], [7]. Among them, median (MED) calculation emerges as a crucial special case, remaining resilient to extreme values and providing unique insights into the central tendency of the dataset.

Unfortunately, computing the exact MED over a set of n elements becomes prohibitive for embedded systems requiring real-time or near real-time responses. Compute-intensive applications, such as real-time monitoring, streaming data, and image processing, often operate under strict resource constraints in embedded systems. The high time complexity of exact MED algorithms, typically O(nlogn) for unsorted data, can lead to significant processing delays, impeding the response essential to these applications. Moreover, conventional exact MED algorithms depend on global or partial sorting steps, which can cause further latency. Therefore, a new design paradigm—*approximate computing* is considered a more feasible solution, which increases performance by controllably reducing accuracy [8]–[11].

Recently, popular approximate median (AM) algorithms for streaming data processing can be categorized into two classes: *bucket-based algorithms* and *compactor-based algorithms*. Bucket-based algorithms, such as DDSketch [12] and HDRHistogram [13], approximate the MED by allocating data into fixed-size buckets and maintaining a compact synopsis of the data distribution. Compactor-based algorithms, including KLL [14] and KLL± [15], rely on layered sampling and compressed representation to efficiently estimate the approximate MED. While effective in some applications, one significant limitation is that these algorithms ignore the spatiotemporal similarity (STS) commonly present in real-time applications [16], where consecutive data segments often exhibit similarity distribution. Specially, they treat each data segment as independent input and compute approximate MED completely from scratch without reusing information from the previous data segments. Thus, this leads to redundant computation in contexts where STS exists.

Another important limitation lies in the limited adaptability of existing algorithms for diverse application needs. The embedded system applications mentioned earlier are inherently error-tolerant, i.e., they generally require acceptable results rather than fully exact [11]. For example, a few pixels drop of processed image are imperceptible to human eyes. However, diverse applications exhibit distinct trade-offs between computational efficiency and accuracy. Limited research has explored tunable mechanisms to adjust approximation strategies based on application-specific requirements, leaving a gap in the current research field.

We propose a novel, simple yet effective algorithm, STREAM. It exploits the STS in application to improve performance. Unlike existing algorithms that process each data segment independently, STREAM allows subsequent data segments to reuse buckets of previous data segments. STREAM integrates bucketing and statistical metric computation of current data segment in a single pass, reducing execution time. Moreover, STREAM provides a tunable option, allowing tunable granularity in bucket strategies (coarse/fined-grained level), to achieve a trade-off between speed and accuracy for diverse applications. In particular, STREAM-CG (coarse-grained) constructs the buckets based on the 68-95-99.7 rule and allocates data into fixed-width buckets, achieving rapid, storage-light MED estimation suited for real-time processing. To enhance accuracy, STREAM-FG (fine-grained) employs a bucket tunning strategy that narrows the boundaries using a scale function. This solution places finer buckets around the region closer to the exact MED, in contrast to the fixed-width bucket of STREAM-CG, thereby achieving higher accuracy.

In summary, this paper makes the following contributions:

- We propose STREAM, a novel bucket-based framework that exploits the potential STS in applications. It computes current approximate MED by reusing buckets of previous data segments if STS exists between consecutive

Corresponding authors: luohuizhang@hnu.edu.cn, liuchubo@hnu.edu.cn

data segments.

- We propose a tunable mechanism to adjust bucket strategy for diverse applications. STREAM-CG provides a coarse-grained implementation, which meets the requirements of applications that tolerate some error with mild perceived effect to obtain significant speed improvement. STREAM-FG provides a fine-grained implementation, which refines the median estimate by concentrating the buckets in regions closer to the exact MED.
- Experimental results show that while maintaining nearly identical accuracy, STREAM outperforms the SOTA approximate methods DDSketch (up to 10x, 4.7x on average) and KLL (up to 71.2x, 10.1x on average).

II. Background and Motivation

A. Approximate Computing

Approximate computing (AxC) is an emerging paradigm for energy-efficient and/or high-performance design [11]. Companies, such as IBM, Google, Intel, and ARM, are actively engaged in pioneering research and developing commercial offerings that incorporate *AxC* strategies. Generally, *AxC* techniques are categorized into six classes: (i) *selective task skipping*, (ii) *approximate memoization*, (iii) *relaxed synchronization*, (iv) *precision scaling*, (v) *data sampling*, and (vi) *approximate programming languages*. *AxC* can provide effective trade-offs between execution time, accuracy, power, and so on, making it a powerful tool for applications with diverse performance needs.

- **Need for Low-Power/Energy Computing**. With transistor scaling reaching deep nanometer domains, power and energy consumption have become the primary constraints in modern computing systems. In this context, *AxC* can effectively balance power consumption with accuracy, area, and other performance requirements of applications.
- **Need for Accelerated Computing**. With the explosive growth in data volume, compute-intensive workloads of applications challenge their efficient implementation in terms of speed. *AxC* addresses this by enabling effective trade-offs between speed and desired accuracy.

Among various approximate computing techniques, approximate MED computation becomes a research hotspot, which can address the challenges faced by resource constraints and provide faster response times and comparable accuracy for compute-intensive applications [17]. However, existing approximate MED algorithms fail to use potential STS presenting in streaming data processing and lack the tunability to adapt to diverse application requirements.

B. Limitations of Approximate Median Calculation

Unexploited Spatiotemporal Similarity. Existing approximate MED algorithms fail to utilize the potential STS between consecutive data segments. To gain insight into STS, we observe the distribution within an application. As shown in Fig. 1, consecutive segments exhibit high similarity, as observed from their overlapping distribution curves. This overlap reflects comparable statistical properties, such as mean and standard deviation, and also validates the premise of reusing bucket partitions across segments (i.e., exploit the properties of $\mathbb{S}_i$ to optimize the computation of $\mathbb{S}_{i+1}$). To further investigate the statistical significance of differences between distributions, we also employ the Mann Whitney U (MWU)[1] test, a non-parametric statistical technique that evaluates the central tendencies of the two distributions (i.e., their medians). As shown in Fig. 2, all p-values of consecutive segments exceed the significance level of alpha 0.05, indicating consecutive segments show no statistically significant median differences. Thus, medians of consecutive segments fall into similar buckets, enabling reduced execution time by reusing buckets from previous data segments.

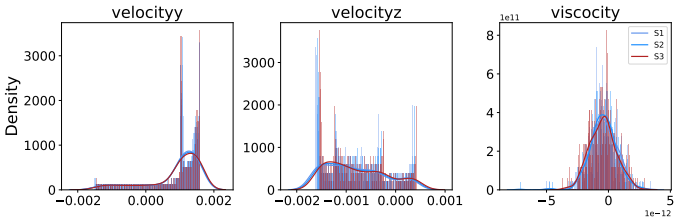


Fig. 1: Distribution of three consecutive data segments ($\mathbb{S}_1$, $\mathbb{S}_2$, $\mathbb{S}_3$) in application $Miranda$[2]. Overlaid on this scatter plot are density curves, fitted with kernel density estimation. $\mathbb{S}_i$ represents the i-th data segment.

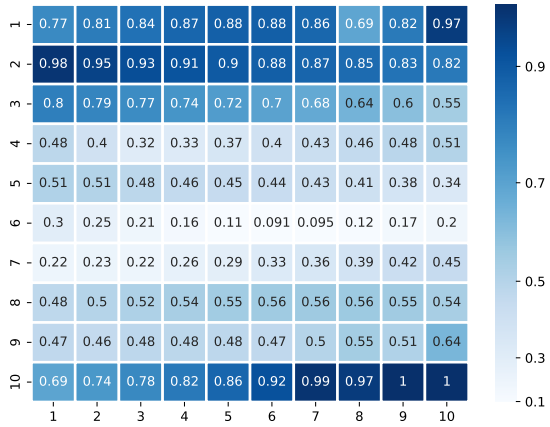


Fig. 2: Heatmap of Mann Whitney U test for consecutive streaming data segments ($\mathbb{S}_1$ to $\mathbb{S}_{101}$). Each cell represents the p-value ($\alpha = 0.05$) between consecutive segments ($\mathbb{S}_i$, $\mathbb{S}_{i+1}$). Darker color indicates higher values.

Limited Adaptability for Application-Specific Needs. Existing bucketing strategies generally adopt a one-size-fits-all approach, which limits their adaptability across various applications with varying requirements for speed and accuracy. Table I compares different applications, highlighting their trade-offs. Power monitoring and image processing often prioritize processing speed over exact accuracy, whereas healthy wearable devices and sensors require high accuracy to ensure the reliability of measurements. For applications with high accuracy requirements, only coarse-grained bucket strategies

[1]https://en.wikipedia.org/wiki/Mann-Whitney_U_test
[2]https://sdrbench.github.io/

TABLE I: Requirements of accuracy and speed for different applications [11].

Applications	Accuracy	Speed
Environmental monitoring	Medium	Medium
Power monitoring	Midum	High
Image processing	Medium to High	High
Health wearable devices	High	Medium to High
Disaster warning sensor	High	High

may lead to significant errors due to unreasonable boundaries. On the contrary, for applications with low accuracy requirements, only fine-grained strategies can result in a waste of computational resources to obtain a fined boundary. Thus, a tunable strategy can more effectively balance speed and accuracy, enhancing adaptability for application-specific.

III. Design and Implementation

A. STREAM Framework

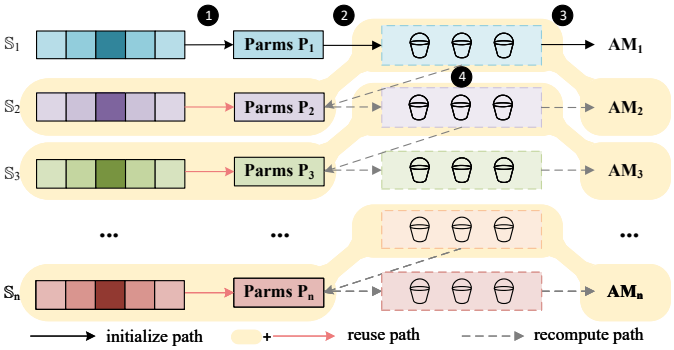


Fig. 3: The framework using STS to get approximate MED, where initialize path follows steps ❶+❷+❸, reuse path follows steps ❷ (❶)+ ❸, and recompute path follows steps ❹+❸.

Fig. 3 illustrates the framework of STREAM that calculates the approximate MED with STS. For the initial streaming data segment $\mathbb{S}_1$, STREAM follows the initialize path: ❶ traversing the segment to determine the bucket partitioning parameters P_1, ❷ re-traversing and allocating data into buckets based on P_1, ❸ finding the bucket holding MED and identifying the approximate MED AM_1 using the bucket's boundaries. For the subsequent streaming data segment $\mathbb{S}_2$, STREAM utilizes the STS to reuse the bucket: STREAM first traverses $\mathbb{S}_2$, while simultaneously performing two operations: allocating data into buckets based on the bucket partitions from $\mathbb{S}_1$ and computing the current bucket parameters P_2 (Step ❷ (❶)). Then, two scenarios may occur: 1) If STS is confirmed, STREAM directly identifies the MED bucket and computes AM_2 using its boundaries (Step ❸). 2) If similarity validation fails, STREAM executes recompute step ❹, which requires bucketing with P_2 before obtaining the approximate MED AM_2 (Step ❸). The same procedure is iteratively applied to subsequent segments, starting with $\mathbb{S}_3$, following the reuse or recompute path based on its STS with the previous segment.

B. Tunable-Grained Bucket Strategy

*1) **Overview**:* Fig. 4 illustrates the overview of a mechanism, which is designed with tunable granularity to meet varying demands for accuracy and speed in different applications. This mechanism attempts to compute the approximate MED with the desired accuracy ϵ for a specific application. For a given application $\mathbb{S}$, we first execute the coarse-grained method, STREAM-CG. It returns an initial approximate MED, AM_1, along with its achieved accuracy ϵ'. Based on the relationship between ϵ' and the desired accuracy ϵ, the following cases are considered:

- Case 1: If $\epsilon' \geq \epsilon$, AM_1 is directly used as final result, i.e., branch (a) in Fig. 4, ensuring both accuracy and speed.
- Case 2: If $\epsilon' < \epsilon$, it executes STREAM-FG, which narrows bucket boundaries to enhance accuracy and obtain a new approximate MED, AM_2, i.e., branch (b) in Fig. 4. While STREAM-FG improves accuracy, it incurs additional computational cost.
- Case 3: If $\epsilon' = 1$, typically observed in highly concentrated data, the MED can be efficiently approximated by the mean (μ). In the extreme case where all data are identical, AM equals μ. Thus, μ can be directly used as the approximate MED, i.e., branch (c) in Fig. 4, bypassing further computation.

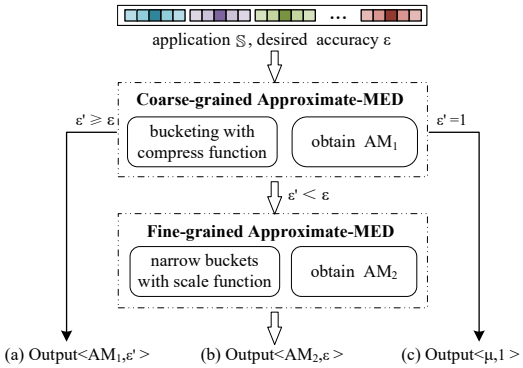


Fig. 4: The tunable mechanism for approximating MED with the desired accuracy ϵ, where ϵ' represents the achieved accuracy of AM_1 at the coarse-grained level.

*2) **Coarse-grained Approximate MED**:* We implement STREAM at the coarse-grained level (STREAM-CG), which involves computing bucket partitioning parameters, bucketing based on these parameters, and estimating AM using the bucket's boundaries. To efficiently represent data distribution, we adopt a synopsis structure $\mathbb{B} = \{B_{\frac{m-1}{2}-}, ..., B_{1-}, B_{0-}, B_{0+}, B_{1+}, ..., B_{\frac{m-1}{2}+}\}$, where m is the size of the synopsis, and each bucket B_i counts the number of data that fall within a specific boundary.

To determine bucket boundaries, we compute the mean (μ) and standard deviations (σ) of data, which is critical for effective bucket partitioning. The choice of μ and σ as partitioning parameters is guided by the 68-95-99.7 rule [18], which ensures $\mu \pm 3\sigma$ covers nearly all observed data. Moreover, in applications with STS, the consistency of data distributions between consecutive data segments results in stable μ and σ, making them ideal for defining bucket boundaries.

With μ and σ as inputs, we define bucket boundaries using the compress function $\gamma(x) := \frac{x-\mu}{\sigma}$, which allocates each

data to fixed-width buckets based on its deviation from μ. This method allows a constant synopsis size. For data with $\gamma(x)$ value outside the bucket range, we assign them to the leftmost or rightmost bucket, ensuring all data are included.

Finally, given that we keep the count of data within each bucket in synopsis $\mathbb{B}$, STREAM approximates the MED by finding the first bucket in $\mathbb{B}$, where the cumulative counts exceed 50% of the data. Within this target bucket, we calculate approximate MED by an interpolation formula [19]

$$AM = L + \frac{\ell/2 - \sum freq}{freq_{med}} width, \tag{1}$$

where L is the lower boundary of the bucket containing MED, ℓ is the total number of data in $\mathbb{S}_i$, $\sum freq$ is the cumulative count of data in all buckets below the MED bucket, $freq_{med}$ is the count in the MED bucket, and $width$ is the width of the MED bucket (e.g., σ in STREAM-CG).

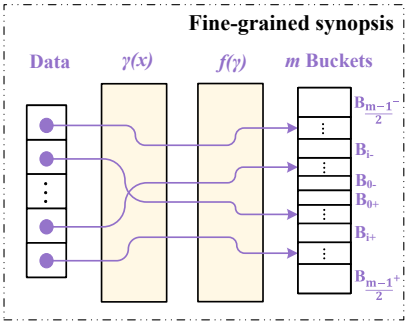


Fig. 5: The structure of STREAM-FG.

*3) **Fine-grained Approximate MED**:* While STREAM-CG efficiently computes the approximate MED with an achieved accuracy ϵ', it may not meet the desired accuracy ϵ for certain applications. To address this limitation, STREAM-FG introduces a fine-grained bucket strategy to improve accuracy. A key feature of STREAM-FG is its use of the compression parameter γ, derived from the bucket partitioning parameters of STREAM-CG. The primary difference between STREAM-CG and STREAM-FG lies in how bucket boundaries are defined: STREAM-CG defines fixed-width bucket boundaries based on μ and σ, whereas STREAM-FG employs γ to refine bucket widths near the middle of the synopsis. Specifically, with γ as input, the bucket boundaries are determined by scale function $f(\gamma) = \omega\pi tan^{-1}(2\gamma) + \delta$, where ω is the expansion coefficient, and δ is set to ensure the scale function can achieve the minimum and maximum values. This results in an uneven-width bucket structure, as shown in Fig. 5. Once the bucket partitioning parameter is determined, STREAM-FG follows the same steps as STREAM-CG for bucket allocation and MED calculation. The fine-grained bucket strategy of STREAM-FG significantly improves the accuracy of the approximate MED, making it well-suited to applications with high accuracy requirements.

*4) **Theoretical Proof**:* We conduct a theoretical proof on STREAM-CG, which proves our STREAM-CG can always work in initialize path and the rationality of STREAM-CG in reuse path.

Initialize Path. A bound of median location is measured by mean and standard deviation in initialize path, where the μ and σ are calculated on the samples where the median is estimated.

Proposition 1. *Let X be a random variable on $\mathbb{R}$, θ be the corresponding distribution (an array of real values is a special case), let $\mu = \mathbb{E}[X]$ be the mean, and $\sigma = \sqrt{\mathbb{E}[(X-\mu)^2]}$ be the standard deviation, then the median m of X satisfies $|m - \mu| \leq \sigma$.*

Proof. Let θ be the distribution of x in $\mathbb{R}$. We first show $\mathbb{E}|x-m| \leq \mathbb{E}|x-c|$ for any c. Without loss of generality, we assume $c > m$, then $\mathbb{E}|x-m| - \mathbb{E}|x-c| = \int_{-\infty}^{m}(m-x)-(c-x)\mathrm{d}\theta(x) + \int_{m}^{c}(x-m)-(c-x)\mathrm{d}\theta(x) + \int_{c}^{\infty}(x-m)-(x-c)\mathrm{d}\theta(x) = (m-c)(\theta((-\infty,m]) - \theta([m,c]) - \theta([c,\infty))) - 2\int_{m}^{c}(c-x)\mathrm{d}\theta(x) \leq 0$. Further, by Jensen's inequality, $|\mu - m| = \mathbb{E}[|X-m|] \leq \mathbb{E}[|X-\mu|] \leq \sqrt{\mathbb{E}[(X-\mu)^2]} = \sigma$. □

The previous proposition guarantees that our method always works in initialize path.

Reuse Path. Consider two random variables X_1 and X_2, representing the distribution of values of two consecutive data segments in reuse path, and producing μ_1, σ_1, μ_2, σ_2 accordingly. The goal is to locate the median of X_2 among intervals generated by μ_1, σ_1 of X_1. From the analysis of initialize path, it is known that $|m - \mu_2| < \sigma$. In the construction of the STREAM-CG algorithm, the threshold of path transfer is set to $3\sigma_1$.

Proposition 2. *If $|\mu_1 - \mu_2| + \sigma_2 < 3\sigma_1$ holds, then the STREAM-CG stays in reuse path in estimating the current median m_2.*

Proof. From Proposition 1, we see that $|m_2 - \mu_2| < \sigma$. By the triangle inequality, $|m_2 - \mu_1| = |\mu_1 - \mu_2| + |m_2 - \mu_2| < |\mu_1 - \mu_2| + \sigma_2 < 3\sigma_1$ if the condition in this proposition holds. Then the STREAM stays in the reuse path for sure. □

As a remark, Proposition 2 is a sufficient condition, meaning there could be cases not satisfying the inequality, but the reuse path still works.

The sufficient condition further transforms into $|\mu_1 - \mu_2| + (\sigma_2 - \sigma_1) < 2\sigma_1$, suggesting that when μ, σ of two consecutive data segments does not vary too much, and σ_1 is not too small to provide a large enough bar range, the STREAM stays in reuse path.

IV. Evaluation and Analysis

A. Experimental Setup

*1) **Platform and Setting**:* All experiments are conducted on a server with 5.2GHz Intel(R) Core (TM) i9-12900K processors, and 32 GB of 5200MHz DDR5 RAM at OS kernel of Linux 5.15. We implement all codes with C and build them with gcc 7.5.0 (Ubuntu 7.5.0-6ubuntu2) and -O3 option. By default, the parameters of the comparing algorithms are set according to the recommendation of their authors.

TABLE II: Dataset description.

Dataset	Description	Dim	Type
Miranda	Turbulence modeling	$256 \times 384 \times 384$	double
Hurricane	Weather modeling	$500 \times 500 \times 100$	float
NYX	Cosmological simulation	$512 \times 512 \times 512$	float
power	Power measurements	1041120	float
Moving MNIST	Video	64×64	integer
Lena	Image	512×512	integer
zipf	Synthetic	10000	integer
norm	Synthetic	10000	float
constant	Synthetic	10000	integer

2) ***Baselines****:* To evaluate the effectiveness of our algorithms, we compare them against the existing exact method, EXACT-MED (based on QUICKSORT [20]), and SOTA approximate algorithms: C++ implementation of DDSketch[3] and KLL [14]. We do not compare against HDRHistogrsm [13], as its C implementation is only applicable to positive integer data.

3) ***Datasets****:* Table II summarizes the 9 applications in our comparison experiments. The first six are real-world applications. Specifically, the $Miranda$, $Hurricane$, and NYX are produced by scientific applications from the Scientific Data Reduction Benchmarks suite[1]. The $power$, used in [12], consists of Global Active Power measurements from the UCI Individual household electric power consumption dataset. The $MovingMNIST$ and $Lena$ are significant applications for computer version from Kaggle public dataset. The last three applications are synthetic, $zipf$ consists of data generated from a zipfian distribution $zipf(1.2, 10000)$, $norm$ consists of data generated from a normal distribution $\mathcal{N}(0, 1)$, and $constant$ consists of data with the same constant value 1.

B. Comparison to Baselines

The time cost and accuracy results are summarized in Table III, with time measured in milliseconds. The accuracy rate of approximate MED relative to exact MED indicates that the relative error of approximate MED, i.e. $\frac{|M\hat{E}D - MED|}{|max(x) - min(x)|}, x \in \mathbb{S}_i$, can be less than 5%.

1) ***Time Cost****:* We experiment ten times on each application and use the average run time as the result. As shown in Table III, STREAM-CG and STREAM-FG achieve significant speed improvements. Specifically, STREAM-CG outperforms EXACT-MED, DDSketch, and KLL up to 40.2x, 10.0x, and 71.2x (17.6x, 7.8x, and 15.8x on average), respectively. STREAM-FG outperforms EXACT-MED, DDSketch, and KLL up to 19.8x, 10.0x, and 71.2x (4.7x, 2.2x, and 7.0x on average), respectively.

STREAM-FG exhibits approximately 4-7x higher time costs than STREAM-CG across all applications (except $constant$). The result is expected, as STREAM-CG needs only a single bucketing calculation, while STREAM-FG requires the fine-grained buckets strategy through the scale function succeeding the compress function. The time costs of STREAM-CG and STREAM-FG are found to be comparable for $constant$. Since they directly return mu as their MED, bypassing further bucketing calculations.

[3]https://github.com/tudor1805/sketches-cpp/

TABLE III: Comparison to different baselines over various applications, where n is the size of the application and "Accu Rate" denotes the accuracy rate of approximate MED relative to exact MED. The parameters of STREAM-FG are set as ω = 5, δ = 23, resulting in the size of synopsis m = 47.

	velocityy (n=37.7M)		*velocityz (n=37.7M)*		*viscocity (n=37.7M)*	
Algorithm	Accu Rate	Time	Accu Rate	Time	Accu Rate	Time
EXACT-MED	/	613.98	/	630.94	/	1222.37
DDSketch	100%	504.75	100%	512.63	100%	577.20
KLL	99.99%	560.83	99.99%	593.73	99.99%	821.03
STREAM-CG (our)	92.78%	**55.01**	92.38%	**62.74**	89.95%	**58.05**
STREAM-FG (our)	**100%**	246.23	**100%**	248.30	**100%**	251.14
	TC (n=25M)		*U (n=25M)*		*PRECIP.lg (n=25M)*	
Algorithm	Accu Rate	Time	Accu Rate	Time	Accu Rate	Time
EXACT-MED	/	260.93	/	307.50	/	320.79
DDSketch	31.72%	266.79	99.99%	167.15	86.60%	268.58
KLL	99.98%	448.02	99.99%	407.67	**99.99%**	467.07
STREAM-CG (our)	91.10%	**33.04**	91.39%	**33.82**	94.02%	**35.44**
STREAM-FG (our)	**99.98%**	167.15	**99.99%**	165.56	99.94%	158.44
	temperature (n=134M)		*velocity_x (n=134M)*		*velocity_y (n=134M)*	
Algorithm	Accu Rate	Time	Accu Rate	Time	Accu Rate	Time
EXACT-MED	/	3624.06	/	2500.35	/	2581.12
DDSketch	100%	1495.17	94.94%	1722.38	98.44%	1731.18
KLL	99.99%	2594.92	99.99%	1971.93	99.99%	2019.34
STREAM-CG (our)	99.91%	**179.48**	89.37%	**183.13**	89.32%	**179.70**
STREAM-FG (our)	**100%**	935.72	**100%**	881.51	**100%**	877.19
	power (n=1041K)		*MovingMNIST (n=7.2M)*		*lena (n=26.2M)*	
Algorithm	Accu Rate	Time	Accu Rate	Time	Accu Rate	Time
EXACT-MED	/	24.80	/	122.09	/	1026.41
DDSketch	**95%**	13.03	100%	14.29	100%	263.46
KLL	49.85%	23.30	99.95%	41.55	50%	480.05
STREAM-CG (our)	62.74%	**1.54**	100%	**12.82**	100%	**34.50**
STREAM-FG (our)	86.95%	7.56	**100%**	48.43	**100%**	186.38
	zipf (n=1M)		*norm (n=10M)*		*constant (n=10K)*	
Algorithm	Accu Rate	Time	Accu Rate	Time	Accu Rate	Time
EXACT-MED	/	391.58	/	821.97	/	0.99
DDSketch	100%	99.91	100%	134.08	100%	0.50
KLL	90.91%	193.17	100%	188.03	100%	3.56
STREAM-CG (our)	**100%**	**12.28**	**100%**	**20.44**	**100%**	**0.05**
STREAM-FG (our)	**100%**	85.06	**100%**	81.30	**100%**	**0.05**

For TC, STREAM-CG achieves optimal time and high accuracy, since it requires only single-pass scan and compress function γ, which ensures relatively even data allocation across buckets for this application. Moreover, our STREAM-FG further improves the accuracy. For $temperature$, STREAM-CG achieves superior performance in both speed and accuracy, while STREAM-FG exhibits poor time efficiency. If $\epsilon' \geq \epsilon$, STREAM-CG directly returns AM_1, making further tuning with STREAM-FG unnecessary.

2) ***Accuracy of Approximation****:* As shown in Table III, STREAM-FG achieves high accuracy across applications while minimizing time costs. Notably, for the highly skewed application $zipf$, STREAM-FG can achieve an accuracy rate of 100%. For the extreme case $constant$, STREAM-FG returns μ as its MED directly, which is exactly the ground truth.

The accuracy rate of STREAM-CG is slightly lower for two main reasons: 1) While STS exists between data segments, the degree of data dispersion for each segment is relatively large (e.g., in $power$). This results in a significant deviation of the μ and σ from the MED, which in turn impacts the accuracy of approximate MED based bucket boundaries. 2) In some applications, data is highly concentrated with a few extreme values (e.g., U exhibits high frequency near zero alongside the distribution of non-near-zero values). This results in most data falling into the bucket closest to the mean, limiting the effectiveness of the compress function. This is also the primary reason why we adopt the fine-grained bucket strategy.

C. Evaluation of Proposed Techniques

1) ***Evaluating the Validity****:* The validity refers to the effectiveness of our approximate MED methods to exploit

TABLE IV: Hit rate of bucket reuse based STS.

	velocityy	*velocityz*	*viscocity*	*TC*	*U*
STREAM-CG	100%	99.96%	99.99%	99.90%	99.92%
STREAM-FG	100%	99.98%	100%	99.94%	91.99%
	PRECIP.lg	*temperature*	*velocity_x*	*velocity_y*	*power*
STREAM-CG	99.76%	100%	100%	100%	81.67%
STREAM-FG	96.81%	100%	100%	100%	96.58%
	MovingMNIST	*lena*	*zipf*	*norm*	*constant*
STREAM-CG	100%	100%	100%	100%	NA
STREAM-FG	100%	100%	100%	100%	NA

the STS of the application. The validity is assessed using the $hitrate$, which reflects the proportion of correctly reused buckets between consecutive data segments. As shown in Table IV, $hitrate$ exceeds 99% for all applications, except $power$ using STREAM-CG and $constant$. For $power$, which represents power measurements recorded over a three-hour period. Due to daily fluctuations in energy consumption, the bucket widths in the STREAM-CG are not narrow enough to precisely capture slight deviations between consecutive segments. For $constant$, where the standard deviation is zero, no bucketing calculation is performed, and the mean is directly returned as the MED. This special case is not included in the standard statistical process. Thus, $hitrate$ is marked as NA.

*2) **Varying the Expansion Coefficient** ω:* The expansion coefficient ω specifies the trade-off between the effectiveness and efficiency of STREAM-FG. We vary ω and apply STREAM-FG to each application, as shown in Fig. 6.

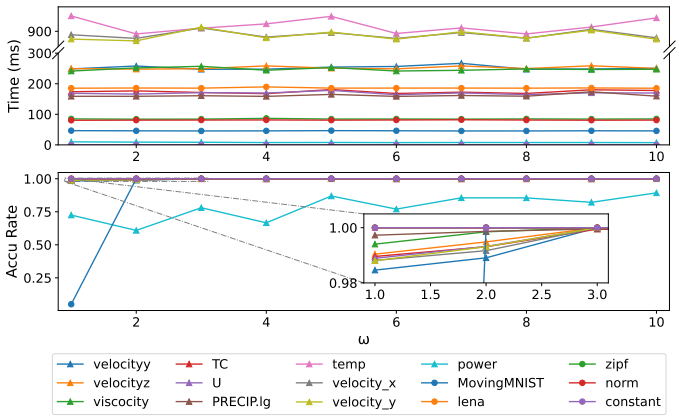


Fig. 6: Varying the expansion coefficient ω.

For higher concentrated applications, their σ are sufficiently small to return the MED directly, resulting in a near-constant and maximal precision. For other cases, sharp upward/downward trends are observed in Fig. 6. It is because when the dispersion of the data is large, the upper and lower bounds of the bucket vary due to changes in ω, which in turn fluctuates the accuracy. Moreover, while increasing ω to enhance accuracy, it simultaneously increases the number of buckets m (i.e., space cost) with a relatively stable time cost. Thereby, for concentrated applications, setting ω to an appropriate value can achieve better accuracy without significantly increasing the space cost.

V. Conclusion

In this paper, we propose STREAM, utilizing STS between data segments to reduce execution time and proving a tunable granularity mechanism to meet the accuracy of application-specific requirements. Our theoretical analysis for correctness and efficiency proves the soundness of the proposed algorithms. The experimental results demonstrate the superiority of our algorithms, i.e., while maintaining nearly identical accuracy, STREAM outperforms the SOTA approximate methods DDSketch (up to 10×, 4.7× on average) and KLL (up to 71.2×, 10.1× on average).

Acknowledgment

This work was partially supported by the National Key R&D Program of China under Grant 2023YFB3001705, the National Natural Science Foundation of China (Grant No. 62321003 (Creative Research Groups), 62472160, and U21A20461), Hunan Province Graduate Research Innovation Project (CX20240425), and by the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG94/23).

References

[1] John Wellbelove, "Embeded Template Library," https://www.etlcpp.com/, 2024.

[2] "Software-Defined Radio Digital Signal Processing Library," https://github.com/jgaeddert/liquid-dsp?tab=readme-ov-file, 2024.

[3] "FPM: A C++ header-only fixed-point math library," https://github.com/MikeLankamp/fpm, 2024.

[4] "Eigen: a C++ template library for linear algebra," https://eigen.tuxfamily.org/index.php, 2024.

[5] M. Liu, Z. Zhou, P. Shang, and D. Xu, "Fuzzified image enhancement for deep learning in iris recognition," *IEEE TFS*, vol. 28, no. 1, pp. 92–99, 2019.

[6] S. Song, F. Gao, A. Zhang, J. Wang, and P. S. Yu, "Stream data cleaning under speed and acceleration constraints," *ACM TODS*, vol. 46, no. 3, pp. 1–44, 2021.

[7] M. M. A. Patwary, S. Byna, N. R. Satish, N. Sundaram, Z. Lukić, V. Roytershteyn, M. J. Anderson, Y. Yao, Prabhat, and P. Dubey, "Bd-cats: big data clustering at trillion particle scale," in *SC*, 2015, pp. 1–12.

[8] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *ETS*. IEEE, 2013, pp. 1–6.

[9] K. Parasyris, G. Georgakoudis, H. Menon, J. Diffenderfer, I. Laguna, D. Osei-Kuffuor, and M. Schordan, "Hpac: evaluating approximate computing techniques on hpc openmp applications," in *SC*, 2021, pp. 1–14.

[10] V. Gupta, T. Li, and P. Gupta, "Lac: Learned approximate computing," in *DATE*. IEEE, 2022, pp. 1169–1172.

[11] V. Leon, M. A. Hanif, G. Armeniakos, X. Jiao, M. Shafique, K. Pekmestzi, and D. Soudris, "Approximate computing survey, part i: terminology and software & hardware approximation techniques," *arXiv preprint arXiv:2307.11124*, 2023.

[12] C. Masson, J. E. Rim, and H. K. Lee, "Ddsketch: A fast and fully-mergeable quantile sketch with relative-error guarantees," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 2195–2205, 2019.

[13] Gil Tene, "HdrHistogram: A High Dynamic Range (HDR) Histogram," https://github.com/HdrHistogram/HdrHistogram, 2024.

[14] "Apache DataSketches Core C++ Library," https://github.com/apache/datasketches-cpp, 2024.

[15] F. Zhao, S. Maiyya, R. Wiener, D. Agrawal, and A. E. Abbadi, "Kll±approximate quantile sketches over dynamic datasets," *Proceedings of the VLDB Endowment*, vol. 14, no. 7, pp. 1215–1227, 2021.

[16] Y. Zhang, X. Gao, Z. Chen, H. Zhong, H. Xie, and C. Yan, "Mining spatial-temporal similarity for visual tracking," *IEEE TIP*, vol. 29, pp. 8107–8119, 2020.

[17] A. McGregor and P. Valiant, "The shifting sands algorithm," in *SODA*. SIAM, 2012, pp. 453–458.

[18] V. Czitrom and P. D. Spagon, *Statistical case studies for industrial process improvement*. SIAM, 1997.

[19] J. Han, J. Pei, and H. Tong, *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[20] C. A. R. Hoare, "Algorithm 64: quicksort," *Communications of the ACM*, vol. 4, no. 7, p. 321, 1961.

MIRACLE: Multimodal Information Retrieval via a Combined In-Memory Processing and Content Addressable Memory Approach

Xuehui Liu*, Xueyan Wang*†, Tianyang Yu‡, Chen Cheng*, Shuo Ran‡,
Bi Wu‡, Xiaotao Jia*†, Weiqiang Liu‡, Gang Qu§, Weisheng Zhao*
*School of Integrated Circuit Science and Engineering, Beihang University, Beijing, China
‡College of Integrated Circuits, Nanjing University of Aeronautics and Astronautics, Nanjing, China
§Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA

***Abstract*—The rapid advancement of information technology has brought multimodal information retrieval into the research spotlight. Neural networks, particularly Transformers, have emerged as the dominant solution for extracting multimodal feature vectors. While neural network acceleration has been extensively explored, the subsequent retrieval stage in multimodal scenarios remains under-optimized. Conventional retrieval approaches, such as cosine similarity sorting on von Neumann architectures, suffer from significant data migration and computational inefficiencies. Hashing methods enhance storage and computation efficiency but encounter challenges in energy-efficient implementation and mitigating accuracy losses due to modal heterogeneity. This paper presents a hybrid architecture that integrates in-memory processing (PIM) and content-addressable memory (CAM) to address these challenges. Transformer-extracted features are processed via in-memory random hashing leveraging device-intrinsic properties, with CAM facilitating parallel search space reduction. A final cosine similarity reranking stage refines the results while balancing accuracy with energy efficiency. Experimental evaluations validate that the proposed method, when compared to the baseline traditional CPU-based cosine similarity retrieval, 1) achieves almost identical level of accuracy, dramatically outperforming other pure CAM-based Hamming distance retrieval approaches; and 2) reduces latency by $9.45\times$ and energy consumption by $30.20\times$.**

***Index Terms*—multimodal information retrieval, hashing, in-memory processing, content-addressable memory**

I. Introduction

The rapid expansion of Internet usage has led to data being represented in diverse forms, collectively known as multimodal data. Efficient retrieval of relevant information from such heterogeneous and vast datasets has become increasingly challenging, necessitating the development of multimodal information retrieval systems capable of delivering comprehensive, multidimensional search results [1].

In multimodal information retrieval, the process begins with feature extraction, where neural networks are used to derive critical feature vectors from diverse data modalities. Transformers [2], with their powerful attention mechanisms and cross-modal adaptability, have emerged as one of the leading architectures for this purpose. The multimodal information retrieval process, typically modeled as an Approximate Nearest Neighbor (ANN) problem, involves identifying entries across modalities with high similarity to the query item. However, traditional methods, such as cosine similarity sorting, have significant energy and latency issues due to frequent data migrations between memory and processors, exacerbated by the von Neumann bottleneck [3].

Hashing methods address some of these challenges by mapping high-dimensional feature vectors into compact hash sequences while preserving similarity relationships [4]. The retrieval workflow in hash-based systems involves three phases: feature extraction, hash mapping, and feature retrieval. During the hash mapping phase, methods like locality-sensitive hashing (LSH) generate hash codes based on matrix-vector multiplications (MVM) with hyperplane normal vectors. However, these operations are computationally intensive, particularly in generating random normal vectors and performing the required MVM calculations. Secondly, hash-based retrieval relies on frequent data transfers for similarity computations and sorting. Thirdly, dimension reduction during hashing often degrades retrieval accuracy in multimodal scenarios.

Emerging technologies such as in-memory processing (PIM) [5]–[7] and content-addressable memory (CAM) [8] offer promising solutions to mitigate the von Neumann bottleneck by enabling in-memory operations for hash mapping and retrieval. Among different PIM technologies, spin-transfer-torque magnetic random access memory (STT-MRAM) stands out as an ideal platform due to its non-volatility, high endurance, and superior storage density [9], [10]. It can efficiently support energy-efficient matrix multiplications and random vector generation through its intrinsic resistive device properties.

CAM, on the other hand, facilitates high-speed parallel matching operations and hence is potentially well-suited for frequent retrieval tasks in multimodal systems. However, recent advancements utilizing CAM for hash sequence retrieval are limited to single-modal in areas such as image retrieval [11] and memory-augmented neural networks [3], [12]. For multi-modal contexts, shorter hash sequences compromise retrieval accuracy, while longer sequences exceed the matching capabilities of CAM.

In this paper, we propose MIRACLE: a Multimodal Information Retrieval Framework via a Combined In-Memory Processing and Content Addressable Memory Approach. MIRACLE integrates PIM's in-situ computation capabilities with

†Corresponding authors are Xueyan Wang and Xiaotao Jia: wangxueyan@buaa.edu.cn, jiaxt@buaa.edu.cn.

BUAA's work was supported in part by the National Natural Science Foundation of China (NO.62474016 and NO.92364201), the Special Fund for Basic Research Funds of Central Universities, and the Young Elite Scientists Sponsorship Program by CAST (2023QNRC001). NUAA's work was supported in part by the National Natural Science Foundation of China (NO.92364201).

CAM's parallel search characteristics, enhanced by a refined cosine similarity ranking stage where we use CAM as a filter to narrow down the search space before performing high-precise cosine similarity reranking. In addition to the novel framework of MIRACLE, we make the following contributions:

- We develop an energy-efficient hashing method to leverage STT-MRAM's intrinsic device properties for random vector generation and efficient matrix multiplications.
- We design a threshold-based filtering strategy that exploits CAM's parallel search capabilities to reduce the retrieval search space, improving both accuracy and energy efficiency.
- We implement MIRACLE and show that it maintains the baseline approach's retrieval accuracy while drastically reducing latency and energy consumption by $9.45\times$ and $30.2\times$, respectively.

The remainder of this paper is organized as follows: Section II presents the background knowledge on multimodal feature extraction, hashing method for the ANN problem, and PIM and CAM based on STT-MRAM. Section III describes the proposed hybrid architecture for multimodal information retrieval. Sections IV and V elaborate the implementation of locality-sensitive hash mapping in PIM and a CAM-based search space filtering, respectively. Section VI reports experimental results and Section VII concludes.

II. Preliminaries

A. Multimodal Feature Extraction Network

Different data modalities require unique feature extraction networks, with the Transformer [2] being prominent due to its effective attention mechanism, flexible tokenization, and universal architecture. It has been successfully applied in text, image, video and audio processing [13], [14], showcasing strong scalability in multimodal contexts. CLIP [15] represents a significant advancement in this area, utilizing two Transformer encoders—one for text and one for images. Through contrastive learning, CLIP establishes a shared representation space that maximizes cosine similarity between matched image-text pairs, thereby enhancing multimodal retrieval capabilities. In this work, we utilize CLIP for feature vector extraction from both image and text modalities.

B. Hashing Method

Hashing methods provide efficient solutions for approximate nearest neighbor search problems. These methods are categorized into data-independent locality-sensitive hashing (LSH) and data-dependent learning to hash approaches [4]. LSH has demonstrated widespread success in both academic research and industrial applications, from computer vision tasks to Google's sim-hash implementation for duplicate webpage detection.

For cosine similarity preservation, sign random projection hashing serves as the corresponding locality-sensitive hashing technique, which partitions the feature space using random hyperplanes. The LSH-generated hash sequence h for a high-dimensional vector x can be expressed as:

$$h = sgn(x \cdot N) \tag{1}$$

where sgn represents the sign function and N denotes a matrix of random zero-mean vectors. Ternary LSH (TLSH) addresses the boundary problem where similar vectors near hyperplanes may receive different hash codes by introducing a third state X, which maintains zero Hamming distance to both 0 and 1. Section IV details the implementation of LSH within PIM crossbars.

C. PIM and CAM based on STT-MRAM

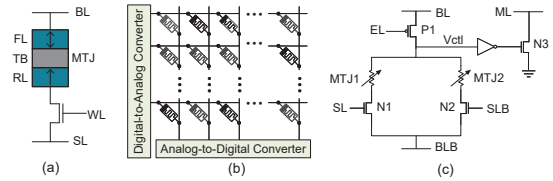


Fig. 1: PIM and CAM based on STT-MRAM: (a) STT-MRAM bitcell, (b) analog crossbar and (c) CAM cell.

Spin-transfer-torque magnetic random access memory (STT-MRAM) represents an advanced non-volatile memory technology utilizing Magnetic Tunnel Junctions (MTJs) for data storage. As illustrated in Fig. 1 (a), each MTJ consists of reference and free ferromagnetic layers, with their relative resistance indicating the logic state. STT-MRAM functions by passing a spin-polarized current to alter the magnetic direction of the free layer, enabling smaller dimensions and enhanced immunity to stray magnetic fields. Read and write operations are conducted by applying voltage across selected SLs and BLs.

Analog in-memory processing exploits resistive memory devices' inherent characteristics for computational operations, particularly matrix-vector multiplications (MVMs). As shown in Fig. 1 (b), matrix elements are encoded as conductances within the array, with input vectors applied as row voltages. Column currents represent MVM results according to Kirchhoff's law, supported by peripheral Digital-to-Analog Converters (DACs) and Analog-to-Digital Converters (ADCs).

CAM facilitates in-memory parallel retrieval of hash sequences with O(1) time complexity, offering dual advantages of accelerated retrieval and elimination of processor-memory data transfer overhead. The structure of the CAM cell based on STT-MRAM is illustrated in Fig. 1 (c). A pair of MTJs encodes the stored values of 0/1 or X through their resistance, while a pair of input voltages (SL/SLB) controls which branch is connected to ground. The left-side PMOS and MTJ create a voltage divider affecting V_{ctl}, which controls the right-side NMOS switching. This mechanism produces distinct match line (ML) discharge patterns for matched versus unmatched entries. Multiple cells can be connected in parallel for word-level matching operations. Threshold filtering implements Hamming distance-based entry selection through comparator-based discharge speed monitoring at the match line terminus.

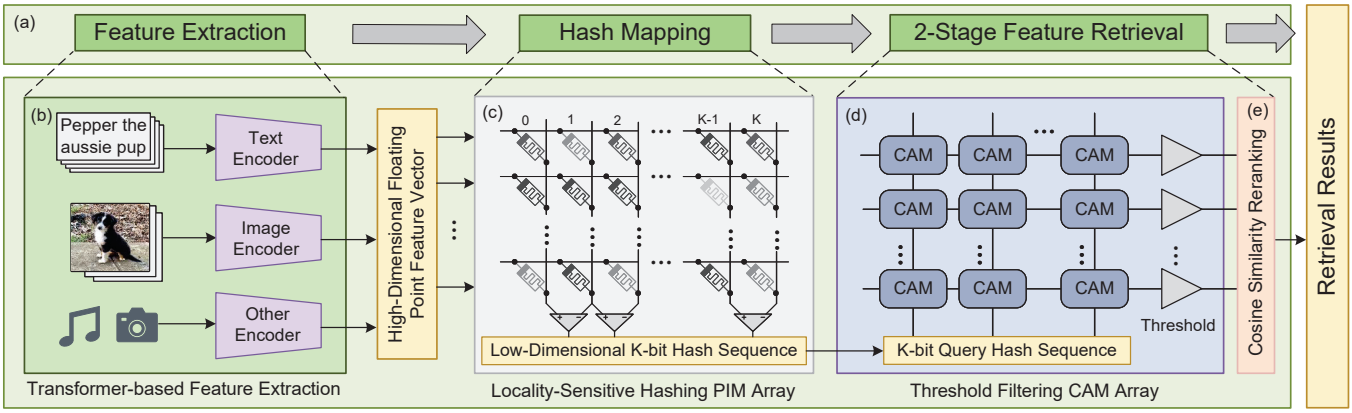


Fig. 2: Overall architecture: (a) workflow of multimodal information retrieval, (b) feature extraction using CLIP, (c) locality-sensitive hashing with analog crossbar, (d) filtering with CAM, and (e) cosine similarity reranking on subset filtered by CAM.

III. Overall Architecture

Hashing-based multimodal information retrieval typically involves three key phases: feature extraction, hash mapping, and feature retrieval.

The first phase is feature extraction, where neural networks extract critical feature vectors. Transformers have gained widespread adoption in multimodal feature extraction due to their powerful attention mechanisms, flexible token representations, and cross-modal universality. Transformer inference involves extensive matrix-vector multiplications, and its in-memory accelerators have been extensively researched.

In the hash mapping phase, high-dimensional feature vectors are compressed into compact hash codes using locality-sensitive hashing (LSH). This technique maps vectors with high cosine similarity to small Hamming distances, reducing storage and computational overhead. However, maintaining accuracy across multiple modalities and achieving energy-efficient hardware implementation remain significant challenges.

The feature retrieval phase computes hash sequence similarities to identify relevant items. While CAM enables rapid, parallel matching, its ability to discriminate similarity degrees diminishes with longer hash sequences.

To overcome these limitations, we propose a hybrid PIM-CAM retrieval architecture, as demonstrated in Fig. 2.

1) **Feature extraction:** A Transformer-based CLIP model extracts high-dimensional feature vectors.
2) **Hash mapping:** Device-level randomness in the PIM array enables energy-efficient locality-sensitive hashing, transforming floating-point vectors into 512-bit hash codes while preserving their similarity relationships.
3) **2-Stage Feature Retrieval:** First is the search space reduction stage. CAM's in-situ search capability filters entries based on a Hamming distance threshold, dramatically reducing the search space. To handle longer hash sequences, a segmented threshold matching and score aggregation mechanism is introduced. Next is the final refinement. Cosine similarity reranking is applied to the CAM-filtered subset to ensure high retrieval accuracy.

This hybrid architecture leverages PIM for computationally efficient hashing and CAM for rapid, parallel search, achieving a balance between performance, energy efficiency, and retrieval accuracy.

IV. PIM for Locality-Sensitive Hashing

A. Motivation and Challenge

Locality-sensitive hashing (LSH), as defined in Eq. (1), introduces several challenges when implemented in hardware. The first challenge is the significant data migration overhead associated with matrix-vector multiplication in traditional von Neumann architectures, which limits computational efficiency. Analog in-memory crossbars address this issue by encoding vectors in device conductance and input voltage, enabling dot product computation through current summation based on Kirchhoff's current law.

Two additional challenges emerge in adapting LSH to analog crossbars: First, efficiently storing random vectors with a mean of zero, as required by LSH principles. Second, extracting the sign of dot product results in an energy-efficient manner without relying on energy-intensive analog-to-digital converters (ADCs). Below demonstrates how to leverage the intrinsic hardware randomness of MTJ device in STT-MRAM and the differential encoding strategies to address the issues.

B. Leveraging Hardware Randomness for Random Vector Storage

The manufacturing process of MTJs introduces process variations, such as deviations in the thickness of the oxide barrier layer (γ_{tox}), the thickness of the free layer (γ_{tf}), and tunnel magnetoresistance (TMR) ratio variations, resulting in stochastic fluctuations in device resistance. These natural variations enable the generation of random vectors, which are essential for LSH.

To encode random vectors in an analog crossbar, the conductance of resistive devices represents individual vector elements. However, because device conductance is inherently positive, directly encoding zero-mean random vectors is infeasible. Differential encoding overcomes this limitation by

representing a random vector as the difference in conductance between adjacent columns in the array. Specifically, each MTJ is initialized to a high-resistance state, producing conductance values that are random yet stable. The conductance difference between adjacent columns encodes a zero-mean random vector. The dot product between an input feature vector x and the encoded random vector a can then be expressed as:

$$\begin{aligned} x \cdot a &= \sum (V_i \cdot (G_i^+ - G_i^-)) \\ &= \sum (V_i \cdot G_i^+ - V_i \cdot G_i^-) \\ &= I^+ - I^- \end{aligned} \tag{2}$$

where input voltage V_i and conductance difference $(G_i^+ - G_i^-)$ between adjacent columns represent the values of the i-th dimension of x and a respectively. Therefore, I^+ and I^- represent the cumulative currents flowing through adjacent columns, effectively capturing the dot product result in the analog domain.

C. Energy-Efficient TLSH Hash Code Generation

Traditional analog computing systems rely on ADCs to digitize analog signals, introducing significant energy overhead. However, LSH requires only the sign of the dot product rather than its precise value. This allows direct conversion of analog signals to hash codes by comparing current magnitudes between adjacent columns, eliminating the need for ADCs. In this approach, the sign of the current difference determines the hash bit (0 or 1), while small differences are encoded as an "X" state to indicate ambiguity. This ternary logic provides flexibility while reducing computational complexity.

Fig. 3 illustrates the energy-efficient TLSH sensing circuit, which eliminates ADC usage by leveraging lightweight analog components. The circuit operates as follows: The circuit employs trans-impedance amplifiers (TIAs) to convert column current sums to corresponding voltage signals. Subtractors generate voltage differences between adjacent columns, reflecting the relative magnitudes of current sums. In the TLSH sensing block, comparators (Comp1, Comp2, and Comp3) process these voltage differences to produce 2-bit current-represented hash codes. Specifically, Comp1 determines the sign of the difference (positive/negative). Comp2 and Comp3 classify differences as small (ambiguous, "X") or significant (definitive, 0 or 1) based on a configurable threshold. The ternary encoding block outputs voltage levels representing hash codes: 0, 1, or grounded (for "X"). By directly converting current differences into hash codes, the TLSH sensing circuit minimizes energy consumption while maintaining high accuracy in hash code generation.

V. CAM For Search Space Filtering

A. Design Motivation

In MRAM-based CAM design, each hash code is represented by the conductance of two MTJs, while the query hash code is encoded in dual voltage. This configuration seamlessly integrates the 2-bit voltage-format hash codes generated by the

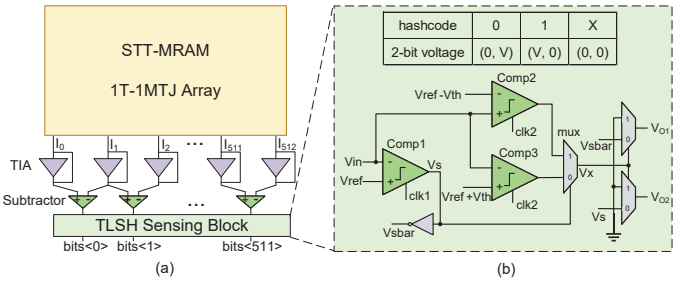


Fig. 3: The peripheral circuit for hashing mapping: (a) overall structure, and (b) TLSH sensing block that converts the input voltage difference into a 2-bit voltage representation of the hash code.

TLSH sensing block as CAM query voltages, enabling simultaneous hash mapping and Hamming distance computation in an energy-efficient manner.

However, deploying CAM for Hamming distance-based retrieval in multimodal scenarios reveals a critical trade-off. While high retrieval accuracy requires long hash sequences, the relatively low tunnel magnetoresistance (TMR) ratio of MRAM devices limits CAM's ability to differentiate similarity levels across extended sequences. This limitation diminishes retrieval precision for longer hash codes.

To address this, we repurpose CAM as a filtering mechanism rather than a comprehensive Hamming distance sorting solution. Specifically, CAM identifies entries with Hamming distances below predefined thresholds, narrowing the search space for subsequent cosine similarity reranking. To complement this filtering step, we propose a segment-based aggregation and concatenation mechanism that uses threshold-based scoring for individual hash code segments. This hybrid two-stage approach reduces latency and energy consumption while preserving high retrieval accuracy. The following sections detail the segmentation and merging process, as well as our methodology for selecting optimal thresholds.

B. CAM-Based Segmentation and Merging

Our proposed threshold filtering approach segments a K-bit hash sequence into n equal parts, each processed in parallel by a CAM sub-array of word length w. Fig. 4 illustrates the architecture.

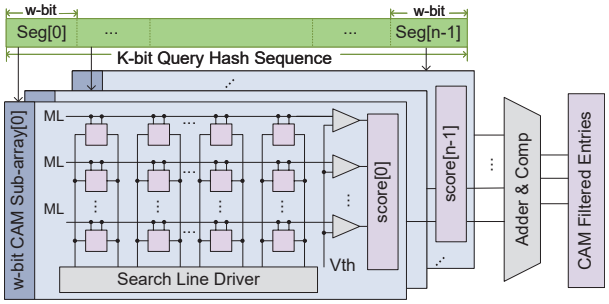


Fig. 4: Segmented threshold matching and merging mechanism of CAM filtering.

To optimize the trade-off between retrieval precision and reduced search space, we introduce a multi-threshold Hamming distance mechanism for each w-bit CAM sub-array. The scoring mechanism assigns a value to the i-th segment based on its Hamming distance relative to m predefined thresholds, as shown below:

$$score(i) = \sum_{j=0}^{m-1} (hamming_dist(i) \leq threshold(j)) \quad (3)$$

For each stored entry, a total score is computed by summing the scores across all segments, as defined in Eq. (4):

$$total_score(entry) = \sum_{i=0}^{n-1} score(i) \quad (4)$$

Entries exceeding a specified threshold are advanced to the next stage for further processing.

In our design, 512-bit hash sequences are divided into eight 64-bit segments, with each segment processed by a dedicated 64-bit CAM sub-array. We employ three distinct thresholds to evaluate Hamming distances within each segment. These thresholds are carefully calibrated to strike a balance between retrieval accuracy and search space reduction.

C. Threshold Selection

Given the fact that intra-class Hamming distances are generally smaller than inter-class distances, we defined a primary threshold th positioned at the boundary where intra-class distances predominantly fall below, while inter-class distances exceed this value.

To further optimize retrieval, two additional thresholds are introduced: A lower threshold (th_{low}) below th, which rewards intra-class pairs with very small Hamming distances, ensuring these entries achieve higher scores than non-matching pairs; A higher threshold (th_{high}) slightly above th, which allows more entries to be included in scoring to enhance overall retrieval accuracy. These thresholds are uniformly distributed across n segments, enabling segment-wise Hamming distance evaluation.

After implementing the segmentation and score aggregation mechanism, we analyzed the total score distributions of intra-class and inter-class pairs. A score threshold was then selected to effectively separate matching pairs (which predominantly score above the threshold) from non-matching pairs (which generally score below). Entries meeting or exceeding this threshold proceed to the next stage for cosine similarity reranking.

VI. Evaluations

A. Experimental Setup

Our evaluation framework encompasses multiple levels of analysis to assess both retrieval accuracy and hardware efficiency. At the device level, we employ the Brinkman model and Landau-Lifshitz-Gilbert (LLG) equation [10] to characterize MTJ behavior. The key parameters for MTJ simulation are demonstrated in TABLE I. Circuit-level analysis utilizes the Cadence suite for detailed simulation of PIM and CAM components. At the behavioral level, we implement Python-based simulations of hardware hash mapping and retrieval processes to evaluate retrieval accuracy.

TABLE I: Key Parameters for MTJ Simulation.

MTJ of STT-MRAM Parameters	Value
Effective Magnetic Anisotropy (H_K)	$3.58e5\ A/m$
Saturation magnetization (M_s)	$6e5\ A/m$
Gyromagnetic ratio	$1.76e7\ rad/(s \cdot T)$
Oxide Layer Thickness (t_{ox})	$0.85nm$
Damping Constant (α)	0.03
Tunnel Magnetoresistance (TMR)	3.6

For fair comparison, we utilize the ViT-L/14@336px model from [15] for feature extraction, mapping 768-dimensional float16 feature vectors to 512-bit hash sequences. We evaluate our method on three widely-used multimodal information retrieval datasets: MSCOCO [16], Flickr8K [17], and Flickr30K [18]. Following the CLIP's evaluation protocol, we employ recall rate (R@k) as the primary accuracy metric.

To provide a comprehensive analysis, we compare both accuracy and hardware efficiency. Accuracy evaluations include comparisons against CPU-based cosine similarity retrieval and hash-based Hamming distance retrieval (both CPU and in-memory implementations). Hardware efficiency metrics, including latency and energy consumption, are benchmarked against PyTorch-based similarity calculation and sorting operations on an Intel(R) Core(TM) i9-14900KF CPU, providing a standardized baseline against conventional computing approaches.

B. Optimal Threshold Selection

We demonstrate the proposed threshold selection approach with the Flickr8K dataset. Fig. 5 (a) reveals distinct Hamming distance distributions: intra-class distances predominantly typically fall below 220, while inter-class distances are generally above this value. Based on this natural separation, 220 is selected as the primary threshold. To refine discrimination, we introduce additional threshold at 210 to reward smaller intra-class distances, and a threshold at 225 to ensure comprehensive coverage.

After aggregating scores across eight segments, Fig. 5 (b) and (c) illustrate the resulting intra-class and inter-class score distributions. Intra-class entries consistently achieve scores of 9 or higher, while inter-class entries cluster below this value. Consequently, we adopt 9 as our score threshold, filtering entries scoring 9 or above into the candidate subset.

Fig. 5 (d) explores the trade-off between search space size and retrieval accuracy under varying thresholds. Lower thresholds expand the filtered set, increasing reranking overhead but improving recall rates. Retrieval accuracy improves rapidly with increasing search space before plateauing. Balancing retrieval accuracy with energy consumption, we select thresholds that filtered approximately 13.9% of the search space.

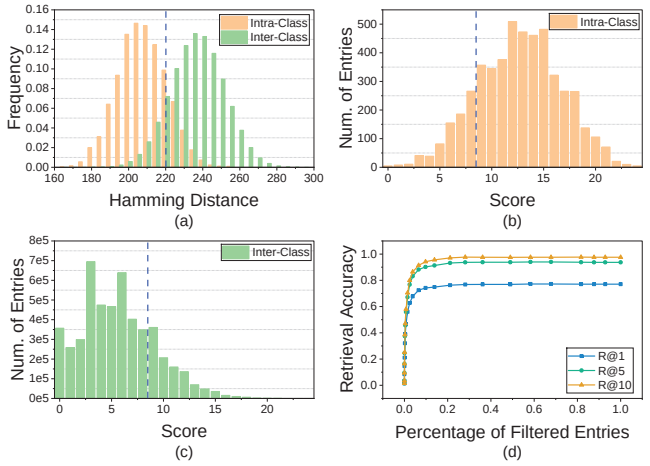


Fig. 5: Threshold selection criteria: (a) distribution of Hamming distances for intra-class and inter-class pairs, (b) distribution of total scores for intra-class pairs, (c) distribution of scores for inter-class pairs, and (d) trade-off between retrieval accuracy and filtered entry ratio across different threshold values.

C. Retrieval Accuracy

For a comprehensive evaluation, we compare recall rates across three approaches: traditional CPU-based cosine similarity retrieval [15], full in-memory hash and Hamming distance retrieval using PIM and CAM [12], and our proposed method. To identify the primary sources of accuracy loss, we also evaluate how different implementations of hash and Hamming distance retrieval affect recall rates, as shown in Fig. 6.

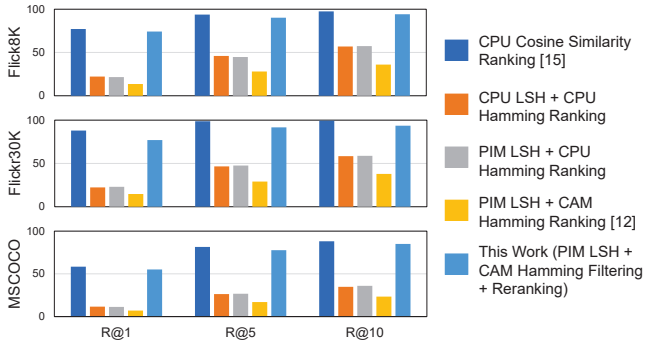


Fig. 6: Recall rate comparison between different implementations of hashing and retrieval.

In multimodal contexts, Hamming distance retrieval on LSH-generated hash sequences shows significant accuracy degradation compared to cosine similarity retrieval, with CAM's matching limitations further exacerbating this loss. While in-memory LSH implementation maintains accuracy comparable to CPU-based approaches, PIM delivers superior energy efficiency benefits. Our approach preserves strong recall rates, with only a $1\% \sim 4\%$ degradation. Notably, despite the MSCOCO dataset containing $25,000$ entries, the recall rate degradation is only 3%, demonstrating the scalability of our approach for large-scale retrieval tasks.

D. Latency and Energy Consumption

CAM-based Hamming distance ranking exhibits Unacceptable low accuracy in multimodal contexts, rendering it impractical for real-world applications. Therefore, we restrict our latency and energy consumption comparisons to CPU-based cosine similarity ranking.

TABLE II compares our proposed method against CPU ranking, examining CAM-filtered entries, latency, and energy consumption. Compared to CPU-based cosine similarity computation and ranking, the hash mapping and CAM filtering account for only a small fraction of the total energy consumption and latency, thanks to the energy-saving in-memory computation and search capabilities of MRAM-based PIM and CAM. Our CAM filtering approach achieves an average $10.85\times$ reduction in cosine similarity search entries, resulting in on average $9.45\times$ lower latency and $30.20\times$ lower energy consumption.

TABLE II: Reduction Ratios in Filtered Search Space, Latency and Energy Consumption Compared to the CPU Baseline.

Dataset	Num. of Entries	Latency	Energy
Flickr8K	10.23	5.45	6.58
Flickr30K	13.80	6.28	7.83
MSCOCO	8.53	16.62	76.20
Average	10.85	9.45	30.20

Note that the efficiency gains scale with dataset size due to the O(nlogn) complexity of CPU-based similarity sorting. For instance, filtering Flickr8K's search space from $5,000$ to 362 entries yields a $7.83\times$ reduction in energy consumption, while filtering MSCOCO from $25,000$ to $2,930$ entries achieves a $76.20\times$ reduction. These results demonstrate our method's significant advantages in large-scale retrieval applications, where the benefits become increasingly pronounced with larger datasets.

VII. Conclusion

This paper presents a novel multimodal information retrieval architecture that integrates PIM and CAM, achieving both high retrieval accuracy and energy efficiency. In the feature extraction stage, the Transformer model is employed to extract multimodal feature vectors. In the hash mapping stage, device-level randomness is leveraged to implement in-memory hashing mapping. In the feature retrieval stage, CAM filters candidate subsets based on Hamming distance thresholds, followed by precise cosine similarity retrieval within the reduced search space. The accuracy has significantly improved compared to pure Hamming distance sorting. When compared to cosine similarity ranking, our method achieves almost identical level of accuracy with a $9.45\times$ reduction in latency and $30.20\times$ reduction in energy consumption. Furthermore, our method exhibits excellent scalability in handling large-scale data, which is of significant practical importance given the exponential growth of multimodal data nowadays.

REFERENCES

[1] R. Zhao, H. Chen, W. Wang, F. Jiao, C. Qin, B. Ding, X. Guo, M. Li, X. Li, S. Joty *et al.*, "Retrieving multimodal information for augmented generation: A survey," in *The 2023 Conference on Empirical Methods in Natural Language Processing*.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[3] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müller, S. Beyer, M. Niemier, X. S. Hu *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[4] J. Wang, H. T. Shen, J. Song, and J. Ji, "Hashing for similarity search: A survey," *arXiv preprint arXiv:1408.2927*, 2014.

[5] Y. Zhao, J. Yang, B. Li, X. Cheng, X. Ye, X. Wang, X. Jia, Z. Wang, Y. Zhang, and W. Zhao, "Nand-spin-based processing-in-mram architecture for convolutional neural network acceleration," *Science China Information Sciences*, vol. 66, no. 4, p. 142401, 2023.

[6] Z. Sun, S. Kvatinsky, X. Si, A. Mehonic, Y. Cai, and R. Huang, "A full spectrum of computing-in-memory technologies," *Nature Electronics*, vol. 6, no. 11, pp. 823–835, 2023.

[7] X. Wang, J. Yang, Y. Zhao, X. Jia, R. Yin, X. Chen, G. Qu, and W. Zhao, "Triangle counting accelerations: From algorithm to in-memory computing architecture," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2462–2472, 2021.

[8] X. Ma, W. Zhang, X. Wang, T. Yu, B. Wu, G. Qu, and W. Zhao, "A combined content addressable memory and in-memory processing approach for k-clique counting acceleration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[9] E. Ambrosi, C. H. Wu, M. Y. Song, H. Y. Lee, K. S. Li, C. C. Lin, C. F. Hsu, C. C. Kuo, W. N. Chang, Y. J. Chen, C. H. Lin, J. M. Shieh, C. H. Shen, T. Y. Lee, T. H. Hou, and X. Y. Bao, "Low voltage ($<$1.8 v) and high endurance ($>$1m) 1-selector/1-stt-mram with ultra-low (1 ppb) read disturb for high density embedded memory arrays," in *2023 International Electron Devices Meeting (IEDM)*, 2023, pp. 1–4.

[10] Z. Guo, J. Yin, Y. Bai, D. Zhu, K. Shi, G. Wang, K. Cao, and W. Zhao, "Spintronics for energy-efficient computing: An overview and outlook," *Proceedings of the IEEE*, 2021.

[11] F. Liu, W. Zhao, Y. Chen, Z. Wang, Z. He, R. Yang, Q. Tang, T. Yang, C. Zhuo, and L. Jiang, "Pim-dh: Reram-based processing-in-memory architecture for deep hashing acceleration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1087–1092.

[12] R. Mao, B. Wen, A. Kazemi, Y. Zhao, A. F. Laguna, R. Lin, N. Wong, M. Niemier, X. S. Hu, X. Sheng *et al.*, "Experimentally validated memristive memory augmented neural network with efficient hashing and similarity search," *Nature communications*, vol. 13, no. 1, p. 6284, 2022.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, 2019, pp. 4171–4186.

[14] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," in *Proc. Interspeech 2021*, 2021, pp. 571–575.

[15] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[16] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer vision–ECCV 2014: 13th European conference, zurich, Switzerland, September 6-12, 2014, proceedings, part v 13*, 2014, pp. 740–755.

[17] M. Hodosh, P. Young, and J. Hockenmaier, "Framing image description as a ranking task: Data, models and evaluation metrics," *Journal of Artificial Intelligence Research*, vol. 47, pp. 853–899, 2013.

[18] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier, "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions," *Transactions of the association for computational linguistics*, vol. 2, pp. 67–78, 2014.

ZK-Hammer: Leaking Secrets from Zero-Knowledge Proofs via Rowhammer

Junkai Liang[1,3,4], Xin Zhang[2,3,4], Daqi Hu[2,3,4], Qingni Shen[2,3,4,*], Yuejian Fang[2,3,4], Zhonghai Wu[1,2,3,4,*]
[1]School of Computer Science, Peking University
[2]National Engineering Research Center for Software Engineering, Peking University
[3]School of Software and Microelectronics, Peking University
[4]PKU-OCTA Laboratory for Blockchain and Privacy Computing, Peking University
Email: ljknjupku@gmail.com, zhangxin00@stu.pku.edu.cn, hudaqi0507@gmail.com
qingnishen@pku.edu.cn,fangyj@ss.pku.edu.cn, wuzh@pku.edu.cn

***Abstract*—Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) schemes have been a promising technique in verified computation. Zk-SNARK schemes were designed to be mathematically secure against cryptographic attacks and it remains unclear whether they are vulnerable to fault injection attacks. In this work, we provide a positive answer by presenting ZK-Hammer, which leaks secrets from zk-SNARK schemes via Rowhammer. We incur faults in the exponentiate variables in the Quadratic Arithmetic Program (QAP) problem. Then we analyze the faulty proof using the bilinear pairing technique and manage to recover the secret. We employ a Rowhammer fault evaluation in `libsnark` and identify 3 CVEs.**

I. Introduction

Zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) schemes have significantly evolved from their conceptual origins in complexity theory and cryptography [1] to their current role as fundamental components, enabling a wide array of practical applications including blockchain payments [2], smart contract [3], and other academic areas like machine learning [4], multiparty computation [5] and post-quantum cryptography [6]. As a mechanism for an untrusted prover to convince a verifier of the correctness of a computation without disclosing any other information (zero-knowledge), zk-SNARKs have transitioned from theoretical constructs to widely used practical implementations over the past decade [7]–[9].

Existing zk-SNARK schemes have been designed to have cryptographic guarantees such as soundness, completeness and zero knowledge. However, those nice properties only hold in ideal conditions, e.g., circumstances that do not consider side-channel attacks, correct implementations of upper-level applications, etc. There are no adequate works that recognise the severity of the hardware fault attacks. A powerful technique in this category is a fault injection attack through Rowhammer [10].

Rowhammer can inject a software-induced fault in the main memory of a commodity system through hardware bit flips in DRAM [11]–[13]. Since the Rowhammer vulnerability was discovered in hardware in 2014 [11], it has been exploited to break two categories of practical cryptographic schemes, signature schemes and encryption schemes (a.k.a key exchange mechanisms). For the first category, there have been numerous successful attacks against ECDSA [14]–[17], EdDSA [18] and post-quantum signatures such as LUOV [19] and Dilithium [20]. For the second category, successful attacks against RSA [21]–[23], Frodo [24] and Kyber [25], [26] have also been reported. Those works expand the impact and highlight the severity of Rowhammer attacks.

Despite the growing threat of Rowhammer and all the progress made in zk-SNARK field, we observe that the vulnerability of zk-SNARK schemes against Rowhammer-based attacks has not been adequately evaluated. Thus, we are interested in the following questions:

1) *If a fault occurs in zk-SNARK schemes, is it meaningful to an attacker?*
2) *If so, can an attacker exploit the vulnerability and deduce some secrets?*
3) *Further, is the vulnerability general to other zk-SNARK schemes and can we give some countermeasures?*

Our work: In this paper, we offer affirmative responses by presenting ZK-Hammer: a thorough fault analysis and evaluation of Quadratic Arithmetic Program (QAP) based zk-SNARK schemes. Due to the complexity of zk-SNARKs (see Section II), most of the faults will simply result in a failure in the proof generation which cannot be exploitable. A general challenge in our attack is to find the meaningful position of the fault which may result in the leakage of secrets. We managed to solve this challenge by designing a *Fault + Recover* process. In the faulting phase, we introduce faults to the secret we aim to leak in the proof generation. The faulty secret will be multiplied by its selector polynomial, subjected to a group exponentiation and then published as part of the proof. In the recovery phase, with the faulty proof, we do a modified verification. Unlike the original verification algorithm, we multiply a correction term in the pairing operation. Note that the term is correlated to the fault and if this modified verification passes, we can deduce the fault as bits of the secret leakage. We have designed an algorithm to do this phase automatically and we further emphasize our analysis applies

This work was supported by the National Natural Science Foundation of China (Grant No. 61672062). * Corresponding authors.

to all the zk-SNARK schemes in the QAP category.

In our evaluation, we have implemented range proof as an example application of `libsnark`. Utilizing Rowhammer to inject a one-bit flip, we got 160 faulty proofs. Utilizing our proposed analysis algorithm, we can leak more than 80% information of the secret (personal balance in the range-proof case) and thus break the zero-knowledge property of zk-SNARK schemes. To mitigate our attack, we provide a few algorithmic defences specific to zk-SNARK and some general hardware defences which we have reported to `libsnark`.

Contributions: The main contributions of this paper are as follows:

- To the best of our knowledge, we introduce the first fault injection analysis and evaluate ZK-Hammer against zk-SNARK schemes for general circuits. ZK-Hammer identifies vulnerable parameters and contains a necessary algorithm to recover the secrets.
- We employ ZK-Hammer to other QAP-based zk-SNARK schemes (i.e., Groth and PGHR) and demonstrate our attack applies to other schemes in the QAP category.
- We evaluate Rowhammer attacks against the GGPR scheme using the recent popular cryptographic library, i.e., `libsnark`. When faulting the secret in our test example, we achieve 80% leakage of information with 160 faulty proofs.
- We have provided countermeasures to mitigate our attack. Our countermeasures receive positive responses from developers of `libsnark`.

Responsible Disclosure: We have disclosed our findings to the developers of `libsnark`. We have been assigned 3 CVE numbers (available at https://github.com/liang-junkai/ZK-hammer), which track the fault-injection vulnerability in three zk-SNARK schemes.

II. Background

A. Rowhammer Attacks Against Cryptography

Rowhammer is a software-induced fault in DRAM that can cause bit flips in the main memory of commodity systems [11]. Specifically, when a DRAM row is accessed frequently (or hammered), it may lead to permanent charge leakage, resulting in bit flips in adjacent rows. The rows that are vulnerable are termed *victim rows*, whereas the frequently accessed rows are referred to as *aggressor rows*. A double-sided Rowhammer is shown in Figure 1.

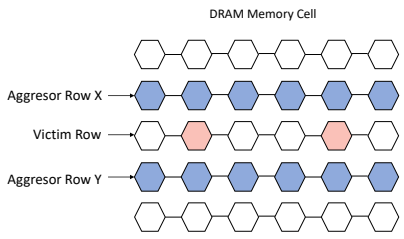


Fig. 1: Demonstration of double-sided Rowhammer

Existing Rowhammer attacks in the context of cryptography target signature schemes [14]–[16], [18]–[21] and key-exchange mechanisms (KEMs) [22]–[26]. For the first category, the attacks against signature schemes use a *Fault+Recover* pattern to leak the secret key. When faulting public keys, a single faulty signature and a valid one are compared to compute the secret key. This comparison process is called differential fault analysis (DFA). When faulting secrets, the attacker leaks one bit of secret key by one faulty signature, and a full recovery requires hundreds of unique faulty signatures. For the second category, the attacker faults the control flow or secret key in the decryption query and recovers usable information from the victim's response. We notice that there is no fault analysis against zk-SNARK schemes and all the analyses above are ad-hoc which cannot apply to other cryptographic schemes. Our work utilizes the *Fault+Recover* idea in the attacks against signature schemes but the derivation is different and more complicated in zk-SNARK schemes than in traditional signature schemes.

B. Zk-SNARK Scheme

Zk-SNARKs are cryptographic proof systems that enable the verification of NP computations with significantly reduced complexity compared to classical NP verification methods. Zk-SNARKs have been instrumental in advancing the field of cryptography by providing efficient and secure methods for proving knowledge without revealing the underlying information. The definition of zk-SNARK is as follows:

Definition II.1 (Zk-SNARK)**.** A zk-SNARK scheme consists of three algorithms: $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ defined as follows:

- $\mathsf{Setup}(\mathsf{pp}) \rightarrow (\mathsf{pk}, \mathsf{vk})$: Given public parameters pp as input, the algorithm outputs proving and verification keys pk and vk.
- $\mathsf{Prove}(\mathsf{pk}, x, w, R) \rightarrow \pi$: The prover generates the proof π related to the key pk, instance and witness pair (x, w), and the relation R.
- $\mathsf{Verify}(\mathsf{vk}, x, \pi) \rightarrow \{0, 1\}$. The verifier uses a verification key vk, the public instance x, and proof π to verify if the proof π is correct.

In addition, the definition of zk-SNARK requires 4 properties:

- **Perfect completeness:** Given $(x, w) \in R$, an honest prover can convince the verifier outputting 1.
- **Knowledge soundness:** Given $x \notin R$, a malicious prover interacting with the verifier can only make it output 1 with negligible probability.
- **Zero knowledge:** Given $x \in R$, a simulator without w can generate a transcript of an interaction between an honest prover and a potentially malicious verifier. This simulated transcript is computationally indistinguishable from a true proof which means the proof reveals no information about w.
- **Succinctness:** The proof size and verification time are sublinear in the statement to be proven.

C. QAP Technique

Our work targets QAP-based zk-SNARKs, which is one of the most fundamental and critical categories. The scheme GGPR [27] is the first practical zk-SNARK for general computation. A main strength of GGPR is that the proof has a constant size (9 group elements). Further, refinements have been made to shrink the proof size from 9 to 3 [28]–[32]. The basic idea of these works is to represent the computation the prover wants to prove as a QAP problem, a.k.a rank-1-constraint-system (R1CS). A formal representation of QAP (R1CS) is as follows:

Definition II.2 (QAP (R1CS))**.** A QAP problem Q over field $\mathbb{F}$ contains three sets of $m+1$ polynomials, $L = \{l_k(x)\}$, $R = \{r_k(x)\}$, $O = \{o_k(x)\}$, for $k = \{0, ..., m\}$, and a target polynomial $q(x)$. Suppose F is the computation we want to prove that has n inputs and $n^{'}$ outputs, for a total $N = n + n^{'}$ IO elements. Then we say Q computes F if there exists $c_1, ..., c_N$ as a valid assignment of inputs and outputs, and coefficients $c_{N+1}, ..., c_m$ such that $q(x)$ divides $p(x)$, where:

$$p(x) = (l_0(x) + \sum_{k=1}^{m}(c_k \cdot l_k(x))) \cdot (r_0(x) + \sum_{k=1}^{m}(c_k \cdot r_k(x))) - (o_0(x) + \sum_{k=1}^{m}(c_k \cdot o_k(x))) \tag{1}$$

III. Threat Model

Aligned with existing Rowhammer attacks against cryptographic schemes [14], [19], [20], our threat model is described as follows:

- An attacker can initiate an arbitrary, unprivileged user process without root privileges, co-located with a victim process in the same DRAM module. The victim process executes a specific zk-SNARK scheme, which the attacker can query for proof. Crucially, the attacker possesses knowledge of the scheme's parameters, including the secret key bit length and the public key description.
- The operating system functions correctly, isolating the attacker from the victim without any software vulnerabilities. The kernel hosting the victim process is considered secure, ensuring effective separation between the attacker and victim. Crucially, the attacker and victim processes do not need to run on the same kernel. As long as they share DRAM modules, a remote attack can be executed.
- The shared DRAM modules are susceptible to Rowhammer bit flips induced by the attacker. The location of a bit flip is specific to a particular DRAM module.

IV. ZK-Hammer: Fault Attack against zk-SNARK

In this section, we present zk-SNARK by analysing the details of QAP-based zk-SNARK schemes and explaining how to inject a fault and make the recovery. Based on the analysis, ZK-Hammer uses 3 components to do a successful attack: pre-processing, online faulting, and post-processing. Generally, ZK-Hammer contains 4 main components (which are shown in Figure 2):

- A guideline: providing the necessary formula derivation about where to inject the faults and how to leak secrets theoretically.
- Pre-processing phase: setups for Rowhammer attack. We use a malicious, unprivileged process to collect system memory information.
- Online faulting phase: mapping the secret to vulnerable locations and starting the row refresh.
- Post-processing phase: deducing the secret leakage according to the faulty proofs.

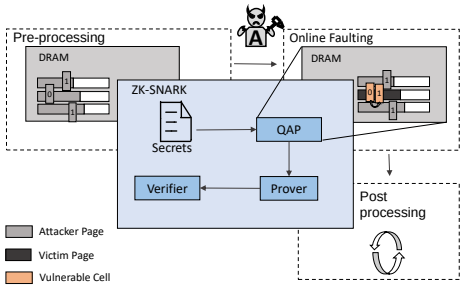


Fig. 2: An overview of ZK-Hammer.

A. Fault Analysis of QAP-based Zk-SNARK

We first briefly describe a classical scheme in this category named GGPR [27], based on which we show how to leak secrets when we incur faults on the parameters.

Gen$(F, 1^{\lambda})$**:** Let F be a function with N input and output values from field $\mathbb{F}$. Use QAP instance $Q = \{q(x), L, R, O\}$ of size m. Let $I_{mid} = N+1, ..., m$ denote the non-IO values. Let e be a bilinear map i.e., $e : G \times G \rightarrow G_T$ and g be a generator of G. Choose random values $s, \alpha, \beta_l, \beta_r, \beta_o, \gamma \leftarrow \mathbb{F}$. Compute public key PK_F for prover:

$$\begin{aligned} PK_F = \{&\{g^{l_k(s)}\}_{k\in I_{mid}}, \{g^{r_k(s)}\}_{k\in[m]}, \{g^{o_k(s)}\}_{k\in[m]}, \\ &\{g^{\alpha l_k(s)}\}_{k\in I_{mid}}, \{g^{\alpha r_k(s)}\}_{k\in[m]}, \{g^{\alpha o_k(s)}\}_{k\in[m]} \\ &\{g^{\beta_l l_k(s)}\}_{k\in I_{mid}}, \{g^{\beta_r r_k(s)}\}_{k\in[m]}, \{g^{\beta_o o_k(s)}\}_{k\in[m]} \\ &\{g^{s^i}\}_{i\in[m]}, \{g_i^{\alpha s^i}\}_{i\in[m]}\} \end{aligned} \tag{2}$$

Compute public key VK_F for verifier:

$$VK_F = \{g, g^{\alpha}, g^{\gamma}, g^{\beta_l\gamma}, g^{\beta_r\gamma}, g^{\beta_o\gamma}, g^{q(s)}, \{g^{l_k(s)}\}_{k\in[N]}\} \tag{3}$$

Here we do not inject faults to PK_F or VK_F, as they are not related to the witness. If we inject faults to the secret parameter s, from the attacker's view, the polynomials are evaluated at another point, which is also valid and we cannot deduce any information. Instead, we choose to inject faults in the next ***Prove*** procedure.

Prove(PK_F, u)**:** On input u, the prover first construct the QAP instance Q for $y \leftarrow F(u)$. As a result, the prover knows every value of $\{c_i\}_{i\in[m]}$ as a solution for the QAP instance

Q. Then she solves $h(x) = (L(x)R(x) - O(x))/q(x)$, and computes the proof π as:

$$\begin{aligned}\{&g^{L_{mid}(s)}, g^{R(s)}, g^{O(s)}, g^{h(s)},\\ &g^{\alpha L_{mid}(s)}, g^{\alpha R(s)}, g^{\alpha O(s)}, g^{\alpha h(s)},\\ &g^{\beta_l L(s)+\beta_r R(s)+\beta_o O(s)}\}\end{aligned} \tag{4}$$

where $L_{mid}(x) = \sum_{k\in I_{mid}} c_k l_k(x)$, $L(x) = \sum_{k\in[m]} c_k l_k(x)$, $R(x) = \sum_{k\in[m]} c_k r_k(x)$ and $O(x) = \sum_{k\in[m]} c_k o_k(x)$

Here $\{c_i\}_{i\in[m]}$ are the witnesses that the prover wants to keep secret. The guarantee of the zero-knowledge property is about the witnesses such that the prover convinces the verifier she knows $\{c_i\}_{i\in[m]}$ which satisfies the QAP problem without revealing those witnesses. When we inject fault on a term in the witness (e.g., c_i) when computing $L_{mid}(s)$, a part of the proof becomes (denote the faulty term as $c_i^{'}$ and the fault as Δc_i):

$$\begin{aligned}g^{L^{'}_{mid}(s)} &= g^{\sum_{k\in I_{mid}, k\neq i} c_k l_k(x) + c_i^{'} l_i(x)}\\ &= g^{\sum_{k\in I_{mid}, k\neq i} c_k l_k(x) + c_i l_i(x) - \Delta c_i l_i(x)}\\ &= g^{L_{mid}(s) - \Delta c_i l_i(x)}\end{aligned} \tag{5}$$

The derivation for $g^{\alpha L^{'}_{mid}(s)}$ is similar, we present as below:

$$g^{\alpha L^{'}_{mid}(s)} = g^{\alpha L_{mid}(s) - \alpha\Delta c_i l_i(x)} \tag{6}$$

The faulty part of the proof can be considered a valid one plus a correction term related to the fault. For $g^{L^{'}_{mid}(s)}$, the correction term is the fault Δc_i multiplies a selector polynomial $l_i(x)$. The selector polynomials are determined by the public function F and thus are public to the attacker. For $g^{\alpha L^{'}_{mid}(s)}$, α is secret but g^{α} is public and the attacker can do linear combinations by group operations. Below we use bilinear pairings in ***Verify*** procedure to eliminate the correction term in the equation and deduce the fault Δc_i. Because the bit flip is either from 0 to 1 or 1 to 0, the fault Δc_i denotes the leakage of the witness c_i.

Verify(VK_F, u, π)**:** First the verifier uses pairing function e to check α and β terms are correct to ensure the consistency of the variables and selector polynomials (e.g., $e(g^{L_{mid}(s)}, g^{\alpha}) = e(g^{\alpha L_{mid}(s)}, g)$). This requires 8 pairings and 3 for the β terms. Then the verifier computes her terms for inputs as $g^{L_{io}(s)} = \prod_{k\in[N]} (g^{l_k(s)})^{c_k}$ and do the final check for QAP divisibility requirement:

$$e(g^{h(s)}, g^{q(s)}) = e(g^{L_{mid}(s)} \cdot g^{L_{io}(s)}, g^{R(s)})/e(g^{O(s)}, g) \tag{7}$$

When a fault (as shown in Equation 5, 6) occurs, the check shown in Equation 7 will fail because of the correction computed above. As the length of c_i is measurable, we can enumerate all the possible Δc_i and compute a related correction term to pass the following equation:

$$\begin{aligned}&e(g^{L^{'}_{mid}(s)} \cdot g^{L_{io}(s)}, g^{R(s)})/e(g^{O(s)}, g)\\ &= e(g^{L_{mid}(s) - \Delta c_i l_i(x)} \cdot g^{L_{io}(s)}, g^{R(s)})/e(g^{O(s)}, g)\\ &= e(g^{-\Delta c_i l_i(s)}, g^{R(s)}) \cdot e(g^{L_{mid}(s)} \cdot g^{L_{io}(s)}, g^{R(s)})/e(g^{O(s)}, g)\\ &= e(g^{h(s)}, g^{q(s)}) \cdot e(g^{-\Delta c_i l_i(s)}, g^{R(s)})\end{aligned} \tag{8}$$

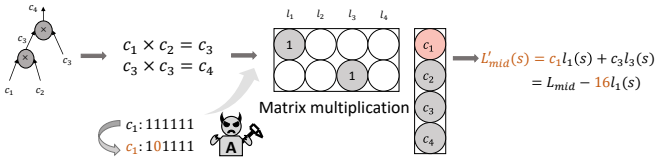


Fig. 3: An example of faulting QAP-based zk-SNARKs. The orange parts illustrate the faulting parameters and values.

when Equation 8 passes, we find the correct fault injected to the witness c_i and we can deduce the original bit value on the same position.

Generality: We apply the fault analysis to other QAP-based zk-SNARKs, such as PGHR [33] and Groth [29] to shown the generality of our attack. As long as the proof is generated by assigning variables to the QAP selector polynomials, our attack applies.

For the PGHR scheme, the only difference is that it uses 3 different group generators g_L, g_R, g_O for $L(x), R(x), O(x)$ respectively. The divisibility check is thus different but we can still extract the faulty term following a similar technique. The derivation is shown as follows:

$$\begin{aligned}&e(g_L^{L^{'}_{mid}(s)} \cdot g_L^{L_{io}(s)}, g_R^{R(s)})/e(g_O^{O(s)}, g)\\ &= e(g_L^{-\Delta c_i l_i(x)}, g_R^{R(s)}) \cdot e(g_L^{L_{mid}(s)} \cdot g_L^{L_{io}(s)}, g_R^{R(s)})/e(g_O^{O(s)}, g)\\ &= e(g^{h(s)}, g^{q(s)}) \cdot e(g_L^{-\Delta c_i l_i(x)}, g_R^{R(s)})\end{aligned} \tag{9}$$

The Groth scheme uses more random values to ensure the zero-knowledge property in only one pairing, e.g., a part of the proof is $L = g_L^{\alpha + L_{mid}(s) + r\delta}, R = g_R^{\beta + R(s) + s\delta}$, when a fault occurs on c_i, we can also extract the fault using a pairing operation:

$$e(L^{'}, R) = e(L, R) \cdot e(g^{\Delta c_i l_i(s)}, R) \tag{10}$$

Figure 3 shows an example of our fault on a simple computation with QAP representation. The computation shown in circuit form is first transformed to 2 constraints represented by multiplications. Then the prover constructs the corresponding selector polynomial matrix and multiplies it with the witnesses. Our fault takes place here right before the polynomial L_{mid} is computed. In our example, we fault the second bit of c_1 from 1 to 0, decreasing the absolute value by 16. The value $L^{'}_{mid}(s)$ used in the proof can be considered as a valid value $L_{mid}(s)$ subtracts a correction term related to the fault. If we find the value of the term, we know the position and direction of the fault and can deduce the original bit of c_1.

B. Offline Preparing Phase

To induce Rowhammer faults in c_i, we identify flippable locations in memory and gather addresses in adjacent hammering rows through *memory profiling*, which occurs before initializing the victim. We organize virtual pages in consecutive rows of the same DRAM bank and decode the DRAM addressing mechanism in two steps. First, do the virtual-to-physical address translation. This can be done through the

leverage of the consistent behaviour of the buddy allocator to encourage the kernel to allocate physically contiguous memory [34]. Second, do the physical-to-DRAM address mapping. This can be done through the leverage of a DRAM row buffer timing side channel [35]. The principle is that when two physical addresses (e.g., A and B) are located in the same bank, the time we alternatively access them is longer than the situation where they are in different banks due to row conflicts.

C. Online Faulting Phase

While Rowhammer-based faults can be injected into arbitrary parameters in the zk-SNARK scheme (e.g., the witness c_i, the evaluation of the selector polynomial $l_i(s)$), two requirements must be met for a successful Rowhammer fault attack. First, the fault must be traceable, meaning it can be identified by an exhaustive number of possible fault patterns. We conduct a cryptographic analysis and conclude that a single-bit fault of c_i is theoretically traceable. Second, the traceable parameter must be located in flippable memory. We employ the memory waylaying technique [36] to continuously evict pages within the page cache, ultimately positioning the targeted parameter at a flippable physical address.

Rowhammer Exploitation: Once the vulnerable bit is propagated to the flippable bits in DRAM, we begin by initializing two aggressor rows to hammer the victim row located in the middle. We employ a random pattern to initialize the aggressor rows. Specifically, we use random data to fill the adjacent aggressor rows, and then perform `Flush+Reload` on these rows repeatedly. This methodology allows us to trigger bit flips without relying on the knowledge of the victim row.

D. Post-processing Phase

In this phase, we recover the flipped witness bits by the faulty proofs. The error in the faulty proof is some multiples of Δc_i. Therefore, if we correct the faulty proof and make it satisfy Equation 8, we are able to find the position of the bit-flip. The main idea of this procedure is to correct the faulty proof by checking it using the proof verification algorithm. We explain it specifically in our bit tracing algorithm.

In Algorithm 1, we give public parameters in the scheme and the proof as inputs. The output is the position and value of the secret bit in the witness. After parsing the proof, we start enumerating the witness from c_0 to c_{m-1} and enumerating the position using bit_index. In lines 7-11, we compute Δ as a correction term the same way defined in Equation 8 and multiply it with the faulty part in the original proof. If the verification passes, the algorithm returns the desired output. Note that if the fault direction is from 0 to 1, we need to do a division and then do the verification as in lines 15-17.

Algorithm 1: The bit tracing algorithm

```
1  Input: Prover key PK_F; Verifier key VK_F and input
     u for the verifier; Faulty proof π' ; Max number of
     witness m
2  Output: (witness_index, bit_index, value) -
     Recovered secret bit
3  Parse the faulty proof as Equation 4 (Use π'[L] to
     denote the first term g^{L_mid(s)} in π')
4  for witness_index from 0 to m − 1 do
5      label witness_index as i
6      for bit_index from 1 to len(c_i) do
7          label bit_index as j
8          multiplier ← 2^{j−1}
9          Get g^{R(s)} from π, g^{l_i(s)} from PK_F
10         Compute Δ = e(g^{l_i(s)·multiplier}, g^{R(s)})
11         π'[L] = π'[L] · Δ
12         if Verify(VK_F, u, π) = true then
13             return (witness_index, bit_index, 1)
14         else
15             π'[L] = π'[L]/Δ
16             if Verify(VK_F, u, π) = true then
17                 return (witness_index, bit_index, 0)
18             end
19         end
20     end
21 end
```

V. Evaluation

A. Experimental setup

We perform the experiments using a machine with an Intel Core i3-10100 CPU (IceLake) and two Apacer DDR4-2666 8G DIMMs. The machine runs default Ubuntu 22.04 with Linux kernel 6.1.66. The zk-SNARK codes belong to the latest version of `libsnark`.

Victim implementation: We use the code of zk-SNARK scheme in `libsnark` and our example is the range proof, a popular application in blockchain. It proves that a value x is in a certain range $[0, 2^{32})$. In the blockchain scenario the value x may be the user's balance in the wallet. In the confidential setting, the value x must be kept secret to protect the user's privacy. Our attack goal is to leak x by ZK-Hammer.

B. Evaluation of Offline Preparing

To find nearby rows in the DRAM bank, we first obtain 2 MB of contiguous memory blocks by leveraging the deterministic behaviour of the buddy allocator. Using the contiguous memory chunk, we find the pages that are mapped into the same DRAM bank using a row buffer timing side channel. We repeatedly access different pages in the blocks and document access count and access time shown in Figure 4. The pages that are physically located on the same DRAM bank take longer to access due to row conflict.

Next, we use the double-sided Rowhammer technique for memory profiling, where each victim row is surrounded by an aggressor row from the top and bottom. Each hammering attempt consists of a finite loop targeting 4 pairs of virtual pages, with each pair being hammered for 250,000 rounds. After each hammering attempt, we scan the remaining unhammered memory for bit flips. This process allows us to collect

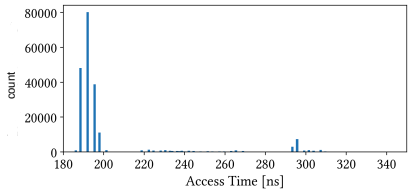
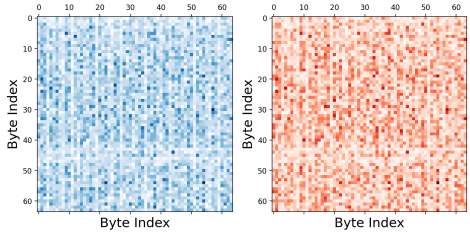


Fig. 4: Histogram of access times to the pages in the buffer.

a set of victim pages along with their corresponding aggressor pages. The page offset of a single-bit flip may vary among the collected victim pages. Figure 5 illustrates the distribution of bit-flip offsets across 4 KB-aligned pages.



(a) (b)

Fig. 5: Results of memory profiling. Bit flips from 1 to 0 (blue) and bit flips from 0 to 1 (red) accumulated over 4 KB pages.

C. Evaluation of Online Faulting

In the range-proof example, the secret value x is separated as a sequence of bits $x_0, ..., x_{31}$ because a proof of $x = \sum_{i=0}^{31} x_i \cdot 2^i$ and $x_i = 1$ or 0 yields the range of x. In this case, the vulnerable parameters $\{c_i\}_{i\in[0,31]}$ in the QAP instance are binary and we only need to fault one bit for each parameter. In the library, the prover first generates the QAP instance (a.k.a constraint system) and assigns values to all the c_i. The prover then calls a built-in function `r1cs_gg_ppzksnark_generator` to generate a key pair, prover key pk and verifier key vk. With pk, the prover can generate a proof. The time for our fault injection is between the value assignment of c_i and the finish of the proof. We measure the available faulting time which is 4245k CPU cycles. The time is not enough for a high-probability bit flip, and we compromise on the time to get 160 simulated faulty proofs. We argue that this will not decrease the severity of our attack as a small number of bits will significantly leak the privacy of users and break the zero-knowledge property of zk-SNARK.

D. Evaluation of Post-processing

We totally get 160 unique faulty proofs for 40 randomly chosen x in the domain. On average there are 4 faulty proofs for each x. Different from the fault attack of signature secret key [14] where only full key recovery is meaningful, we show that the 1 to 4-bit leakage of x reveals much information. In Figure 6, we compute the average of the percentage of the shrunk range with the 40 x's for each bit leak. We find that a 3 or 4-bit leak can shrink the guessing range of x by more than 70% for the attacker. A single-bit leak can also reveal nearly 20% information.

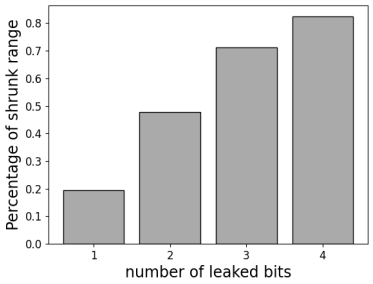


Fig. 6: Histogram of the attacking results. The ylabel represents the shrunk range percentage when an attacker guesses the value x after faulting.

VI. Countermeasures

In this section, we first discuss specific solutions to mitigate fault-injection attacks against zk-SNARK schemes. We then talk about general strategies to counteract Rowhammer.

A. Algorithmic Defenses

Verifying after proving: When a fault occurs in c_i, a faulty proof is generated. Given that the verifying algorithm can detect the fault by rejecting the faulty proof, the cryptographic library developer can append the verification step immediately after proving. However, this may incur significant overhead. In our example, this includes nine full pairing computations, slowing down the process by a factor of ten.

Redundancy check: The redundancy check includes both temporal and spatial checks [20]. The temporal redundancy check re-executes proofs and compares them. The spatial redundancy check involves storing multiple copies of secret parameters in random DRAM locations and comparing them after proving. Different results indicate a potential Rowhammer fault. These methods are generally more efficient but can be compromised.

B. General defenses

There are two categories of general countermeasures: software-only defenses [37], [38], which are compatible with existing commodity systems, and hardware-based defenses [39], which require hardware modifications. However, most of these defences have not been adopted by the industry and are not applicable to commodity systems and hardware.

VII. Conclusion and Future Work

In this paper, we presented the first novel fault attack against zk-SNARKs, coined as ZK-Hammer, that finds exploitable and traceable fault to a general category of zk-SNARK schemes and perform a post-Rowhammer analysis for secret recovery. To validate the vulnerability, we perform Rowhammer evaluations against the zk-SNARK codes in `libsnark`. Future work involves fault analysis of other categories of zk-SNARKs and more effective algorithmic countermeasures.

REFERENCES

[1] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *Providing sound foundations for cryptography: On the work of shafi goldwasser and silvio micali*, 2019, pp. 203–225.

[2] S. Bowe, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu, "Zexe: Enabling decentralized private computation," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 947–964.

[3] Z. Wan, Y. Zhou, and K. Ren, "Zk-authfeed: Protecting data feed to smart contracts with authenticated zero knowledge proof," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 2, pp. 1335–1347, 2022.

[4] T. Liu, X. Xie, and Y. Zhang, "Zkcnn: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2968–2985.

[5] D. Beaver, "Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority," *Journal of Cryptology*, vol. 4, pp. 75–122, 1991.

[6] M. Chase, D. Derler, S. Goldfeder, J. Katz, V. Kolesnikov, C. Orlandi, S. Ramacher, C. Rechberger, D. Slamanig, X. Wang *et al.*, "The picnic signature scheme," *Submission to NIST Post-Quantum Cryptography project*, 2020.

[7] "libsnark," Github https://github.com/scipr-lab/libsnark, 2014.

[8] M. Bellés-Muñoz, M. Isabel, J. L. Muñoz-Tapia, A. Rubio, and J. Baylina, "Circom: A circuit description language for building zero-knowledge applications," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 6, pp. 4733–4751, 2022.

[9] "Zokrates/zokrates," Github https://github.com/Zokrates/ZoKrates, 2021.

[10] Z. Zhang, D. Chen, J. Qi, Y. Cheng, S. Jiang, Y. Lin, Y. Gao, S. Nepal, Y. Zou, J. Zhang *et al.*, "Sok: Rowhammer on commodity operating systems," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 436–452.

[11] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: an experimental study of DRAM disturbance errors," in *International Symposium on Computer Architecture*, 2014, pp. 361–372.

[12] A. Kogler, J. Juffinger, S. Qazi, Y. Kim, M. Lipp, N. Boichat, E. Shiu, M. Nissler, and D. Gruss, "Half-double: Hammering from the next row over," in *USENIX Security Symposium*, 2022.

[13] H. Hassan, Y. C. Tugrul, J. S. Kim, V. Van der Veen, K. Razavi, and O. Mutlu, "Uncovering in-dram rowhammer protection mechanisms: A new methodology, custom rowhammer patterns, and implications," in *IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1198–1213.

[14] K. Mus, Y. Doröz, M. C. Tol, K. Rahman, and B. Sunar, "Jolt: Recovering tls signing keys via rowhammer faults," in *IEEE Symposium on Security and Privacy*. IEEE, 2023, pp. 1719–1736.

[15] K. Ryan, "Hardware-backed heist: Extracting ecdsa keys from qualcomm's trustzone," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 181–194.

[16] S. Weiser, D. Schrammel, L. Bodner, and R. Spreitzer, "Big numbers-big troubles: Systematically analyzing nonce leakage in ({EC) DSA} implementations," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1767–1784.

[17] K. Ryan, "Return of the hidden number problem.: A widespread and novel key extraction attack on ecdsa and dsa," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 146–168, 2019.

[18] D. Poddebniak, J. Somorovsky, S. Schinzel, M. Lochter, and P. Rösler, "Attacking deterministic signature schemes using fault attacks," in *IEEE European Symposium on Security and Privacy*, 2018, pp. 338–352.

[19] K. Mus, S. Islam, and B. Sunar, "Quantumhammer: a practical hybrid attack on the luov signature scheme," in *ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1071–1084.

[20] S. Islam, K. Mus, R. Singh, P. Schaumont, and B. Sunar, "Signature correction attack on dilithium signature scheme," in *IEEE European Symposium on Security and Privacy*, 2022, pp. 647–663.

[21] S. Bhattacharya and D. Mukhopadhyay, "Curious case of rowhammer: flipping secret exponent bits using timing analysis," in *Cryptographic Hardware and Embedded Systems*, 2016, pp. 602–624.

[22] Z. Weissman, T. Tiemann, D. Moghimi, E. Custodio, T. Eisenbarth, and B. Sunar, "Jackhammer: Efficient rowhammer on heterogeneous fpga-cpu platforms," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020.

[23] K. Razavi, B. Gras, E. Bosman, B. Preneel, C. Giuffrida, and H. Bos, "Flip Feng Shui: Hammering a needle in the software stack," in *USENIX Security Symposium*, 2016, pp. 1–18.

[24] M. Fahr Jr, H. Kippen, A. Kwong, T. Dang, J. Lichtinger, D. Dachman-Soled, D. Genkin, A. Nelson, R. Perlner, A. Yerukhimovich *et al.*, "When frodo flips: End-to-end key recovery on frodokem via rowhammer," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 979–993.

[25] P. Mondal, S. Kundu, S. Bhattacharya, A. Karmakar, and I. Verbauwhede, "A practical key-recovery attack on lwe-based key-encapsulation mechanism schemes using rowhammer," in *International Conference on Applied Cryptography and Network Security*. Springer, 2024, pp. 271–300.

[26] S. Amer, Y. Wang, H. Kippen, T. Dang, D. Genkin, A. Kwong, A. Nelson, and A. Yerukhimovich, "Pq-hammer: End-to-end key recovery attacks on post-quantum cryptography using rowhammer," in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 48–48.

[27] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, 2013, pp. 626–645.

[28] G. Danezis, C. Fournet, M. Kohlweiss, and B. Parno, "Pinocchio coin: building zerocoin from a succinct pairing-based proof system," in *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, 2013, pp. 27–30.

[29] J. Groth, "On the size of pairing-based non-interactive arguments," in *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, 2016, pp. 305–326.

[30] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Succinct {Non-Interactive} zero knowledge for a von neumann architecture," in *23rd USENIX Security Symposium (USENIX Security 14)*, 2014, pp. 781–796.

[31] A. Chiesa, E. Tromer, and M. Virza, "Cluster computing in zero knowledge," in *Advances in Cryptology-EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II 34*, 2015, pp. 371–403.

[32] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 253–270.

[33] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[34] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "RAMBleed: Reading bits in memory without accessing them," in *IEEE Symposium on Security and Privacy*, 2020.

[35] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "{DRAMA}: Exploiting {DRAM} addressing for {Cross-CPU} attacks," in *25th USENIX security symposium (USENIX security 16)*, 2016, pp. 565–581.

[36] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl, and Y. Yarom, "Another flip in the wall of rowhammer defenses," in *IEEE Symposium on Security and Privacy*, 2018, pp. 245–261.

[37] C. Gongye, Y. Luo, X. Xu, and Y. Fei, "Hammerdodger: a lightweight defense framework against rowhammer attack on dnns," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[38] J. Jung, H. So, W. Ko, S. S. Joshi, Y. Kim, Y. Ko, A. Shrivastava, and K. Lee, "Maintaining sanity: Algorithm-based comprehensive fault tolerance for cnns," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[39] N. Mishra, R. A. Mool, A. Chakraborty, and D. Mukhopadhyay, "Plug your volt: Protecting intel processors against dynamic voltage frequency scaling based fault attacks," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

Comprehensive Placement and Routing Framework with Guaranteed In-Cell Routability for Synthesizing Complementary-FET Cells

Zhengzhe Zheng[1], Yinuo Wu[1], Keyu Peng[1], Chao Wang[1,2,†], Ziran Zhu[1,2,†]
[1]National ASIC System Engineering Center, Southeast University, Nanjing, China
[2]National Center of Technology Innovation for EDA, Nanjing, China
Email: zzzkhorne@gmail.com, {220231843, kypeng, wangc, zrzhu}@seu.edu.cn

***Abstract*—As the technology node advances beyond 5nm, the conventional FinFET architecture encounters substantial scaling issues. Complementary-FET (CFET) technology, characterized by the vertical stacking of P-FET over N-FET or vice versa, has emerged as a promising solution. However, the inherent characteristics of CFET architecture, particularly the scarcity of routing resources, pose significant obstacles to in-cell routability and layout generation. In this paper, we develop a comprehensive placement and routing framework for synthesizing CFET cells. We first present a partitioning technique followed by a heuristic quality maintenance strategy for large-scale cells to ensure scalability and efficiency. Then, we propose a novel satisfiability modulo theories (SMT)-based placement method that incorporates partial routing to achieve minimum-width placement while ensuring in-cell routability. Particularly, the placement method also determines the pin positions for each net, which simplifies subsequent routing complexity. Finally, we propose a progressive metal routing method to address the challenges of routing resource scarcity and unidirectional routing in CFET technology, which includes a manual-inspired M0 routing followed by an integral linear programming (ILP)-based M1 and M2 routing. Compared with the state-of-the-art CFET cell generators, experimental results show that our algorithm achieves the smallest cell width for all tested cells, with 7 out of 30 cells exhibiting smaller widths. For the remaining 23 cells, which have the same cell width as those in other generators, our algorithm achieves the smallest M2 usage and total metal length.**

***Index Terms*—Standard cell, CFET, placement, routing, routability**

I. Introduction

With the continuous advancement of technology nodes, the conventional FinFET structure encounters scaling-related challenges [15]. Various methods have been proposed to reduce cell size, including fin depopulation technology [2] and nano-sheet FET structure [1, 9]. However, these scaling techniques are constrained by physical limitations, such as lateral spacing requirements. Recently, complementary-FET (CFET) technology [10, 12, 14] has emerged as a promising solution. Compared to conventional FinFET structures, CFET technology effectively reduces area by stacking NMOS and PMOS devices, as illustrated in Figure 1(a).

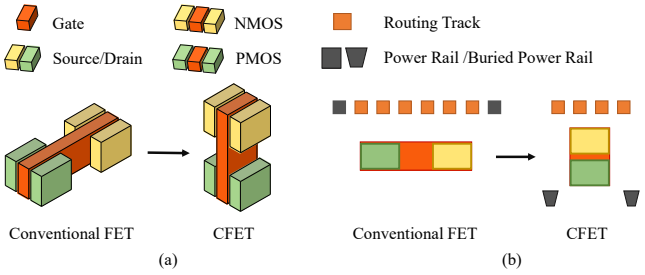


Fig. 1. (a) Example of conventional FET and CFET structures. (b) Cross-sectional view of conventional FET and CFET structures with routing tracks.

While CFET technology offers significant potential for reducing cell area, it also leads to fewer available routing tracks [13], as shown in Figure 1(b). To manage routing within the constrained resources, buried power rails (BPR) are incorporated, allowing power delivery through a backside network that frees up more routing space [11]. However, the limited routing resources and stringent design rules still pose considerable challenges for CFET in-cell routability. Ensuring in-cell routability during the placement stage is essential for a seamless transition to routing and achieving high-quality layouts.

In recent years, several works [4, 7] have developed innovative methods for CFET layout generation. The method in [4] proposes a satisfiability modulo theories (SMT)-based method for simultaneous placement and routing, enabling it to ensure in-cell routability and achieves the desired CFET layouts. However, due to its complex formulations and high computational requirements, the method is time-consuming and struggles with scalability, particularly when applied to large-scale cells. For faster layout generation, several acceleration techniques, such as those in [8], are typically incorporated into the SMT formulation. But such acceleration techniques may have a negative impact on solution quality, as noted in [6]. Furthermore, the work [7] proposes an efficient search-based method to generate CFET placements, carefully considering transistor folding and routing feasibility during the search process. This method excels at achieving multiple high-quality placement solutions. However, since the method does not tightly integrate placement and routing, the resulting placements may not fully ensure in-cell routability. Therefore, it is desirable to develop a more efficient and effective method to synthesize CFET cells that ensure in-cell routability within limited routing resources.

In this paper, we develop a comprehensive placement and routing framework to efficiently synthesize high-quality CFET cells. Our major contributions are summarized as follows:

- For large-scale cells, we propose a breadth-first search (BFS)-based partitioning technique to ensure scalability and efficiency. Additionally, we employ a strategy that combines heuristic sorting with sequential partition processing to mitigate the quality degradation that may be caused by partitioning.
- A novel SMT-based placement method that integrates partial routing is proposed to achieve optimal-width placement while ensuring in-cell routability. Particularly, this method also determines the pin positions for each net, which simplifies subsequent routing complexity and improves overall efficiency.
- Based on the pin positions of each net determined during the placement stage, we propose a progressive metal routing method that includes a manual-inspired M0 routing followed by an ILP-based M1 and M2 routing. This method effectively addresses the challenges of routing resource scarcity and unidirectional routing in CFET technology.
- Compared with the state-of-the-art CFET cell generators [7] and [4], experimental results show that our algorithm achieves an average reduction of 3% in cell width, with 7 out of 30 cells exhibiting smaller widths. For the remaining 23 cells, which have the same cell width as those in [7] and [4], our algorithm achieves the smallest M2 usage and total metal length.

II. Preliminaries

In this section, we first give the problem statement, and then introduce the cell architecture and challenges of CFET.

This work was partially supported by the National Natural Science Foundation of China under Grant 92373207, and the National Key Research and Development Program of China under Grant 2022YFB4400500. (†*Corresponding authors: Chao Wang and Ziran Zhu.*)

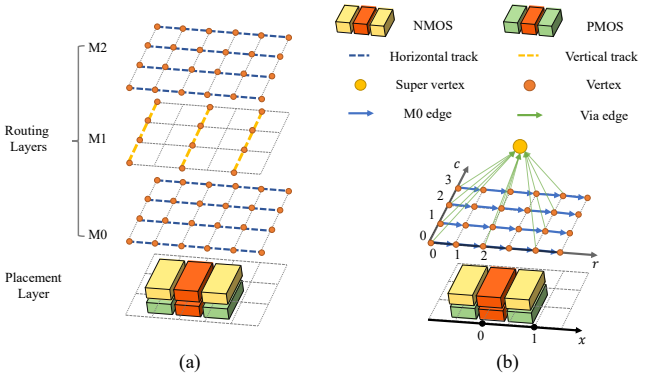


Fig. 2. (a) CFET cell architecture. (b) Modeling example for placement and partial routing.

A. Problem Statement

Given a SPICE netlist, cell architecture, and design rules, the objective of the CFET cell layout generation is to determine optimal transistor placement and routing paths that minimize cell width and overall wire length. Each transistor in the SPICE netlist includes three terminals (i.e., source, gate, and drain) associated with specific nets. Besides, the generated layout must meet the given placement and routing rules. The placement rules mainly include the non-overlap rule, diffusion sharing rule, and diffusion jog rule, while the routing rules encompass the minimum area rule (MAR), end-of-line (EOL), via rule (VR), parallel run length (PRL), step height rule (SHR), and minimum I/O pin length (MPL), consistent with [4, 7], and more details can be found in [4].

B. Cell Architecture and Challenges

The CFET cell architecture used in this paper, aligned with the architecture in [7] and [4], follows a 4.5T structure with layers arranged as placement, M0, M1, and M2 (see Figure 2(a)). The 4.5T structure incorporates four horizontal routing tracks in both the M0 and M2 layers for horizontal routing, while the M1 layer is designated for vertical routing, with vias enabling interlayer connections. Since the distribution of routing resources and specific routing rules, this CFET architecture poses the following two significant challenges to in-cell routability.

M1 routing resources: In the routing process, M1 tracks are required for I/O nets and the nets that cannot fully connect in the M0 layer. However, as shown in Figure 2(a), the vertical tracks on the M1 layer are sparsely arranged, limiting available routing resources. Additionally, the complex routing rules further restrict these resources; for instance, Figure 3(a) shows that a single vertical track cannot accommodate two different nets due to MAR and EOL rules.

M0 routing legalization: The routing results in the M0 layer tend to be more compact than those in the M1 and M2 layers, which presents greater challenges for legalization on the M0 layer. Figure 3(b) illustrates an example of a PRL violation; due to the need to adhere to other rules (such as the EOL rule), adjustments to rectify the issue are often not feasible. In fact, during M0 routing, it is common to encounter situations where adjusting a metal segment to meet one rule might lead to a conflict with another.

III. The Proposed Algorithm

Figure 4 summarizes the overall flow of our algorithm, which mainly consists of three stages: (1) preprocessing, (2) SMT-based routability-guaranteed placement, and (3) progressive metal routing with placement-specified pin positions. We shall detail these three major parts in the following subsections.

A. Preprocessing

The preprocessing stage includes three main steps: transistor folding, BFS-based partitioning, and a heuristic strategy for quality maintenance. These steps ensure compliance with the design rule for maximum

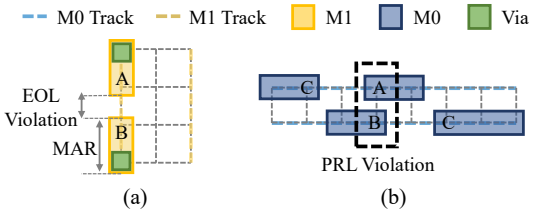


Fig. 3. Applying MAR/EOL/VR/PRL/SHR/MPL = 1/2/1/1/2/2. (a) Example of EOL rule violation on the vertical track of the M1 layer. (b) Example of routing result with PRL violation.

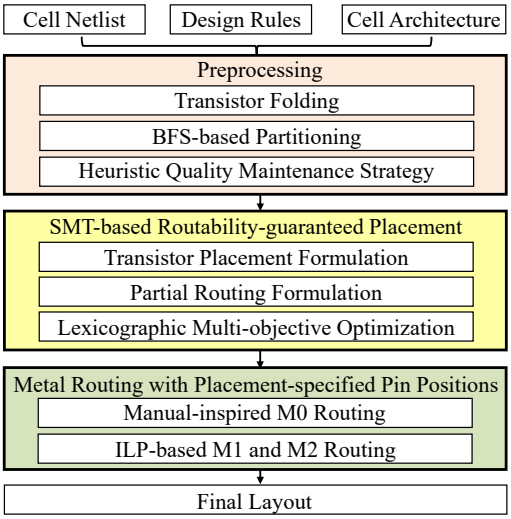


Fig. 4. The overall flow of our algorithm.

transistor width and also accelerate the subsequent placement process with minimal impact on solution quality.

1) Transistor Folding: When a transistor's width exceeds the specified maximum, it must be divided into smaller segments, known as "fingers". To reduce the risk of diffusion jog violations, our algorithm strives to maintain the uniform widths of the fingers while minimizing the total number of fingers after folding.

2) BFS-based Partitioning: To ensure the scalability and efficiency of the algorithm, a BFS-based partitioning method is employed for cells containing more than 20 transistors. The input netlist is represented as a graph, where each net connected to the active region is treated as a vertex, and each transistor is represented as an edge. For each vertex that connects both NMOS and PMOS transistors, we initiate a BFS search from this vertex, continuing until reaching a VDD or VSS. Transistors visited during a BFS iteration are grouped into the same partition. This method effectively ensures that transistors are systematically grouped based on their potential for diffusion-sharing.

3) Heuristic Quality Maintenance strategy: After partitioning, the partitions are organized by considering the following key factors to mitigate the potential impact on the solution quality: (1) Diffusion sharing potential: If a partition cannot share diffusion with both adjacent partitions simultaneously, the final cell width will increase. (2) Crossing nets range: When nets span more widely across partition boundaries, the final routing length tends to increase accordingly.

To find the best arrangement, all possible orderings are explored to maximize diffusion sharing and minimize the range of crossing nets. Considering these factors helps minimize the impact of partitioning on solution quality. After sorting, a partition list is generated and the partitions are sequentially placed in the specified order.

B. SMT-based Routability-Guaranteed Placement

In this section, we propose a novel SMT-based placement method that integrates partial routing to achieve optimal-width placement while en-

suring in-cell routability. Our routability-guaranteed placement model is depicted in Figure 2(b), which preserves the structure of the placement and M0 layers, while the M1 and M2 layers are abstracted as a super vertex. This abstraction significantly reduces the number of variables associated with vertices and edges.

For transistor placement, a one-dimensional x-coordinate system is established on the placement layer, as shown in Figure 2(b). When a transistor terminal needs to connect to the routing layer, four M0 vertices directly above the terminal are designated as potential pin positions. For partial routing, our model includes concrete vertices on the M0 layer and a super vertex that represents the higher routing layers. Besides, two types of directed edges are established: horizontal M0 edges connecting vertices from left to right, and via edges linking M0 vertices to the super vertex. To facilitate the estimation of M2 usage and wire length, a two-dimensional coordinate system is established on the M0 layer, with coordinates designated as r and c, as shown in Figure 2(b). Importantly, it should be noted that when a net connects to the super vertex through a via edge, it represents the net links to the M1 tracks in the actual cell architecture, as illustrated in Figure 5.

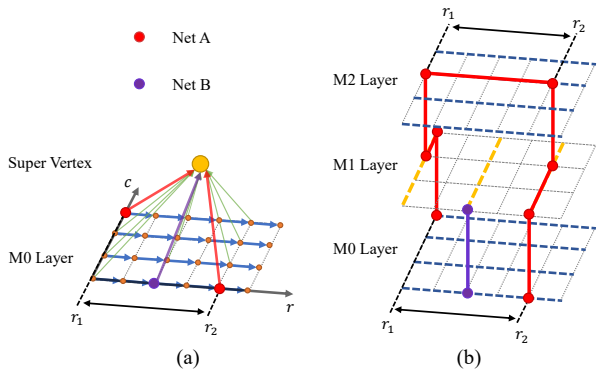


Fig. 5. (a) Example of nets connecting to the super vertex through via edges. (b) Mapping of (a) in the actual cell architecture.

In the following subsections, we shall detail the formulation for our routability-guaranteed placement model. The primary variables and symbols are listed in Table I.

TABLE I
NOTATIONS USED IN OUR FORMULATION.

Symbol	Description
T	The set of transistors.
t_i	The ith transistor.
$Nmos$	The set of NMOS.
$Pmos$	The set of PMOS.
x_i	Unsigned integer variable. Represents the x-coordinate of t_i.
f_i	0/1 variable. Indicates whether transistor t_i is flipped.
N	The set of nets.
n	The nth net.
V	The set of vertices.
V_c	The set of M0 vertices.
v	A vertex.
sv	The super vertex representing the higher layers (i.e., M1 and M2 layers).
E	The set of edges.
$e_{u,v}$	Edge connecting vertices v and u.
E_v	The set of edges connected to the vertex v.
d_e^v	-1/1 constant. Indicates the directional parameter for edge e connected to vertex v, d_e^v=-1 if e directs toward v and d_e^v=1 otherwise.
s_n^v	0/1 variable. Indicates whether v is a pin of net n.
v^n	0/1 variable. Indicates whether v is occupied by net n.
f_n^v	Signed integer variable. Represents the total flow of net n for v.
e^n	0/1 variable. Indicates whether e is occupied by net n.
f_n^e	Signed integer variable. Represents the flow of net n through edge e.
p_n^v	0/1 variable. Indicates whether v is used as a patch of net n.
m_r	Unsigned integer variable. Represents the estimated usage of M2 track at row r.
m_r^n	0/1 variable. Indicates whether one of the M2 tracks is used for net n at row r.

1) Transistor Placement Formulation: A placement result can be obtained by determining the variables x_i and f_i for each transistor t_i. Additionally, to mitigate the complexity of the subsequent routing problem, the pin positions for each net are defined by introducing the

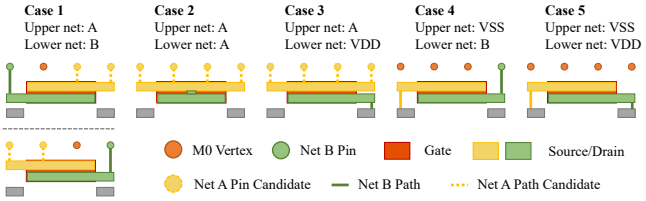


Fig. 6. Five distinct cases for pin selections.

s_n^v variables for each M0 vertex. The constraints associated with the transistor placement rules are outlined as follows, using NMOS as an example; the constraints for PMOS are identical.

Non-overlap constraints: Constraint (1) ensures that transistors do not occupy the same location.

$$\forall t_i, t_j \in Nmos, \quad \bigwedge_{i \neq j} (x_i \neq x_j)\,. \tag{1}$$

Diffusion sharing constraints: For two adjacent transistors, t_i (left) and t_j (right), we define the function $\text{Share}(t_i, t_j, f_i, f_j)$, where f_i and f_j denote the flip types of t_i and t_j, respectively. The function returns 1 if the diffusion region can be shared between t_i and t_j under the flip types f_i and f_j; otherwise, it returns 0.

$$(x_i + 1 = x_j) \rightarrow (\text{Share}(t_i, t_j, f_i, f_j) = 1)\,. \tag{2}$$

Diffusion jog constraints: For three transistors designated as t_i (left), t_j (middle), and t_k (right), with widths w_i, w_j, and w_k respectively. The following constraint must be satisfied for these transistors:

$$(x_i + 1 = x_j) \wedge (x_j + 1 = x_k) \rightarrow (w_j \geq w_i) \vee (w_j \geq w_k). \tag{3}$$

Pin position allocation: In scenarios where a transistor terminal must be connected through the routing layer, the terminal is extracted to the M0 layer, with the exit vertex designated as the pin for the corresponding net. Taking the active region as a reference, as illustrated in Figure 6, five distinct cases are identified, each requiring individual consideration. Let v_1, v_2, v_3, and v_4 represent the four candidate pins arranged from left to right on M0, and nets A and B representing the associated nets. In Case 5, extraction to M0 is unnecessary. The constraints for the remaining four cases are formulated as follows:

$$\begin{aligned}
&\forall n \in \{A, B\}, \quad (s_n^{v_1} + s_n^{v_2} + s_n^{v_3} + s_n^{v_4}) \leq 1,\\
&\text{For case 1: } \left((s_B^{v_1} = 1) \wedge (s_A^{v_3} + s_A^{v_4} = 1)\right) \vee \left((s_B^{v_4} = 1) \wedge (s_A^{v_1} + s_A^{v_2} = 1)\right),\\
&\text{For case 2 and case 3: } \quad (s_A^{v_1} + s_A^{v_2} + s_A^{v_3} + s_A^{v_4} = 1),\\
&\text{For case 4: } \quad (s_B^{v_4} = 1).
\end{aligned} \tag{4}$$

It is important to note that the constraints on pin selection for the gate in CFET follow the same principles as those applied in the active region.

2) Partial Routing Formulation: This section first presents a theorem and an accompanying lemma that ensures in-cell routability, followed by the formulation of partial routing.

In-Cell Routability Assurance: As illustrated in Figure 5(b), the M2 routing outcomes are governed by the usage of M1 tracks, which in turn depend on the usage of vias from M0 to M1 and the M0 routing result. Furthermore, based on the CFET architecture and observations from experimental results in [7] and [4], CFET cells generally exhibit minimal use of M2 resources. Accordingly, we have the following assumptions: (A1) The routing resources on the M2 layer are sufficient, and routing legalization on the M2 layer can be ensured. (A2) If a net requires multiple vias from M0 to M1, it will occupy an M2 track spanning the horizontal range between its leftmost and rightmost vias.

Given Assumptions A1 and A2, along with the two challenges discussed in Section II-B, we present the following theorem:

Theorem 1. *For a given placement, in-cell routability is guaranteed if the routing solution on the M0 layer satisfies the following two conditions: (C1) The M0 routing solution is legal. (C2) In this M0*

routing solution, I/O nets and nets that are not fully routed on M0 are linked to M1 tracks through vias. Additionally, the selection of vias and M1 tracks must comply with the VR rule, with each M1 track accommodating only one net.

Proof. We categorize the nets into three types based on Conditions C1 and C2:

1) Nets fully routed on M0: Since the M0 routing solution is legal (Condition C1), these nets are considered fully routed.
2) Nets occupying a single M1 track: These nets include I/O nets that are fully routed on M0, such as net B in Figure 5(b). Since each M1 track can accommodate only one net (Condition C2), legalization of the M1 routing is ensured, and these nets are regarded as fully routed.
3) Nets spanning multiple M1 tracks: These nets require M2 routing for connectivity. As an example, consider net A in Figure 5(b): this net should occupy an M2 track between r_1 and r_2 (Assumption A2), where r_1 and r_2 represent the horizontal coordinates of the leftmost and rightmost vias utilized by the net. Since sufficient M2 routing resources are available and M2 legalization is guaranteed (Assumption A1), complete routing for these nets is ensured.

Hence, by meeting Conditions C1 and C2, all nets achieve complete and legal routing, ensuring in-cell routability. This concludes the proof. □

As shown in Figure 5, the M0 routing and via edges in our model correspond directly to those in the cell architecture. Based on the preceding analysis and Theorem 1, we present the following lemma:

Lemma 2. *The in-cell routability of the placement can be guaranteed by satisfying the following three conditions during the partial routing in our model: (C3) Each pin is connected to the super vertex. (C4) Both the M0 routing and the selection of via edges comply with the design rules. (C5) The usage of M2 tracks, as estimated based on Assumption A2, does not exceed the available resources.*

Proof. Condition C5 reinforces Assumption A1, and by satisfying Conditions C3 and C4 in the model, Conditions C1 and C2 are fulfilled in the cell architecture, thereby ensuring the in-cell routability (Theorem 1). This concludes the proof. □

Note that Condition C3 tightens Condition C2. Our approach does not distinguish whether a net requires the use of higher routing layers, but explicitly mandates that each net must connect to the M1 layer.

Formulation: With notations provided in Table I, the partial routing formulation utilizes s_n^v as input to generate v^n and p_n^v for each vertex, and e^n for each edge, where p_n^v plays a pivotal role in the legalization.

To facilitate routing, we adopt the concepts of source, sink, flow, and commodity from the multi-commodity flow (MCF) problem, involving the transport of multiple commodities from sources to sinks. Constraint (5) defines the MCF formulation, with the definitions of f_n^v, f_n^e, E_v and d_e^v provided in Table I.

$$\forall n \in N, v \in V, \quad f_n^v = \sum_{e \in E_v} \left(f_n^e \times d_e^v\right). \tag{5}$$

Next, we present Constraint (6), which specifies the determination of e^n and v^n by f_n^e. The function Binary returns 0 if the input number is 0, and 1 otherwise.

$$\begin{aligned} \forall e \in E, n \in N, \quad & e^n = \text{Binary}\left(f_n^e\right), \\ \forall v \in V_c, n \in N, \quad & v^n = \text{Binary}\left(\sum_{e \in E_v} e^n\right). \end{aligned} \tag{6}$$

Constraint (7) defines the sink and sources for each net by determining the value of f_n^v for each vertex. To satisfy condition C3, each pin is designated as a source with a commodity of quantity 1, while the super vertex serves as the shared sink for all nets. This configuration ensures that commodities flow from the pins to the super vertex, thereby guaranteeing that each pin can reach the super vertex and satisfying Condition C3.

$$\begin{aligned} \forall n \in N, \quad & f_n^{sv} \le 0, \\ \forall n \in N, v \in V_c, \quad & f_n^v = s_n^v. \end{aligned} \tag{7}$$

The routing rules are formulated as constraints within the SMT solver, following a similar approach to that outlined in [4], to ensure the legality of M0 routing and via edge selection, thereby satisfying Condition C4.

To enhance the flexibility of legalization, the patch variable p_n^v is introduced. For example, as shown in Figure 7(a), when I/O net A has only one pin, it requires a connection solely to the super vertex. However, this configuration would violate the MAR rule, necessitating an extension on M0. Figure 7(b) shows the legalization result without the utilization of p_n^v based on the MCF formulation. In contrast, by incorporating p_n^v, we reduce unnecessary extensions, as illustrated in Figure 7(c).

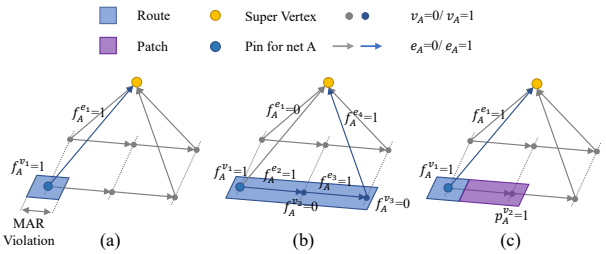


Fig. 7. (a) Partial routing result for net A without adherence to routing rules. (b) Legalization for net A without the utilization of p_n^v. (c) Legalization for net A with the incorporation of p_n^v.

The methodology for calculating the estimated usage of M2 tracks is described in Constraint (8), where LR_n and RR_n denote the leftmost and rightmost horizontal coordinates of the via edges utilized by net n. The definitions of m_r and m_r^n are provided in Table I. The function InRange returns 1 when the input r falls within the range defined by LR_n and RR_n; otherwise, it returns 0.

$$\begin{aligned} m_r^n &= \text{InRange}\left(r, LR_n, RR_n\right), \\ m_r &= \sum_{n \in N} m_r^n. \end{aligned} \tag{8}$$

Based on Constraint (8), the constraint required to satisfy Condition C5 is derived and presented in Constraint (9), where H_n denotes the horizontal track number.

$$m_r \le H_n. \tag{9}$$

Thus, it has been ensured that the partial routing solution satisfies Conditions C3, C4, and C5, thereby guaranteeing the in-cell routability (Lemma 2). Our routability-guaranteed placement method significantly reduces the scale of modeling and formulation compared to the approach in [4]. For instance, for a placement with width k, the model established in [4] contains $(20k-8)$ vertices and $(27k-16)$ edges, while our method results in $(8k-3)$ vertices and $(12k-8)$ edges. Furthermore, for a net with l pins in the aforementioned placement, the formulation in [4] requires distinguishing between different commodities, leading to $(l-1) \times (47k-24)$ flow variables associated with that net. In contrast, our method does not require distinguishing between different commodities for the same net, allowing each vertex and edge to use a single flow variable, resulting in a total of $(20k-11)$ flow variables.

3) Lexicographic Multi-Objective Optimization: The framework incorporates multiple optimization objectives during the placement stage. The primary optimization objective is the cell width, as detailed in Expression (10). X_r represents the horizontal coordinate value of the rightmost transistor.

$$\text{Min.} \quad X_r. \tag{10}$$

If the partitioning method is applied, we modify Expression (10) to Expression (11). C_w represents the width correction term which is assigned -1 if the placement result from the current iteration allows diffusion sharing with the subsequent partition, and 0 otherwise.

$$\text{Min.} \quad X_r + C_w. \tag{11}$$

The second objective is the estimated usage of M2 tracks , as detailed in Expression (12). $RLimit$ represents the upper bound of the horizontal coordinate r, which is determined by the total number of transistors.

$$\text{Min.} \sum_{r \in [0, RLimit]} m_r. \tag{12}$$

The third optimization objective is the half-perimeter wire length (HPWL) defined by the positions of the pins for each net , as detailed in Expression (13). $MaxR_n$, $MinR_n$, $MaxC_n$, and $MinC_n$ denote the values of r and c corresponding to the leftmost, rightmost, topmost, and bottommost pins of net n, respectively.

$$\text{Min.} \sum_{n \in N} (MaxR_n - MinR_n + MaxC_n - MinC_n). \tag{13}$$

These three optimization objectives are integrated into the overall formulation by employing a lexicographic multi-objective optimization approach.

C. Progressive Metal Routing with Placement-Specified Pin Positions

After obtaining the placement result and the pin positions for each net, we perform a progressive metal routing method that includes a manual-inspired M0 routing followed by an ILP-based M1 and M2 routing. Figure 8 illustrates our routing method.

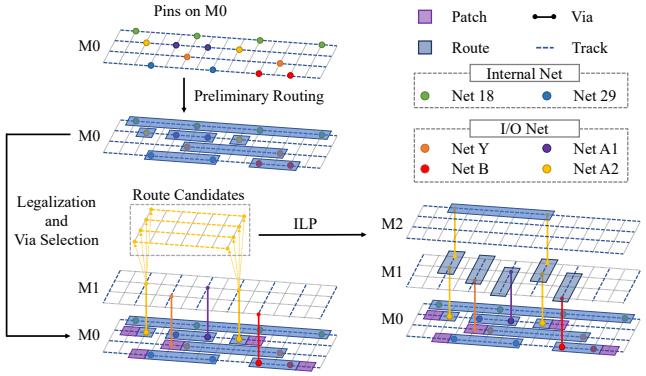


Fig. 8. Illustration of our routing method using the AOI21x1 as an example.

1) Manual-inspired M0 Routing: The M0 routing process is inspired by manual routing. Initially, a preliminary routing step is performed for each net, establishing direct connections between its pins along the M0 tracks, provided they are unobstructed by pins from other nets. Then, the M0 routing undergoes a legalization step, where segments requiring extension to comply with routing rules are lengthened using patches. During this step, appropriate via positions are determined for pins that need to connect to M1, with the objective of minimizing the distance between vias when multiple vias are required for a net.

2) ILP-based M1 and M2 Routing: After completing the M0 routing, the via positions connecting M0 to M1 for each net are treated as pin positions on M1. For nets with multiple pins on M1, the nets are partitioned into several 2-pin subnets, and each subnet presents four potential routing paths, as depicted in the "Route Candidates" of Figure 8. An ILP formulation is then employed to select an appropriate routing path for each subnet while conducting the legalization process.

IV. Experimental Results

Our algorithm was implemented in Python and executed on a workstation with Intel i7-12700H 2.70GHz CPU and 32GB memory. The SMT solver Z3 [3] (version 4.13.0) was employed to obtain the optimal solution for the SMT-based routability-guaranteed placement formulation presented in Section III-B and the ILP problem detailed in Section III-C2.

Besides, the same as in [4] and [7], we used the SPICE netlists of ASAP 7nm standard cells from [5] as the input netlists for synthesizing CFET cells. To ensure a fair comparison, we followed the same placement and routing rules[1] as those used in [4] and [7], with NMOS stacked on PMOS in the placement.

TABLE II
Comparison of the CFET cells generated by DAC-2024 [7], TVLSI-2021 [4] and our algorithm. The best results are highlighted in BLUE based on a prioritized order of Cell Width, M2 Usage, and Metal Length.

Cell Netlist		Cell Width (#CPP)			M2 Usage			Metal Length			Runtime (sec.)		
Name	#FET	[7]	[4]	Ours	[7]	[4]	Ours	[7]	[4]	Ours	[7]	[4]	Ours
AND2x2	6	**6**	**6**	**6**	**0**	**0**	**0**	**56**	60	**56**	**0.5**	8.1	5.3
AND3x1	8	**6**	**6**	**6**	**0**	**0**	**0**	**63**	68	**63**	**1.4**	30.1	8.8
AND3x2	8	**7**	**7**	**7**	**0**	**0**	**0**	**69**	76	**69**	**1.6**	28.4	10.9
AOI22x1	8	**11**	**11**	**11**	1	1	**0**	203	240	**176**	**20.5**	363.2	131.3
BUFx2	4	**5**	**5**	**5**	**0**	**0**	**0**	**39**	40	**39**	**0.2**	4.9	3.2
BUFx3	4	**6**	**6**	**6**	**0**	**0**	**0**	56	53	**52**	**0.6**	11.2	4.2
BUFx4	4	**7**	**7**	**7**	**0**	**0**	**0**	62	59	**58**	**0.8**	7.9	5
BUFx8	4	**12**	**12**	**12**	**0**	**0**	**0**	107	105	**103**	**4.4**	43.7	11.5
DFFHQN	24	**15**	**16**	**15**	1	**0**	**0**	203	182	**172**	410.3	6831.8	**93**
FA	24	**14**	**14**	**14**	3	**2**	**2**	276	379	**249**	484.6	6653.1	**191.8**
INVx1	2	**3**	**3**	**3**	**0**	**0**	**0**	**22**	23	**22**	**0.1**	0.5	1.3
INVx2	2	**4**	**4**	**4**	**0**	**0**	**0**	**27**	29	**27**	**0.1**	1	1.8
INVx4	2	**6**	**6**	**6**	**0**	**0**	**0**	47	48	**46**	**0.6**	3.5	3
INVx8	2	**10**	**10**	**10**	**0**	**0**	**0**	111	92	**86**	**3.4**	19.1	6.7
NAND2x1	4	**6**	**6**	**6**	**0**	**0**	**0**	70	74	**65**	**1.3**	15.9	5
NAND2x2	4	**10**	**10**	**10**	**0**	**0**	**0**	125	131	**118**	**5.3**	33.8	13.8
NAND3x1	6	**11**	**11**	**11**	**0**	**0**	**0**	147	146	**142**	**12.2**	124.1	15.2
NOR2x1	4	**6**	**6**	**6**	**0**	**0**	**0**	70	74	**66**	**1.3**	12.9	4.7
NOR2x2	4	**10**	**10**	**10**	**0**	**0**	**0**	119	131	**118**	**5**	27.9	31.9
NOR3x1	6	**11**	**11**	**11**	**0**	**0**	**0**	163	156	**138**	**11.1**	52.3	17.9
OR2x2	6	**6**	**6**	**6**	**0**	**0**	**0**	**56**	60	**56**	**0.5**	13	5.5
OR3x1	8	**6**	**6**	**6**	**0**	**0**	**0**	**63**	68	**63**	**1.5**	76.8	10.3
OR3x2	8	**7**	**7**	**7**	**0**	**0**	**0**	**69**	76	**69**	**1.7**	89.2	13.7
Avg.1†	6.61	**8.04**	8.09	**8.04**	0.22	0.13	**0.087**	96.65	103.04	**89.26**	42.1	628.4	**25.9**
Norm.1†		**1.00**	1.00	**1.00**	2.44	1.44	**1.00**	1.08	1.15	**1.00**	1.63	24.26	**1.00**
AOI21x1	6	9	9	**8**	**0**	**0**	1	**122**	142	129	**6.3**	119.7	19
NAND3x2	6	21	21	**20**	**0**	**0**	1	303	**283**	308	**81.9**	2869.5	641.9
NOR3x2	6	21	21	**20**	1	**0**	1	320	**286**	310	**82.3**	1897.5	483.7
OAI21x1	6	9	9	**8**	**0**	**0**	1	**131**	149	139	**7.9**	52.5	18.2
OAI22x1	8	11	11	**10**	**0**	1	2	**174**	255	208	**19.3**	612.6	89.2
XNOR2x1	10	11	11	**10**	**1**	**1**	2	180	223	**170**	**47.5**	977	58.9
XOR2x1	10	11	11	**10**	**1**	**1**	2	175	213	**170**	**50.4**	134.9	84.5
Avg.2†	6.8	9.27	9.3	**9.03**	0.27	**0.2**	0.4	120.93	130.7	**116.23**	**42.15**	703.9	66.4
Norm.2†	–	1.03	1.03	**1.00**	0.68	**0.5**	1.00	1.04	1.12	**1.00**	**0.63**	10.6	1.00

† Avg.1 and Norm.1 are calculated after excluding the 7 cells that our algorithm achieves smaller widths, while Avg.2 and Norm.2 include all cells.

Table II compares the layouts generated by our algorithm with those produced by the state-of-the-art CFET cell generators, DAC-2024 [7] and TVLSI-2021 [4]. In the table, the column "#FET" represents the number of transistors, and "Cell Width" provides the width of the generated cell. "Metal Length" is calculated as the weighted sum of vias and metal grids, following the method used in [7] and [4], where vias are weighted $4\times$ compared to metal grids to account for parasitic resistance. "M2 Usage" indicates the usage of the M2 tracks, and "Runtime" shows the runtimes in seconds. Additionally, since the codes of [7] and [4] are not available for us, the experimental results of [7] and [4] are quoted from the respective publications directly.

As shown in Table II, the layouts generated by our algorithm achieve an average reduction of 3% in cell width compared to those in [7] and [4]. Notably, our algorithm achieves the smallest cell width for all tested cells, with 7 out of 30 cells exhibiting smaller widths. For the remaining 23 cells, which have the same cell width as those in [7] and [4], our algorithm achieves the best M2 usage and total metal length. Besides, although the runtime in Table II is just for reference due to the differences in machines, our algorithm should still run much faster than [4]. This demonstrates the effectiveness of our routability-guaranteed placement formulation, as well as the efficiency of our progressive metal

[1]MAR/EOL/VR/PRL/SHR/MPL=1/2/1/1/2/2

TABLE III
RE-GENERATED CFET LAYOUTS BY OUR GENERATOR AFTER ADJUSTING OPTIMIZATION OBJECTIVES.

Cell Netlist	Cell Width			M2 Usage			Metal Length		
Name	[7]	[4]	Ours	[7]	[4]	Ours	[7]	[4]	Ours
AOI21x1	**9**	**9**	**9**	**0**	**0**	**0**	122	142	**118**
NAND3x2	**21**	**21**	**21**	**0**	**0**	**0**	303	283	**272**
NOR3x2	**21**	**21**	**21**	1	**0**	**0**	320	**286**	288
OAI21x1	**9**	**9**	**9**	**0**	**0**	**0**	131	149	**127**
OAI22x1	**11**	**11**	**11**	**0**	1	**0**	**174**	255	175
XNOR2x1	**11**	**11**	**11**	**1**	**1**	**1**	180	223	**176**
XOR2x1	**11**	**11**	**11**	**1**	**1**	**1**	**175**	213	**175**
Avg.	**13.29**	**13.29**	**13.29**	0.43	0.43	**0.29**	200.71	221.57	**190.14**
Norm.	**1.00**	**1.00**	**1.00**	1.48	1.48	**1.00**	1.05	1.17	**1.00**

TABLE IV
COMPARISON OF LARGER CFET CELLS GENERATED BY DAC-2024 [7] AND OUR GENERATOR.

Cell Netlist		Cell Width		M2 Usage		Metal Length		Runtime(sec.)	
Name	#FET	[7]	Ours	[7]	Ours	[7]	Ours	[7]	Ours
DHLx1	16	**11**	**11**	**0**	**0**	117	**115**	441	**210.71**
DHLx2	16	**12**	**12**	**0**	**0**	**117**	**117**	**121**	169.17
DHLx3	16	**13**	**13**	**0**	**0**	146	**134**	**131**	136.29
DFFHQNx2	24	**16**	**16**	1	**0**	212	**174**	420	**98.46**
DFFHQNx3	24	**17**	**17**	1	**0**	227	**191**	502	**103.07**
SDFHx4	32	N/R	24	N/R	1	N/R	337	N/R	1217.14
SDFLx4	32	N/R	24	N/R	1	N/R	338	N/R	1195.4
Avg.†	19.2	**13.8**	**13.8**	0.4	**0**	163.8	**146.2**	323	**143.54**
Norm.†		**1.00**	**1.00**			1.12	**1.00**	2.25	**1.00**

† The calculations for Avg. and Norm. exclude the two cells, SDFHx4 and SDFLx4, which are not reported (N/R) in [7].

routing with placement-specified pin positions. Although the overall runtime of our algorithm may be higher than that of [7], it remains reasonable and worthwhile given our algorithm's superiority in metrics such as cell width. Furthermore, for the two largest cells, $DFFHQN$ and FA, we employed the partitioning method discussed in Section III-A2, resulting in a runtime shorter than that of [7]. This further illustrates the effectiveness of our preprocessing strategy for managing large circuits.

For the 7 cells that our algorithm achieves smaller widths, the M2 usage and metal length in some layouts produced by our algorithm exceeds those of [7] and [4]. This is due to the fact that the reduced cell widths necessitate additional long-distance M2 tracks to accommodate the smaller cell widths. To validate this perspective, we combined Expression (10) and Expression (12) by applying weighted parameters and re-experimented on these 7 cells, which led to improvements in both metal length and M2 usage compared to [7] and [4], as shown in Table III. Nevertheless, it is important to note that optimizing cell width remains a primary concern in standard cell layout generation. Therefore, compromising other objectives to achieve smaller cell widths is justifiable, provided the trade-off stays within an acceptable range.

To further assess the robustness and scalability of our algorithm on larger cells, we conducted experiments similar to those in [7]. Table IV compares the results, showing that our algorithm produces better layouts with smaller M2 usage and metal length, while maintaining the same cell width and achieving comparable runtime.

Finally, Figure 9 provides a comparison of the layouts generated by [7] and our algorithm before and after adjusting objectives. The layout produced by our algorithm before objective adjustment demonstrates a smaller cell width, while the layout generated after the adjustment shows notable reductions in both M2 usage and metal length. This highlights the flexibility and effectiveness of our algorithm.

V. CONCLUSIONS

In this paper, we have presented a comprehensive placement and routing framework for synthesizing CFET cells. We first presented a partitioning technique followed by a heuristic quality maintenance strategy for large-scale cells to ensure scalability and efficiency. Then, we proposed a novel SMT-based placement method incorporating partial routing to achieve minimum-width placement while ensuring in-cell routability. Finally, we proposed a progressive metal routing method

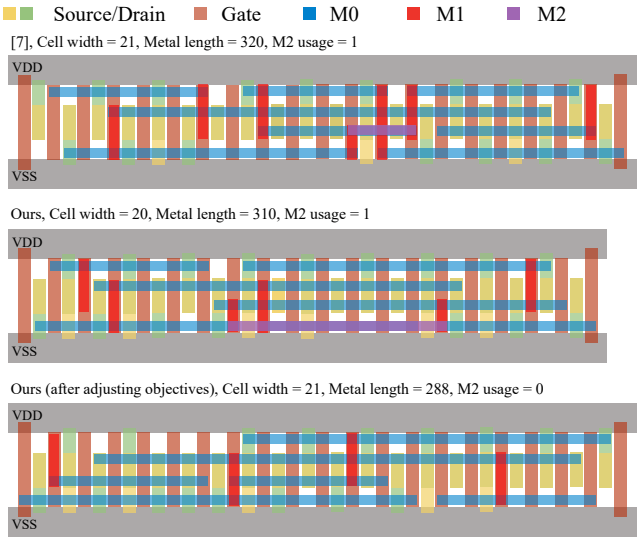


Fig. 9. Comparison of the layout generated by [7], our algorithm and our algorithm after adjusting objectives for netlist NOR3x2.

that includes a manual-inspired M0 routing followed by an ILP-based M1 and M2 routing. Experimental results have shown that our algorithm achieves the smallest cell width for all tested cells, with 7 out of 30 cells exhibiting smaller widths. For the remaining 23 cells, which have the same cell width as those in other generators, our algorithm achieves the smallest M2 usage and total metal length.

REFERENCES

[1] G. Bae et al. 3nm GAA technology featuring multi-bridge-channel FET for low power and high performance applications. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 28.7.1–28.7.4, 2018.

[2] M. G. Bardon et al. Dimensioning for power and performance under 10nm: The limits of FinFETs scaling. In *2015 International Conference on IC Design & Technology (ICICDT)*, pages 1–4, 2015.

[3] N. Bjørner, A.-D. Phan, and L. Fleckenstein. νz - an optimizing SMT solver. In C. Baier and C. Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 194–199, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[4] C.-K. Cheng et al. Complementary-FET (CFET) standard cell synthesis framework for design and system technology co-optimization using SMT. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(6):1178–1191, 2021.

[5] L. T. Clark et al. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

[6] K. Jo and T. Kim. Optimal transistor placement combined with global in-cell routing in standard cell layout synthesis. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 517–524, 2021.

[7] S. Kim and T. Kim. Optimal transistor folding and placement for synthesizing standard cells of complementary FET technology. In *2024 61st ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2024.

[8] D. Lee et al. SP&R: SMT-based simultaneous place-and-route for standard cell synthesis of advanced nodes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(10):2142–2155, 2021.

[9] N. Loubet et al. Stacked nanosheet gate-all-around transistor to enable scaling beyond FinFET. In *2017 Symposium on VLSI Technology*, pages T230–T231, 2017.

[10] E. Park and T. Song. Complementary FET (CFET) standard cell design for low parasitics and its impact on VLSI prediction at 3-nm process. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(2):177–187, 2023.

[11] D. Prasad et al. Buried power rails and back-side power grids: Arm® CPU power delivery network design beyond 5nm. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 19.1.1–19.1.4, 2019.

[12] J. Ryckaert et al. The complementary FET (CFET) for CMOS scaling beyond n3. In *2018 IEEE Symposium on VLSI Technology*, pages 141–142, 2018.

[13] S. M. Y. Sherazi et al. CFET standard-cell design down to 3track height for node 3nm and below. In *Advanced Lithography*, 2019.

[14] S. Subramanian et al. First monolithic integration of 3D complementary FET (CFET) on 300mm wafers. In *2020 IEEE Symposium on VLSI Technology*, pages 1–2, 2020.

[15] H.-S. P. Wong. Beyond the conventional transistor. *IBM Journal of Research and Development*, 46(2.3):133–168, 2002.

Weighted Range-Constrained Ising-Model Decoder for Quantum Error Correction

Xinyi Guo
Graduate School of Informatics
Kyoto University
Kyoto 606-8501, Japan
guoxinyi@fuji.waseda.jp

Hiromitsu Awano
Graduate School of Informatics
Kyoto University
Kyoto 606-8501, Japan
awano@i.kyoto-u.ac.jp

Takashi Sato
Graduate School of Informatics
Kyoto University
Kyoto 606-8501, Japan
takashi@i.kyoto-u.ac.jp

***Abstract*—Ising model-based Quantum Error Correction decoders reduce topological complexity compared to classical decoders. However, the SOTA Ising decoder has a higher time complexity than union-find (UF) and a lower threshold than minimum-weight perfect-matching (MWPM). We propose the Weighted Range-Constrained Ising Model-Based (WRIM) decoder. WRIM uses a polygonal region to enclose flipped syndromes, ensuring the coverage of all potential error chains while optimizing coupling and external field coefficients. WRIM reduces the variable count by 97.8x, achieves microsecond-level decoding, and has a worst-case time complexity of $O(n)$, outperforming UF. WRIM exhibits threshold behavior up to 10.7-11.0%, surpassing the MWPM's highest reported threshold.**

***Index Terms*—Quantum error correction, Ising model, fast decoding method, surface code, variable reduction.**

I. Introduction

Quantum computers introduce a new computing paradigm with the potential to solve problems that are intractable in polynomial time on classical systems [1], [2]. One of the main barriers to realize practical quantum computers is decoherence—a loss of quantum coherence that occurs when qubits interact with their environment [3], [4]. Therefore, achieving reliable quantum computation requires fault-tolerant quantum computing (FTQC), with quantum error correction (QEC) as its cornerstone to detect and correct errors during computations [5], [6].

In QEC, multiple physical qubits are combined to form a single logical qubit, thereby enhancing resistance to quantum errors. This method of constructing logical qubits is called a "code," and various methods have been proposed. Among them, Kitaev's surface code [7] and its variants [8], [9] are considered the leading candidates for realizing QEC due to their superior quantum error resilience and compatibility with integrated qubits, as they only require gate operations with neighboring qubits. The surface code utilizes two types of qubits: data qubits for information storage and ancilla qubits for error detection.

In the surface code, "decoding" is necessary to estimate the data qubits where errors have occurred based on the measurements of the ancilla qubits (referred to as syndromes). Effective QEC decoders must operate fast enough to prevent error accumulation [10] (e.g., on a microsecond timescale for superconducting transmon qubits [11], [12] and on a millisecond timescale for trapped ions [13]) and achieve target logical error rates.

Numerous QEC decoders have been developed, such as Minimum-Weight Perfect Matching (MWPM) decoders [14]–[16], which transform the decoding problem into a graph matching problem. However, MWPM decoders have a worst-case time complexity between $O(n^3)$ and $O(n^7)$ relative to the number of data qubits n, limiting the code distances d at most 9 when adhering to real-time latency constraints [15], and up to 33 with fast MWPM when assuming a circuit-level noise of 0.1% [17]. Additionally, these decoders demand substantial memory resources [18]. The tensor network decoder [19] reformulates the decoding problem as a machine learning problem and solves the decoding problem with a time complexity of $O(n^2)$. Neural network (NN) decoders [20]–[22] also show promise, though training a suitable NN for QEC demands substantial computational resources and time. Currently, the fastest decoder is the Union-Find (UF) decoder [23], [24], which leverages the union-find data structure and has a worst-case runtime of $O(n\alpha(n))$ ($\alpha(n) \leq 3$). However, when implemented on Systolic Arrays, it is constrained to code distances due to limited memory and processing capacity, handling up to $d = 15$ [25].

Recently, an Ising model-based decoder was introduced to address these challenges [26], [27], framing QEC decoding as an Ising model task solvable by annealing machines [28], with scalability up to a code distance of 46. However, despite its polynomial computational scaling slightly better than MWPM decoders, its decoding time remains suboptimal.

This paper introduces the Weighted Range-Constrained Ising Model-Based (WRIM) decoder, a fast Ising model-based approach for efficient QEC. The WRIM decoder employs a polygonal region to enclose flipped syndromes with their neighbors, aiming to limit the number of nodes while ensuring the connectivity of all flipped syndromes. By limiting qubits in this region, the Ising model for error decoding is formulated to capture all potential error chains and possible error sites. It then assigns optimal weights to the Ising model's coupling coefficient and external field coefficient based on the satisfaction of syndrome parity for each error pattern, aiming to minimize the length of each error chain.

This paper presents the following novel contributions.

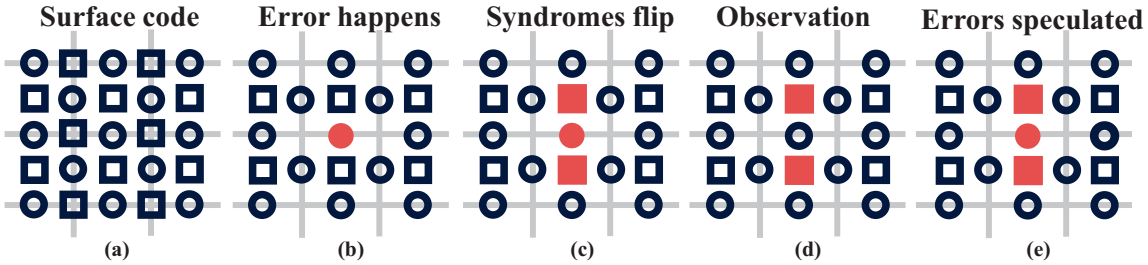


Fig. 1. (a) Planar surface code with code distance $d = 3$. Along the gray lines representing the surface code, open circles indicate data qubits, while squares denote X and Z ancilla qubits. X ancilla qubits are positioned on the lattice grid, while Z ancilla qubits are positioned between the grids. (b)–(e) depict the cases with only X errors. An X error on a data qubit is represented as a red-filled circle (b), flipping the syndrome from $+1$ to -1, thereby turning the open blue squares into filled red squares (c). By analyzing these Z syndromes measured by the Z ancilla qubits (d), the location of X error can be identified (e).

- The WRIM decoder achieves up to a 97.8x reduction in the number of variables compared to the state-of-the-art (SOTA) Ising model-based decoder [26], significantly decreasing computational resource demands.
- The WRIM decoder's region formation has a time complexity of $O(kn)$ with $k \leq 1$, and the worst-case complexity is $O(n)$. The WRIM decoder's Ising decoder has a time complexity of $O(n^a)$, with $a \leq 1$. Specifically, at a 20% physical error rate, its complexity is $O(n^{0.97})$, which represents the worst-case scenario in practice. This results in an overall $O(n)$ complexity in WRIM, which is lower than $O(n \log n)$ complexity of the fastest SOTA decoder, UF with a worst-case complexity of $O(n\alpha(n))$ where $\alpha(n) \leq 3$. Experiments on existing quantum annealers [29] at 0.1% and 1% physical error rates demonstrated decoding time in the order of microseconds.
- By retaining nearly all error chains and adaptively assigning Ising-weights for each error pattern, WRIM achieves a logical error threshold of 10.7-11.0%, exceeding the highest threshold of 10.3% achieved by the MWPM decoder, demonstrating better fault tolerance.

In addition to these contributions, it is important to note that quantum decoding, particularly in QEC, poses significant circuit design challenges. Addressing these challenges via techniques in electric design automation is crucial for advancing quantum computing and unlocking its potential to surpass classical computation.

The rest of the paper is organized as follows: Sec. II provides a brief introduction to surface codes and the mapping of the QEC decoding to the Ising model. Sec. III details the proposed WRIM decoder. Sec. IV presents the experimental results, including the number of variables needed for WRIM decoder, decoding time, and logical error rate threshold. Sec. V concludes the paper.

II. Surface Code and Ising Model

A. Surface Code and Quantum Error Correction

This paper focuses on the planar surface code [7], [8], widely regarded for its practical implementation potential. However, the proposed decoder is also applicable to other surface codes, such as the rotated surface code [30], [31] and the color code [32].

In QEC, as depicted in Fig. 1 (a), the linear length of the lattice defines the code distance. The lattice comprises two qubit types: data qubits (circles on lattice edges), and ancilla qubits (squares). These qubits are prone to errors induced by various noise sources, which can result in bit-flip errors (X errors) or phase-flip errors (Z errors). Ancilla qubits are placed to detect errors in data qubits without directly observing data qubits. There are two types of ancilla qubits: X ancilla qubits, located on the lattice grid, detect Z errors, while Z ancilla qubits, positioned between grids of the lattice, are designed to detect X errors.

Syndromes represent the collective logical state of data qubits as measured by ancilla qubits. Syndrome $+1$ indicates there is no error or an even number of errors, while -1 reflects an odd number of errors in the surrounding data qubits. As shown in Fig. 1 (b)–(e), an X error on a data qubit changes its representation from an open circle (blue) to a filled circle (red). This flips syndrome from $+1$ to -1, altering the corresponding ancilla qubit's symbol changes from an open square (blue) to a filled square (red). The QEC decoder deduces error locations from these syndrome observations.

For a planar surface code with code distance d, the number of data qubits is $d^2 + (d-1)^2$, while the number of X and Z ancilla qubits is both $d \times (d+1)$. This arrangement ensures a Hilbert space dimension of 2, which represents a logical qubit [33]. Therefore, the Z ancilla qubits in the top and bottom rows, as well as X ancilla qubits in the left and right columns, are each connected to three data qubits.

For simplicity, X and Z errors are assumed to occur independently with probability p, allowing them to be corrected separately. In this paper, we focus solely on X errors detected by Z ancilla qubits. The treatment of Z errors is entirely analogous, as rotating the surface code by 90 degrees swaps the positions of X ancilla qubits with those of Z ancilla qubits.

B. Error Pattern

An error in a data qubit flips two adjacent syndromes, as illustrated in Fig. 2 (a). If a connected syndrome does not flip (Fig. 2 (b)), it indicates errors in an even-numbered of

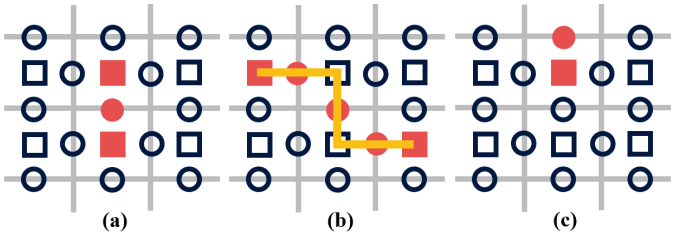


Fig. 2. Error patterns. (a) An error can flip two connected syndromes. (b) Neighboring errors generate an error chain, and the syndrome flips at both ends of the error chain. (c) The error in the first and last rows of the surface code will only flip one syndrome.

connected data qubits, forming an error chain. These chains cause the syndromes at their endpoints to flip. Any error chain connecting the flipped syndromes (without crossing the square) is a valid solution. A special case, known as a boundary condition, occurs when errors in data qubits on the first or last row flip only one syndrome (Fig. 2 (c)). In such cases, syndromes can form chains with other syndromes or the boundary.

C. Ising Model for Quantum Error Correction

The Ising model-based decoder [26] transforms the QEC into the following Ising model:

$$H = -J \sum_{v}^{N_v} b_v \prod_{i \in \delta v}^{4 \text{ or } 3} \sigma_i - h \sum_{i}^{N_d} \sigma_i.$$

Data qubits are represented by variables σ_i, which flip from $+1$ to -1 during the annealing process if an error occurs. An ancilla qubit is denoted by v, with δv representing the list of connected qubits. The syndrome b_v is $+1$ for even or no errors and -1 for odd errors. N_v and N_d represent the numbers of syndromes and data qubits, respectively.

The first term, with coupling coefficient J, ensures that the decoded error pattern aligns with the syndromes for accurate correction. The second term, with external field coefficient h, minimizes the total errors by favoring solutions with fewer flips. Setting $J > h$ guarantees the correctness of the solution. Thus, the lowest energy state of the Ising model represents an error pattern that fits the syndromes while minimizing the total number of errors.

III. Weighted Range-Constrained Ising Model-Based Decoder

Unlike the Ising model-based decoder [26], which includes all syndromes and qubits across the entire surface code, the WRIM decoder defines a polygonal region around flipped syndromes. This region ensures the shortest error chains between flipped syndromes and encompasses potential error chains meeting boundary conditions, as shown in Fig. 2 (c).

The WRIM decoder operates in three stages. First, it determines allowable row and column ranges for flipped syndromes in the surface code. Second, it constructs an Ising model using syndromes and variables within this region. Finally, multiple weights for coupling and external field coefficients are applied to the annealing machine. The solution with the largest external field coefficient satisfying syndrome parity identifies the most likely error locations.

Algorithm 1: Range-Constrained Polygon Region Formation

Input: Flipped syndrome set S_F
Output: Dictionaries Boundary and Row recording the region.

1 Create three dictionaries Row, Col and Boundary and one set Non_empty.
2 **foreach** S_F **do**
3 Update Row[row_i] and Col[col_j] ;
4 Row[row_i] = [min(col_i), max(col_i)] ;
5 Non_empty add row_i ;
6 Col[col_j] = [min(row_j), max(row_j)] ;
7 **if** min(row_j) $\neq$ max(row_j) **then**
8 Boundary[col_j] = $[0, 2d-1]$;
9 **else**
10 **if** min(row_j) + max(row_j) $< 2d-1$ **then**
11 Boundary[col_j] = [0, min(row_j)] ;
12 **else**
13 Boundary[col_j] = [max(row_j), $2d-1$] ;
14 **for** row $\in$ [min Row.keys(), max Row.keys()] **do**
15 **if** row $\notin$ Non_empty **then**
16 Find nearest upper row up_row
17 and lower row lo_row in Non_empty ;
18 min_inter = max(Row[up_row][0], Row[lo_row][0]) ;
19 max_inter = min(Row[up_row][1], Row[lo_row][1]) ;
20 **if** max_inter - min_inter $\geq$ *2* **then**
21 Row[row] = [min_inter, max_inter] ;
22 **else**
23 min_union = min(Row[up_row][0], Row[lo_row][0]) ;
24 max_union = max(Row[up_row][1], Row[lo_row][1]) ;
25 Row[row] = [min_union, max_union] ;
26 Row[up_row] = [min_union, max_union] ;
27 Row[lo_row] = [min_union, max_union] ;

A. Range-Constrained Polygon Region Formation

The process of forming the range-constrained region is detailed in Algorithm 1 and illustrated in Fig 3. In steps 2–6, each flipped syndrome updates row and column dictionaries to define the range of columns for each row and rows for each column, encompassing all flipped syndromes. Steps 7–13 handle boundary conditions: if a column includes a single flipped syndrome, the closer boundary is connected; if it includes two or more, both upper and lower boundaries are linked. Steps 14–27 ensure row connectivity by linking flipped syndromes across rows. If two rows have at least two columns, the path

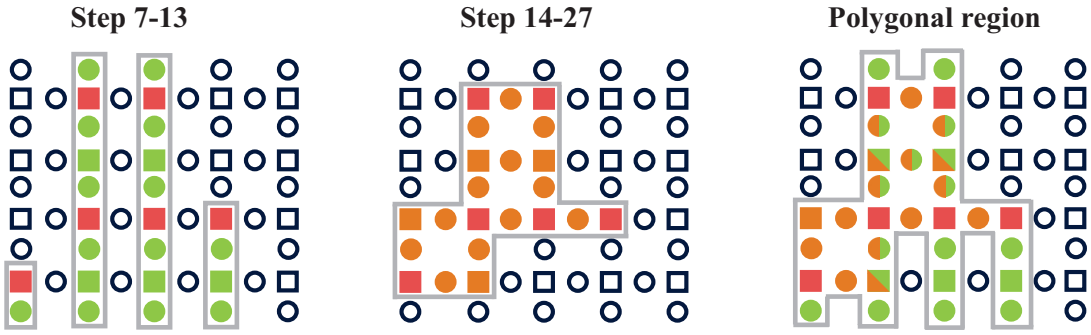


Fig. 3. Steps 7-13 include the non-flipped syndromes and qubits located between flipped syndromes and boundaries, represented as green-filled squares and circles. Steps 14–27 add non-flipped syndromes and qubits forming paths between flipped syndromes across rows, represented as orange-filled squares and circles. In the polygonal region, squares and circles in both colors indicate areas where non-flipped syndromes and qubits are included during both steps.

TABLE I
THE WEIGHT SETTING WITH DIFFERENT LENGTH OF ERROR CHAIN (J = 1024)

Length	1–2	3–4	5–6	7–8	>8
h	900	400	260	20	1

width is set to the intersection; otherwise, the row ranges are merged to determine the maximum range, maintaining nearly all paths.

The resulting polygonal region includes the error chains between flipped syndromes and satisfies boundary conditions, ensuring it captures the necessary and sufficient information to locate errors accurately.

B. Range-Constrained Ising Model-Based (RIM) Decoder

After determining the polygonal region, both syndromes and qubits within the regions are configured to form the Ising model. All syndromes are represented as b_v, while the connected data qubits are designated as σ. The Ising model is constructed as:

$$H = -J \sum_{v \in \text{region}} b_v \prod_{i \in \delta v}^{1\sim 4} \sigma_i - h \sum_{i \in \text{region}} \sigma_i$$

The value of J must ensure that the error pattern matches the expected error syndrome, while h aims to reduce the number of errors.

In this RIM decoder, not only are syndromes and qubits outside the region excluded, but the Ising model no longer requires new variables to reduce the order of products involving four qubit variables. For an Ising machine, handling terms beyond quadratic is impractical, necessitating the introduction of new variables to reduce higher-order terms. For example, a cubic term $x_1x_2x_3$ is transformed into quadratic terms using a penalty function: $x_1x_2x_3 = yx_3 + P \cdot (x_1x_2 - 2x_1y - 2x_2y + 3y)$ [34]. Here, y is the newly introduced variable and P is the penalty value that enforces $y = x_1x_2$.

C. Weighted RIM (WRIM) Decoder

For an error chain of length l, the change in Hamiltonian energy caused by flipping the corresponding variable must be negative, i.e., $-2J + lh < 0$ [26]. By appropriately setting h and J, the Hamiltonian energy landscape can be optimized for efficient solution finding. For error chains of length less than or equal to 2, J is set slightly greater than h, creating a distinct energy gap between global and local minima. For the longest error chain of length 3–4, J should be adjusted to slightly exceed $2h$. Table I provides parameter settings of h for various chain lengths with J fixed at 1024.

If the longest error chain in the decoded error pattern exceeds the length used for weight settings (J and h coefficients), the annealing solution may fail to satisfy syndrome parity. To address this, the Weighted RIM (WRIM) decoder applies the Ising model simultaneously across all weight settings in Table I. Models with larger values of h solve faster. If a valid solution satisfying syndrome parity is found under such a model, it is accepted, and further searches with different weights are halted. Otherwise, the process continues until a suitable solution is identified.

IV. NUMERICAL EXPERIMENTS

The experiments were conducted using Python 3.10.12 on a classical computer with a Core i9-13900KF CPU and 125 GB of memory for Ising model construction. The model was solved using two methods: simulated quantum annealing (SQA) using a GPU annealing cloud server—Amplify Annealing Engine (AE) [35] and real quantum annealing (QA) via the D-Wave cloud server [29]. In AE, the annealing time determines the duration of each process, with numerous annealing iterations performed to locate the lowest energy state. Increased problem complexity and variable count increases result in longer iteration times. After annealing on the cloud server, solutions are returned to the classical computer. D-Wave, employing superconducting qubits, allows fine control over annealing durations and iteration counts. The Ising model is sent to the D-Wave cloud, where QA is performed to find the lowest energy state, with solutions subsequently returned to the classical computer.

TABLE II
THE COMPARISON OF NUMBER OF VARIABLES BETWEEN [26] AND WRIM

d	qubits	[26]	WRIM with physical error rate (%)				
			1	2	5	10	15
5	41	109	2.35	5.63	20.94	41.82	48.73
7	85	247	9.18	21.9	79.58	112.48	135.18
9	145	441	31.73	70.99	159.96	217.17	254.08
11	221	691	74	154.6	253.01	353.43	444.56

TABLE III
TIME COMPLEXITY FOR THE FIXED PHYSICAL ERROR RATES OF THE WRIM, ISING [26], AND MWPM DECODERS.

p (%)	WRIM's Ising	WRIM	Ising [26]	MWPM	UF
0.1	$O(n^{0.002})$	$O(n^{0.002})+O(kn)$	$O(n^{1.01})$	$O(n^{2.06})$	$O(n\log n)$
1.0	$O(n^{0.01})$	$O(n^{0.01})+O(kn)$	$O(n^{1.79})$	$O(n^{2.10})$	
2.0	$O(n^{0.04})$	$O(n^{0.04})+O(kn)$	$O(n^{1.74})$	$O(n^{2.24})$	
5.0	$O(n^{0.09})$	$O(n^{0.09})+O(kn)$	$O(n^{1.84})$	$O(n^{2.50})$	
10	$O(n^{0.24})$	$O(n^{0.24})+O(kn)$	$O(n^{1.81})$	$O(n^{2.72})$	
20	$O(n^{0.97})$	$O(n^{0.97})+O(kn)$	$O(n^{1.54})$	$O(n^{2.83})$	

A. Number of variables

The number of variables in WRIM was compared to the Ising model-based decoder [26] across code distances ranging from 5 to 11 using 100 random error patterns per distance, and physical error rates of 1%, 2%, 5%, 10%, and 15%. In [26], the variable count remains constant for a given code distance, as all syndromes and data qubits are considered. In contrast, WRIM's variable count varies with error patterns and physical error rates. The average number of variables for each distance and error rate was recorded. Table II details the code distance (d), the number of data qubits (qubits), the variable count for the Ising model-based decoder [26], and WRIM's average variable count under different error rates.

As the code distance increases, so does the number of variables. The WRIM decoder significantly reduces the required variables compared to [26], with fewer variables needed at lower physical error rates. This reduction occurs because WRIM excludes syndromes and qubits outside the defined region and simplifies the Ising model by eliminating the need for additional variables to reduce higher-order terms.

B. Decoding Time

Decoding time is critical as qubit states may collapse before decoding completes. The decoding time of the WRIM decoder consists of two components: the time to form range-constrained regions and the time required by the annealing machine to solve the Ising model. Communication time between classical computers and cloud servers, such as AE and D-Wave, are excluded, as the QEC decoders as well as qubit controllers are expected to be deployed near qubits to enable high-fidelity and real-time error correction [13], [36].

1) Range-Constrained Region Formation Time:

Algorithm 1 shows a worst-case time complexity of $O(n)$. Experiments were conducted for d values from 3 to 12 and physical error rates of 1%, 5%, 10%, 15%, and 20%. The

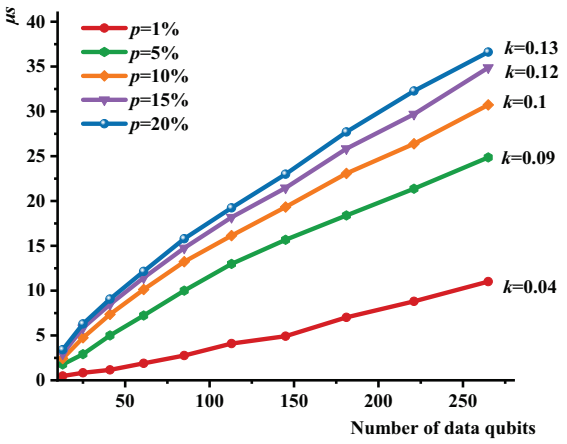


Fig. 4. Region formation time as a function of data qubits count under different physical error rates.

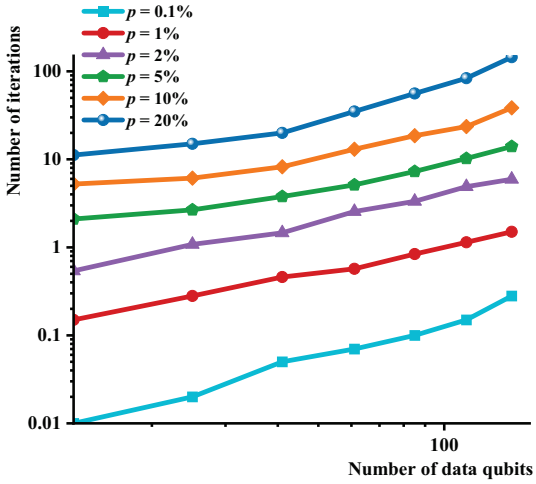


Fig. 5. Number of iterations as a function of data qubits count under different physical error rates.

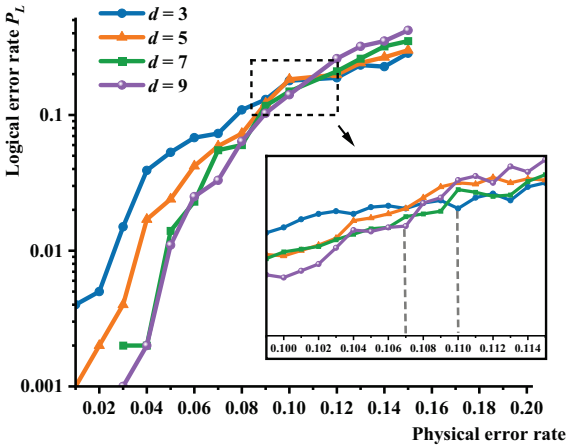


Fig. 6. Correlation between the calculated logical error rate P_L and the physical error rate p for d from 3 to 9.

TABLE IV
THE DECODING TIME USING QUANTUM ANNEALER D-WAVE.

d	p (%)	link time (μs)	annealing time (μs)	decoding time (μs)
5	0.1	0.26	0.03	0.29
7	0.1	0.33	0.05	0.38
9	0.1	0.89	0.10	0.99
5	1	7.84	37.90	45.74
7	1	13.28	63.50	76.78
9	1	14.46	195.30	209.76

time required to form regions is shown in Fig. 4. The time complexity with respect to the number of data qubits is $O(kn)$, with k increasing as the physical error rate rises.

2) Decoding Time by Ising Machine:

The average number of iterations for d from 3 to 9 with physical error rates of 0.1%, 1%, 2%, 5%, 10%, and 20% were tested. The results are presented in the double logarithmic plot in Fig. 5. The plot exhibits approximately linear behavior, indicating that the number of iterations scales as a polynomial function of the number of data qubits. Through regression analysis, the polynomial degree is found to be 0.002 (minimum) for $p = 0.1\%$ and 0.97 (maximum) for $p = 20\%$.

We examine the time complexity of the WRIM decoder, including range-constrained region formation and Ising annealing iterations, and compare it with other decoders, including the Ising model-based decoder [26], MWPM [26] and UF, as shown in Table III. Compared to SOTA Ising-model decoder [26], the reduction in time complexity of WRIM results from eliminating unnecessary variables, lowering the overall problem complexity significantly. Additionally, the weight (h and J) settings also help distinguish between the global minimum and local minima during the solution search, further reducing the search complexity. As a result, the complexity of decoding the Ising model is reduced. Compared to currently the fastest decoder UF, it is evident that the time complexity of iteration in WRIM is less than $O(n)$, resulting in an overall time complexity $O(n)$, which is lower than the time complexity of UF ($n \log n$).

The actual decoding time is also crucial for practical applications. The machines that solve the Ising model such as the SQA machine AE [35], simulated annealing like OpenJij [37], or true QA machines like D-Wave [29], have varying annealing times. In this experiment, real QA machine D-Wave is used to test the average annealing time for 100 error patterns at physical error rates of 0.1% and 1% to demonstrating the potential of QA and the WRIM decoder, as shown in Table IV.

At physical error rates of 0.1% and 1%, consistent with current advanced error rates ranging from 0.1% [11] to 1% [38], WRIM achieves decoding speeds on the order of microseconds, meeting the high-speed requirements for superconducting transmon qubits. It is important to note that QA machines are still in the developmental stage, particularly in terms of hardware scale and graph topology. As QA technology advances, the decoding speed of WRIM is expected to improve, along with an expansion in the decoding scale.

C. Logical Error Rate

The logical error rate is another key metric for QEC decoders. We evaluate the logical error rate threshold across code distances from 3 to 9. The threshold is defined as the physical error rate below which the logical error rate can be reduced by increasing the code distance. For each physical error rate, 1000 samples were tested to calculate the logical error rate.

The results are shown in Fig. 6. The threshold for the WRIM decoder lies between 10.7% and 11.0%, which is higher than the highest threshold of 10.3% achieved by the MWPM. The reason is that, under the condition of satisfying syndrome parity, weights with larger values of h tend to find error patterns that shorten each individual error chain, rather than minimizing the total number of errors. In contrast, the MWPM minimizes the total weight of the error chain, which typically results in patterns that reduce the overall error count but may not be the shortest for each error chain.

V. CONCLUSION

This paper proposed the Weighted Range-Constrained Ising Model-Based Decoder (WRIM), a fast Ising model-based decoder for decoding surface codes. The WRIM decoder employs a polygonal region to enclose flipped syndromes with their neighbors, limiting the number of nodes while ensuring the connectivity of all flipped syndromes. By constraining qubits to this region, the Ising model for error decoding is formulated to capture all potential error chains and error sites. It then assigns optimal weights to the Ising model's coupling coefficients and external field coefficient based on the satisfaction of syndrome parity for each error pattern, aiming to minimize the length of each error chain. Experimental results demonstrate the superior performance of WRIM, achieving up to a 97.8x reduction in variables compared to SOTA Ising model-based decoders while also enabling microsecond-level decoding at 0.1% and 1% physical error rates on the real quantum annealer D-Wave. Furthermore, the region algorithm of WRIM on a classical computer has a complexity of $O(kn)$ ($k \leq 1$), with decoding by Ising machine achieving $O(n^a)$ ($a \leq 1$), resulting in overall time complexity of decoding time $O(n)$. This is lower than the $O(n \log n)$ time complexity of the current fastest decoder, UF. Additionally, because WRIM uses weight to ensure that every error pattern has relatively shorter error chains rather than globally shortest chains, WRIM exhibits threshold behavior, maintaining performance at physical error rates up to 10.7-11.0%, making it surpass the highest threshold decoder MWPM.

ACKNOWLEDGMENT

This work was partially supported by Japan Science Technology (JST) Moonshot R&D Grant Number JPMJMS226A. It was also supported by JSPS KAKENHI grant Nos. 23K28052 and 23K18462. Additional support was provided by JST SPRING, Grant Number JPMJSP2110.

REFERENCES

[1] Michael A Nielsen and Isaac Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2002.

[2] Torsten Hoefler, Thomas Häner, and Matthias Troyer. Disentangling hype from practicality: On realistically achieving quantum advantage. *Communications of the ACM*, 66(5):82–87, 2023.

[3] Adarsh Kumar, Carlo Ottaviani, Sukhpal Singh Gill, and Rajkumar Buyya. Securing the future internet of things with post-quantum cryptography. *Security and Privacy*, 5(2):e200, 2022.

[4] Miguel Orszag and Maritza Hernandez. Coherence and entanglement in a two-qubit system. *Advances in Optics and Photonics*, 2(2):229–286, 2010.

[5] John Preskill. Fault-tolerant quantum computation. In *Introduction to quantum computation and information*, pages 213–269. World Scientific, 1998.

[6] Frank Gaitan. *Quantum error correction and fault tolerant quantum computing*. CRC Press, 2008.

[7] A Yu Kitaev. Fault-tolerant quantum computation by anyons. *Annals of physics*, 303(1):2–30, 2003.

[8] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A—Atomic, Molecular, and Optical Physics*, 86(3):032324, 2012.

[9] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[10] Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307–346, 2015.

[11] Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, 2023.

[12] J Ferreira Marques, BM Varbanov, MS Moreira, Hany Ali, Nandini Muthusubramanian, Christos Zachariadis, Francesco Battistel, Marc Beekman, Nadia Haider, Wouter Vlothuizen, et al. Logical-qubit operations in an error-detecting surface code. *Nature Physics*, 18(1):80–86, 2022.

[13] Ciaran Ryan-Anderson, Justin G Bohnet, Kenny Lee, Daniel Gresh, Aaron Hankin, JP Gaebler, David Francois, Alexander Chernoguzov, Dominic Lucchetti, Natalie C Brown, et al. Realization of real-time fault-tolerant quantum error correction. *Physical Review X*, 11(4):041058, 2021.

[14] Zvi Galil. Efficient algorithms for finding maximum matching in graphs. *ACM Computing Surveys (CSUR)*, 18(1):23–38, 1986.

[15] Suhas Vittal, Poulami Das, and Moinuddin Qureshi. Astrea: Accurate quantum error-decoding via practical minimum-weight perfect-matching. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pages 1–16, 2023.

[16] Oscar Higgott. Pymatching: A python package for decoding quantum codes with minimum-weight perfect matching. *ACM Transactions on Quantum Computing*, 3(3):1–16, 2022.

[17] Yue Wu and Lin Zhong. Fusion blossom: Fast MWPM decoders for QEC. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, volume 01, pages 928–938, 2023.

[18] Ariful Azad, Aydin Buluç, Xiaoye S Li, Xinliang Wang, and Johannes Langguth. A distributed-memory algorithm for computing a heavy-weight perfect matching on bipartite graphs. *SIAM Journal on Scientific Computing*, 42(4):C143–C168, 2020.

[19] Sergey Bravyi, Martin Suchara, and Alexander Vargo. Efficient algorithms for maximum likelihood decoding in the surface code. *Physical Review A*, 90(3):032326, 2014.

[20] Yosuke Ueno, Masaaki Kondo, Masamitsu Tanaka, Yasunari Suzuki, and Yutaka Tabuchi. Neo-qec: Neural network enhanced online superconducting decoder for surface codes. *arXiv preprint arXiv:2208.05758*, 2022.

[21] Savvas Varsamopoulos, Koen Bertels, and Carmen Garcia Almudever. Comparing neural network based decoders for the surface code. *IEEE Transactions on Computers*, 69(2):300–311, 2019.

[22] Spiro Gicev, Lloyd CL Hollenberg, and Muhammad Usman. A scalable and fast artificial neural network syndrome decoder for surface codes. *Quantum*, 7:1058, 2023.

[23] Nicolas Delfosse and Naomi H Nickerson. Almost-linear time decoding algorithm for topological codes. *Quantum*, 5:595, 2021.

[24] Maximilian Jakob Heer, Emanuele Del Sozzo, Keisuke Fujii, and Kentaro Sano. Novel union-find-based decoders for scalable quantum error correction on systolic arrays. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 524–533. IEEE, 2023.

[25] Maximilian Jakob Heer, Emanuele Del Sozzo, Keisuke Fujii, and Kentaro Sano. Novel union-find-based decoders for scalable quantum error correction on systolic arrays. In *2023 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 524–533, 2023.

[26] Jun Fujisaki, Hirotaka Oshima, Shintaro Sato, and Keisuke Fujii. Practical and scalable decoder for topological quantum error correction with an ising machine. *Physical Review Research*, 4(4):043086, 2022.

[27] Jun Fujisaki, Kazunori Maruyama, Hirotaka Oshima, Shintaro Sato, Tatsuya Sakashita, Yusaku Takeuchi, and Keisuke Fujii. Quantum error correction with an Ising machine under circuit-level noise. *Physical Review Research*, 5(4):043261, 2023.

[28] Computing as a service digital annealer. https://www.fujitsu.com/jp/digitalannealer/, Last accessed on September 2, 2024.

[29] Wave systems: The practical quantum computing company. https://www.dwavesys.com/, Last accessed on March 5, 2023.

[30] Dominic Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[31] J Pablo Bonilla Ataides, David K Tuckett, Stephen D Bartlett, Steven T Flammia, and Benjamin J Brown. The XZZX surface code. *Nature communications*, 12(1):2172, 2021.

[32] Hector Bombin and Miguel Angel Martin-Delgado. Topological quantum distillation. *Physical review letters*, 97(18):180501, 2006.

[33] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[34] Fred Glover, Gary Kochenberger, Rick Hennig, and Yu Du. Quantum bridge analytics I: a tutorial on formulating and using QUBO models. *Annals of Operations Research*, 314(1):141–183, 2022.

[35] The quantum computing cloud - fixstars amplify. https://amplify.fixstars.com/, Last accessed on September 2, 2024.

[36] SJ Pauka, K Das, R Kalra, A Moini, Y Yang, M Trainer, A Bousquet, C Cantaloube, N Dick, GC Gardner, et al. A cryogenic cmos chip for generating control signals for multiple qubits. *Nature Electronics*, 4(1):64–70, 2021.

[37] OpenJij : Framework for the ising model and QUBO. https://openjij.github.io/OpenJij/reference/openjij/, Last accessed on September 2, 2024.

[38] Austin G Fowler, Ashley M Stephens, and Peter Groszkowski. High-threshold universal quantum computation on the surface code. *Physical Review A—Atomic, Molecular, and Optical Physics*, 80(5):052312, 2009.

PatLabor: Pareto Optimization of Timing-Driven Routing Trees

Zhiyang Chen[1], Hailong Yao[23], Xia Yin[1]
[1]Tsinghua University, [2]University of Science and Technology Beijing,
[3]Key Laboratory of Advanced Materials and Devices for Post-Moore Chips, Ministry of Education of China

***Abstract*—Wirelength is the fundamental metric for VLSI routing. With the advancement of new technologies, wire delay has also become a significant factor for timing performances. It is thus necessary to consider both wirelength and delay in routing tree construction, i.e., timing-driven routing trees. Prior methods propose various heuristics to balance wirelength and delay with a tunable parameter, which cannot compute the full Pareto frontier. In this work, we propose PatLabor, a practical method for timing-driven routing. PatLabor directly optimizes the Pareto set, which obtains tighter Pareto curves than prior methods and does not require parameter tuning. PatLabor obtains all Pareto-optimal solutions on small-degree nets up to 9 pins and is theoretically guaranteed by provable time complexity and approximation bounds. Experimental results verify our theoretical findings and show that PatLabor obtains tighter Pareto curves than state-of-the-art methods on ICCAD-15 benchmarks. For example, PatLabor obtains up to 58.5% more Pareto-optimal solutions than prior methods for degree-9 nets.**

I. Introduction

Timing has become a critical performance metric for modern VLSI design. As Markov [1] points out, with the advancement of new technologies, wire delay has become a significant factor in performance issues. It is hence necessary to consider both wirelength and delay in the routing stage [2]. Therefore, we study the problem of *timing-driven routing trees*: Given a degree-n net with one pin as the source and other $n-1$ pins as sinks, the objective is to construct a rectilinear Steiner tree topology for these pins that minimize both the total wirelength and the path lengths from the source to sinks. However, these two objectives may be contradictory to each other. Instead of finding a best-of-both-world solution, we aim to find a set of solutions, targeting the Pareto frontier of these two objectives. We study and propose algorithmic methods for bicriterion Pareto optimization of timing-driven routing trees (see Figure 1).

Why Pareto optimization? Computing a Pareto set of solutions for bicriterion problems provides more decision options and better characterizes the relationship between objectives. Recent work on global routing [3] shows that selecting net topologies from a candidate solution set may improve the performance of global routers. Pareto optimization for timing-driven routing trees can be naturally applied to timing optimization in global routing.

A. Our Contributions

Our main contributions are listed as follows:

- We study the size of Pareto frontiers for timing-driven routing trees. We prove that routing instances have only a polynomial size of the Pareto frontier in expectation using smoothed analysis. This provides theoretical foundations for Pareto optimization of timing-driven routing trees. (Section III)
- We propose algorithms for Pareto optimization of timing-driven routing, including an exponential-time exact algorithm and a polynomial-time approximation algorithm. We prove time complexity and approximation bounds. Although these algorithms cannot be directly applied in practice, they serve as important building blocks for our practical methods. (Section IV)

Corresponding author: Hailong Yao. This work was supported by the Key Program of National Natural Science Foundation of China (No. 62034005).

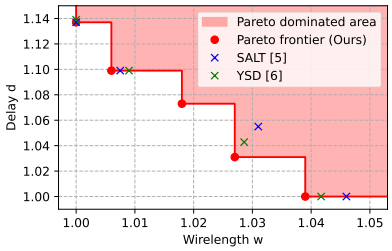


Fig. 1. An example of Pareto optimization from ICCAD-15 benchmarks. Our method obtains the full Pareto frontier while prior methods cannot.

- Based on the theoretical results, we propose **PatLabor**, a practical method for Pareto optimization of timing-driven routing trees. For small-degree nets ($n \leq 9$), we construct lookup tables to compute the Pareto frontier optimally and efficiently. For large-degree nets, we propose a local search heuristic to approximate the Pareto frontier. (Section V)
- Experimental results on ICCAD-15 benchmarks and randomly generated instances illustrate the effectiveness of PatLabor. We first empirically verify our theoretical results in Section III. Then, we present results of constructing lookup tables for small-degree nets and show that our construction method is about $441\times$ faster than FLUTE [4]. Finally, we empirically show that PatLabor obtains tighter Pareto curves than previous timing-driven routing algorithms [5] [6]. For example, PatLabor obtains up to 58.5% more solutions on the Pareto frontier than prior methods for degree-9 nets. (Section VI)

B. Comparisons with Related Work

Rectilinear Steiner minimum trees (RSMT). The most basic routing tree problem is to minimize the total wirelength. Many methods [7] [8] [4] [9] have been proposed to solve RSMT, among which the most popular method is FLUTE [4]. It is worth noting that our idea of using lookup tables for small-degree nets is motivated by FLUTE. FLUTE uses an exhaustive search on boundary compaction to generate lookup tables. However, we propose a new dynamic-program-based method to generate lookup tables, which is more efficient than FLUTE.

Rectilinear Steiner minimum arborescence (RSMA). The RSMA problem [10] [11] [12] aims to find the shortest path Steiner tree for the net with minimum total wirelength. Córdova and Lee (CL) [11] propose a 2-approximation algorithm, which is the most popular for RSMA construction in practice.

Note that RSMT and RSMA are NP-hard [13] [14]. Both RSMT and RSMA are single-objective problems, which are insufficient for timing-driven routing since it has two objectives.

Timing-driven routing trees. Alpert et al. [2] propose Prim-Dijkstra (PD-II), an algorithm blending Prim and Dijkstra to balance the total wirelength and the maximum path length. Chen and Young [5] propose SALT, a method based on a classic shallow-light tree algorithm [15], together with post-processing heuristics. The most recent work is due to Yang, Sun, and Ding (YSD) [6], which combines

TABLE I
THE LIST OF NOTATIONS.

Notation	Implication
$\tilde{O}(f(n))$	$f(n) \cdot (\log n)^{O(1)}$
$O^*(f(n))$	$f(n) \cdot n^{O(1)}$
n	The number of pins
$\|\cdot\|_1$	The l_1 norm (rectilinear distance)
$P = \{p_1, \ldots, p_n\}$	A set of pins with each $p_i = (x_i, y_i)$
(r, P)	A problem instance with source r and pins P
T	A rectilinear Steiner routing tree
$w(T)$	The wirelength of T
$d(T)$	The delay of T
$s(T) = (w(T), d(T))$	The optimization objectives of T
$s_1 \preceq s_2$	Pareto dominance
$S = \{s_i(T_i)\}_i$	A set of solution objectives
$l_1, \ldots, l_{2n-2}$	The lengths of Hanan grids (see Figure 3)

a neural network for small-degree nets, and a divide-and-conquer framework for large-degree nets. More older work [16] [17] [18] are omitted due to the page limit. Prior methods introduce parameters to balance wirelength and delay, which is different from our Pareto optimization method.

We outline some superior features of our method:

The first Pareto optimization method. To the best of our knowledge, this is the first work that directly optimizes the Pareto curve of timing-driven routing. Previous methods introduce parameters to balance the two objectives and generate only one solution for a single parameter value. Tuning parameters is usually painful for heuristic algorithms. Li et al. [19] use a complicated neural network to predict the parameters for PD-II and SALT, which brings heavy training and inference overhead. Our method directly outputs a set of solutions on the Pareto curve, and users can choose any solution they need, without spending time on parameter tuning.

Optimality on small-degree nets. Our method ensures Pareto optimality for nets with $n \leq 9$ pins. Note that none of the prior methods [2] [5] [6] provide optimality guarantees even for nets with $n = 5$ pins. Our method applies an exact dynamic programming algorithm to produce lookup tables containing all potentially optimal tree topologies for small-degree nets, and is hence able to produce all solutions on the Pareto frontier.

Tighter Pareto curves on large-degree nets. For nets with $n > 9$ pins, our method theoretically has the potential to obtain tighter Pareto curves. In Section IV, we prove an approximation bound showing that our method can approximate *every* solution on the Pareto frontier. This is a stronger guarantee compared with SALT, which uses a parameter to control the tradeoff of the approximation ratios. Moreover, our method avoids the issue caused by the weighted sum method in YSD, which only obtains convex Pareto curves.

II. PRELIMINARIES

Notations. Following the convention of algorithm analysis, we use $\tilde{O}(f(n)) = f(n) \cdot (\log n)^{O(1)}$ to suppress logarithmic factors, and use $O^*(f(n)) = f(n) \cdot n^{O(1)}$ to suppress polynomial factors for simplicity of notations.

We follow classic notations for multi-objective optimization. We use $s = (s_1, s_2)$ to denote two objectives of a solution. Since timing-driven routing is a minimization problem, given two solutions $s \neq s'$, we say s *dominates* s' (formally, $s \preceq s'$) if $s_1 \leq s'_1$ and $s_2 \leq s'_2$. The *Pareto curve* (or *Pareto set*) is a set of solutions $\{s_1, \ldots, s_k\}$ such that $s_i \not\preceq s_j$ for any $i \neq j$. The *Pareto frontier* is the optimal Pareto curve. Solutions on the Pareto frontier are *Pareto-optimal*.

Problem formulation. Following previous work [2] [6], we define the *timing-driven routing* problem. We are given a set of points $P = \{p_1, \ldots, p_n\}$ from the metric space $(\mathbb{R}^2, \|\cdot\|_1)$, where each point $p_i = (x_i, y_i)$ denotes a pin from the net to be routed. We use $r = p_1 = (x_1, y_1)$ to denote the *source* of the net and the remaining $n - 1$ pins are sinks. The target is to construct a rectilinear Steiner tree $T = (V, E)$ for P rooted at r, considering both the sum of edge lengths of T (wirelength $w(T)$) and the maximum path length from source r to any sink $p_i \in P\,(i \geq 2)$ (delay $d(T)$). Previous work minimizes a weighted sum of $w(T)$ and $d(T)$. We instead aim to compute the Pareto frontier of $(w(T), d(T))$ (an example of three Pareto-optimal solutions is shown in Figure 2).

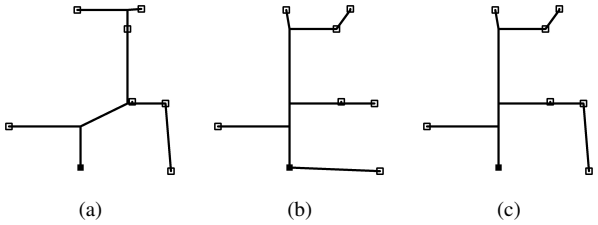


Fig. 2. An example of timing-driven routing trees from ICCAD-15 benchmarks with three Pareto-optimal solutions. Solid squares are source pins. (a) Minimizing total wirelength ($w = 23882, d = 13397$); (b) Minimizing sink delay ($w = 26084, d = 11538$); (c) Balancing wirelength and delay ($w = 24879, d = 11682$).

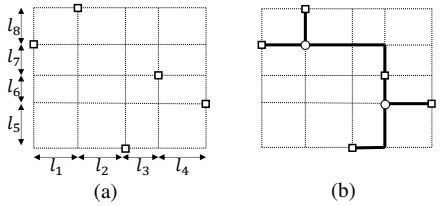


Fig. 3. (a) The Hanan grid; (b) A routing tree constructed on the grid. Squares represent pins and circles represent Steiner points.

It is folklore that an optimal RSMT can be constructed on the Hanan grid [20]. We point out that the same holds for the Pareto-optimal timing-driven routing trees. We use $l_1, \ldots, l_{n-1}$ to denote lengths of horizontal grid edges, and $l_n, \ldots, l_{2n-2}$ to denote lengths of vertical edges on the Hanan grid (see Figure 3). The notations are summarized in Table I.

III. ANALYSIS OF PARETO FRONTIERS

In this section, we analyze the Pareto frontier of timing-driven routing trees. It is known that a multi-objective problem usually has an exponential size of the Pareto frontier [21], which makes solving multi-objective optimization intractable (even if $\mathsf{P} = \mathsf{NP}$).

Theorem 1: *There exists a timing-driven routing instance* (r, P) *such that the number of solutions on the Pareto frontier is* $2^{\Omega(n)}$.

Proof Sketch: The construction of the instance is illustrated in Figure 4. The instance consists of m "S-shape" gadgets, with 11 pins each. The m gadgets are placed in a diagonal pattern, with a source pin at the lower-left corner and another pin at the upper-right corner. In the k-th gadget, we set $x = 2^{k-2}$ and $y = 2^{k-1} + 2^{k-3}$. In total, we have $n = 11m + 2$ pins. It can be verified that the number of Pareto-optimal solutions is at least $2^m = 2^{(n-2)/11} = 2^{\Omega(n)}$. □

However, such bad instances never appear in practice. In the following, we show that the size of the Pareto frontier for timing-driven routing in real-world testcases is only polynomial. Based on this result, we present efficient algorithms for computing Pareto frontiers of timing-driven routing in the next section.

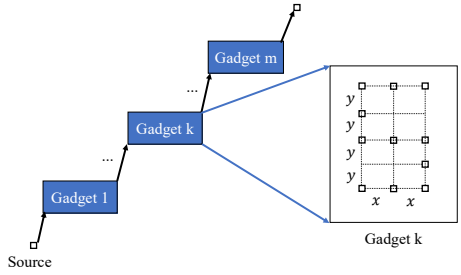


Fig. 4. The construction of an instance with an exponential number of solutions on the Pareto frontier in Theorem 1.

One way of modeling real-world instances is to use average-case analysis, assuming the pins are uniformly random points on the routing plane. However, this assumption is too strong. In practice, the problem instance cannot be seen as purely random instances on the entire routing plane. Alternatively, our idea is to study the Pareto frontier using *smoothed analysis*, which is a popular algorithmic framework to explain the practical behavior of many algorithms proposed by Spielman and Teng [22]. The philosophy is that in practice, there is always some small noise in the problem instance, and the performance of an algorithm seldom achieves the worst-case bound. In timing-driven routing, the positions of pins are generated by VLSI placement. A small perturbation on the positions of a pin does not impact placement-and-routing results greatly. The assumption is hence reasonable.

Smoothed analysis interpolates between worst-case analysis and average-case analysis. It is a relaxation of worst-case analysis, but with a weaker assumption than average-case analysis.

Definition 1 (Smoothed timing-driven routing instances): Without loss of generality, we assume the coordinates of pins in the routing instance $P = \{p_1, \ldots, p_n\}$ lie in $[0, 1]$. In a *κ-smoothed instance*, the coordinates x_i and y_i of pins are sampled independently from distributions with probability density at most κ on $[0, 1]$.

For example, consider a degree-2 net with x_1 uniformly sampled from $[0, 0.25]$, y_1 sampled from $[0.1, 0.35]$, x_2 sampled from $[0.6, 0.85]$ and y_2 sampled from $[0.65, 0.9]$. We say this net is a 4-smoothed instance. As $\kappa \to \infty$, the framework reduces to worst-case analysis. If $\kappa = 1$, the framework reduces to average-case analysis.

Theorem 2: *The expected number of solutions on the Pareto frontier for κ-smoothed timing-driven routing is $O(n^3\kappa)$.*

Proof Sketch: Due to the page limit, we only sketch the proof ideas. The main proof technique is to divide the range of objectives into small intervals. Since the edge weights are κ-smoothed, the probability density is at most κ. Therefore, the probability of a Pareto-optimal solution falling into an interval of length l is only $l \cdot \kappa$. The total length of all intervals is only $\mathrm{poly}(n)$, and the expected number of optimal solutions is hence at most $\mathrm{poly}(n) \cdot \kappa$. □

In practice, we assume the nets contain small randomness, and thus regard κ as a constant. Based on Theorem 2, it is possible to design efficient algorithms for timing-driven routing trees.

IV. Theoretical Algorithms for Timing-Driven Routing

In this section, we present two theoretical algorithms for timing-driven routing trees. The time complexities of both algorithms depend on the size of Pareto frontiers, and are thus efficient by Theorem 2. Although they cannot be directly applied in practice, their ideas serve as significant building blocks for our practical method in the next section.

A. An exponential-time exact algorithm

Dreyfus-Wagner (DW) [23] is a classic algorithm for minimum Steiner trees on graph metrics, which finds an exact solution in $O^*(3^n)$ time using a dynamic program. We can adapt this algorithm to compute the full Pareto frontier of timing-driven routing trees.

Our algorithm, namely **Pareto-DW**, is described as follows: Let $S_{v,Q}$ denote the Pareto frontier of routing trees with source v and sinks Q. Here, v is an arbitrary node on the Hanan grid, and $Q \subseteq P$ is a subset of pins. Each $S_{v,Q}$ is a Pareto set of solution objectives (w, d). Then, $S_{r,P}$ is our desired result. We compute $S_{v,Q}$ using dynamic programming,

$$S_{v,Q} = \mathrm{Pareto} \begin{cases} \cup_u \left\{ S_{u,Q} + \|u - v\|_1 \right\}, \\ \cup_{Q_1 \subseteq Q} \left\{ S_{v,Q_1} \oplus S_{v,Q \setminus Q_1} \right\}. \end{cases} \tag{1}$$

Note that $S_{v,Q}$ is a set of vectors. For any solution sets S, S' and any $x \in \mathbb{R}$, we define

$$\begin{aligned} S + x &= \{(w + x, d + x) \,|\, (w, d) \in S\}, \\ S \oplus S' &= \{(w_1 + w_2, \max\{d_1, d_2\}) \\ &\qquad |\, (w_1, d_1) \in S, (w_2, d_2) \in S'\}. \end{aligned}$$

Note that $S + x$ and $S \oplus S'$ can be computed in $O(|S|)$ and $O(|S| \cdot |S'|)$ time, respectively. We also define $\mathrm{Pareto}(S) = \{s \in S \,|\, \nexists s' \in S, s' \preceq s\}$, i.e., eliminating all solutions in S off the Pareto curve. Note that this is equivalent to finding maximal points on a plane, which can be computed in $O(|S| \log |S|)$ time [24].

Theorem 3: *Pareto-DW finds the Pareto frontier of timing-driven routing in $O^*(3^n \cdot |S|^2)$ time, where $|S|$ is the largest number of solutions in $S_{v,Q}$ for any v, Q.*

This theorem can be proved by exploiting the recursive structure of Steiner trees, and the proof is omitted due to the page limit. We point out that by Theorem 2, the size of Pareto frontier $|S|$ is only polynomial in n. Therefore, in practice, we can solve timing-driven routing using $O^*(3^n)$ time, the same as the vanilla Dreyfus-Wagner. Without smoothed analysis, the time complexity may become $O^*(3^n) \cdot 2^{O(n)} \cdot 2^{O(n)} = 12^{O(n)}$, which is unacceptable even for $n \leq 10$ pins.

There are also many acceleration techniques for Pareto-DW. Although they cannot improve the worst-case complexity theoretically, they can effectively prune the states of the dynamic program. We introduce them in Section V.

B. A polynomial-time approximation algorithm

Pareto-DW can only solve small-degree instances due to the exponential complexity. Using Pareto-DW as a subroutine, we can design a polynomial-time approximation algorithm for timing-driven routing. The idea is to extend the Kalpakis-Sherman (KS) heuristic[1] [26] to a multi-objective version: We use divide-and-conquer to partition the routing plane into sub-problems. When the number of pins is small enough, we use Pareto-DW to compute the exact Pareto frontier. We formally describe the algorithm as follows.

Algorithm Pareto-KS(P):

1. If $|P| \leq \log n$, return Pareto-DW(P) to obtain the Pareto frontier.
2. We find a pin $p_i \in P$ such that there are at least $\lfloor |P|/2 \rfloor - 1$ pins on each side of p_i (divide on the x- or y-axis alternatively) to partition the pins into P_1 and P_2.

[1] In essence, the classic RSMT heuristic, FLUTE [4], is a variant of the KS algorithm. The effectiveness of KS in practice is the reason why we choose to adapt KS instead of other algorithms, e.g., Arora's PTAS for geometric Steiner trees [25] (although Arora's may achieve better approximation bounds).

3. Call Pareto-KS(P_1) and Pareto-KS(P_2) to obtain S_1 and S_2 recursively. We select the pin closest to r as the source pin.
4. Return the combination of the solutions in S_1 and S_2.

Definition 2 (Pareto approximation): Given a multi-objective minimization problem, and an algorithm that outputs a Pareto set S for any instance of the problem, we say the algorithm *c-approximates* the Pareto frontier, if, for any solution s on the Pareto frontier, we can find $s' \in S$, such that $s' \preceq c \cdot s$.

Theorem 4: *Pareto-KS is an $O(\sqrt{n/\log n})$-approximation algorithm in $\tilde{O}(n^2 \cdot |S|^2)$ time, where $|S|$ is the size of the largest Pareto set during the execution of Pareto-KS.*

Proof Sketch: We only prove the time complexity due to the page limit. The proof of the approximation ratio can be obtained by showing that timing-driven routing is a *sub-additive Euclidean functional* [26] and is omitted.

Pareto-KS partitions the routing plane into $\Theta(n/\log n)$ sub-rectangles. In each rectangle, there are at most $\log n$ pins. The complexity of computing the Pareto frontier in each rectangle is hence $O(3^{\log n} \cdot \mathrm{poly}(\log n)) = \tilde{O}(n)$. Given two Pareto sets S_1, S_2, we can compute their Pareto sum in $\tilde{O}(|S_1| \cdot |S_2|)$ time. Therefore, the total complexity is $\tilde{O}(n) \cdot O(n/\log n) \cdot \tilde{O}(|S|^2) = \tilde{O}(n^2 \cdot |S|^2)$. □

V. Practical Methods for Timing-Driven Routing

In this section, based on Pareto-DW and Pareto-KS, we present **PatLabor** (Pareto optimization with Lookup tables and local search), a practical method for timing-driven routing. PatLabor consists of two parts: For small-degree nets, we build lookup tables to efficiently and optimally solve the problem. For large-degree nets, we present a local search heuristic to approximate the Pareto frontier.

A. Lookup tables for small-degree nets

Although the Pareto-DW algorithm is fast for small-degree nets, it is not efficient enough to route millions of nets in VLSI designs. Motivated by the lookup-table-based FLUTE heuristic [4] for RSMTs, we store all potentially Pareto-optimal routing tree topologies in a lookup table for all instances with $n \le \lambda$ so that we can directly access the Pareto frontier efficiently. Following FLUTE, we set $\lambda = 9$. We use modified Pareto-DW to generate the lookup tables.

For each possible permutation of pins P and a source r, we can build a Hanan grid for pins and use $l_1, \ldots, l_{2n-2}$ to denote the lengths of grid edges as in Figure 3. For degree-n nets, there are at most $n!$ different patterns of the Hanan grid. As in Pareto-DW, we use $S_{v,Q}$ to denote the set of potential Pareto-optimal topologies with source v and pins Q. We follow the DP Equation (1). However, a solution in $S_{v,Q}$ is no longer $(w, d) \in \mathbb{R}^2$, but in the form of

$$\left(\sum_{i=1}^{2n-2} w_i l_i, \max_i \sum_{j=1}^{2n-2} d_{ij} l_j \right).$$

We instead use a vector $W = (w_i)$ and a matrix $D = (d_{ij})$ to represent a solution. By definition, we have the following proposition.

Lemma 1: *Given two solutions $(W^{(1)}, D^{(1)})$ and $(W^{(2)}, D^{(2)})$, $(W^{(2)}, D^{(2)})$ can be safely pruned if for any $l_1, \ldots, l_{2n-2} \ge 0$,*

$$\sum_{i=1}^{2n-2} (w_i^{(2)} - w_i^{(1)}) \ge 0 \ \wedge \ \max_i \sum_{j=1}^{2n-2} d_{ij}^{(1)} l_j \le \max_i \sum_{j=1}^{2n-2} d_{ij}^{(2)} l_j. \quad (2)$$

The condition in Equation (2) is a first-order proposition, and hence can be verified by an SMT solver. We use an SMT solver [27] to safely prune non-optimal solutions during the generation of lookup tables. Since the number of pins is small, the SMT solver is efficient enough.

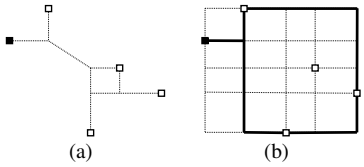


Fig. 5. Pruning techniques for Pareto-DW. The solid square is the source pin. (a) The pruned Hanan grid for Figure 3 after applying Lemma 2; (b) An example of the source on the outside of the bounding box for Lemma 3.

To accelerate generating lookup tables, we apply some reduction tricks. We reduce the number of Hanan grid patterns by breaking symmetries. There are in total $n! \cdot n$ choices of (r, P). However, if two patterns are equivalent under mirror and rotation transformations, only one pattern is needed to store in the table. Another important trick is to prune useless states in dynamic programming (Figure 5(a)).

Lemma 2: *A node $v = (x, y)$ on the Hanan grid is called a lower-left-corner node*[2]*, if there are no pins $p_i = (x_i, y_i) \in P$ such that $x_i \le x$, $y_i \le y$. The states $S_{v,Q}$ for any lower-left-corner node v can be safely pruned in Equation (1).*

We can also avoid subset enumeration in the dynamic program for many states (Figure 5(b)).

Lemma 3: *For a node v on the Hanan grid and a subset of pins $S \subseteq P$, if v is outside $\mathsf{BB}(S)$ (the bounding box of S), we have $S_{v,Q} = S_{u,Q} + \|v - u\|_1$, where u is the projection of v on $\mathsf{BB}(S)$.*

The last technique is based on *separators* for *outer-planar* graphs. For the definition of separators, see any parameterized algorithm textbooks, e.g., Cygan et al. [28].

Lemma 4: *For a node v and a subset of pins $S \subseteq P$, if all pins in S lie on the boundary of the grid and we relabel them by $1, 2, \ldots, |S|$ in the clockwise order (the labels are circular, i.e., $|S|$ and 1 are adjacent), then, in Equation (1), it suffices to consider transitions $S_{v,Q_1} \oplus S_{v,Q \setminus Q_1}$ for Q_1 such that the pins have consecutive labels.*

Lemmas 2, 3 and 4 can greatly reduce the running time of Pareto-DW. We omit the proofs of lemmas due to the page limit.

B. Local search heuristics for large-degree nets

For nets with degree $n > \lambda$, we reduce the problem to small-degree nets and apply the lookup table. A direct method is to replace dynamic programming in Pareto-KS with lookup tables.

Remark 1: If we use lookup tables for nets with degree-λ or less in Pareto-KS, we can instead obtain an $O(\sqrt{n/\lambda})$-approximation bound and an $\tilde{O}(n\lambda|S|^2)$ time bound in Theorem 4. In practice, most nets have $n \le 50$ pins, and hence n/λ can be seen as $O(1)$.

Although Pareto-KS is good in theory, it is not good enough in practice. The main reason is that the divide-and-conquer framework cannot appropriately exploit the relative positions of the source pin r and a sub-problem. Following the philosophy of Pareto-KS, we instead apply a local search method. We still reduce the problem to sub-problems, but focusing on pins with large delay. The algorithm is formally described as follows.

PatLabor for $n > \lambda$:

1. We maintain a Pareto set of tree solutions $\mathcal{T}$. Initially, we call FLUTE [4] to build an RSMT T_0 and set $\mathcal{T} = \{T_0\}$.
2. Select a tree $T \in \mathcal{T}$ with the largest maximum delay $d(T)$. Choose $\lambda - 1$ pins in T based on a given policy π. Together with source r, regenerate the topology of these λ pins by accessing lookup tables and obtain a set of new tree topologies $T_1, \ldots, T_m$.

[2]We can similarly define corner nodes on the other three corners.

3. Let $\mathcal{T} \leftarrow \mathcal{T} \cup \{T_1, \ldots, T_m\}$. Eliminate solutions in $\mathcal{T}$ that are off the Pareto curve.
4. Repeat steps 2 and 3 for $\lfloor n/\lambda \rfloor$ times.

Note that if we restrict the local search to access each pin only once, the algorithm is indeed a variant of Pareto-KS. In step 2, after a local search iteration, the new topologies may be sub-optimal since the local topology may be intersected with other $n - \lambda$ pins. We use post-processing techniques as in SALT [5] to refine these issues.

An important component in PatLabor is the policy π to select pins on which we perform local search. We define a heuristic scoring function for each pin, and iteratively select $\lambda - 1$ pins with the maximum score. Suppose we have already selected λ' pins $p_1, \ldots, p_{\lambda'}$, for each unselected pin p, let

$$\begin{aligned}\mathsf{score}(p) = {} & \alpha_1 \cdot \|r - p\|_1 + \alpha_2 \cdot \mathsf{dist}_T(r, p) \\ & - \alpha_3 \cdot \min_{1 \le \lambda_0 \le \lambda'} \|p - p_{\lambda_0}\|_1 - \alpha_4 \cdot \mathsf{HPWL}(p, p_1, \ldots, p_{\lambda'}),\end{aligned}$$

where $\alpha_1, \alpha_2, \alpha_3, \alpha_4 \ge 0$ are parameters. The first term $\|r - p\|_1$ is the rectilinear distance between source r and p. The second term $\mathsf{dist}_T(r, p)$ is the distance between r and p on the current tree topology T. We always select pins far from the source such that the delay is large. The third term $\min_{1 \le \lambda_0 \le \lambda'} \|p - p_{\lambda_0}\|_1$ is the minimum distance between p and already selected pins. The fourth term $\mathsf{HPWL}(p, p_1, \ldots, p_{\lambda'})$ is the half-perimeter wirelength of p and already selected pins. If no pins have been selected, these two terms are zero. After an iteration of local search, the selected pins are connected in a subtree. We hope these pins are near to each other. Thus, it is better for the first two terms to be large and the last two terms to be small.

We select the best parameters for policy π using a *reinforcement learning* apparatus. We use *policy iteration* [29] to train the parameters α. Concretely, we first randomly sample the selection of pins and run PatLabor on randomly generated testcases. We collect a set of selections of pins $(p_1, \ldots, p_{\lambda-1})$ such that the algorithm performances are better, and use linear regression to maximize the score function on these collected data. Motivated by the philosophy of *curriculum learning* [30], we choose different values of $\alpha^{(n)}$ for different degrees of n. We initially train the value of $\alpha^{(n)}$ for $n = \lambda + 1 = 10$. Suppose we have already selected $\alpha^{(n)}$ for some degree-n. Then, we use $\alpha^{(n)}$ as a warm-start for degree-$(n+1)$ and further optimize $\alpha^{(n+1)}$. Finally, we obtain the policy π for each $n \ge 10$. We finish training when $n = 100$ since most nets in our experiments have at most 100 pins.

We can use statistical learning theory to guarantee the generalization ability of the learned parameters. The following theorem is obtained by an upper bound on the pseudo-dimension [31] of the performance metric. We omit the proof due to the page limit.

Theorem 5: *Let $\mathcal{D}$ be an arbitrary distribution of degree-n timing-driven routing instances. Suppose the parameters α are learned based on m i.i.d. samples $x_1, \ldots, x_m$ from $\mathcal{D}$. Let $\mathsf{perf}(x, \alpha)$ denote any performance metric of PatLabor with parameters α on instance x. Then, with probability at least $1 - \delta$, for any $\alpha \in \mathbb{R}^4$, we can bound the generalization gap by*

$$\left| \frac{1}{m} \sum_{i=1}^{m} \mathsf{perf}(x_i, \alpha) - \mathbb{E}_{x \sim \mathcal{D}}[\mathsf{perf}(x, \alpha)] \right| \le \tilde{O}\left(\sqrt{\frac{n}{m}} \right).$$

VI. Experimental Results

In this section, we present experimental results. Following previous work, we use the ICCAD-15 benchmark to evaluate PatLabor and other baselines. The benchmark consists of 8 designs, with about 1.3×10^6 nets (we omit degree-2 and -3 nets since they are trivial).

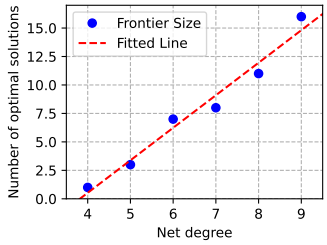


Fig. 6. Sizes of the maximum Pareto frontier from the ICCAD-15 benchmark. The fitted line ($y = 2.85x - 10.9$) is computed by linear regression.

TABLE II
Statistics of the lookup table.

Degree	#Index	#Topo	Size (MB)	Time
4	24	1.67	< 0.01	0s
5	220	4.6	< 0.01	0s
6	1008	10.67	< 0.01	0s
7	5824	32.52	0.19	4.9s
8	46880	107.05	6.23	276s
9	429516	378.05	240	4.68h
Total	483472	-	246	4.76h

#Index = the number of pairs (r, P), i.e., the source pin and the pin permutation.
#Topo = the average number of potentially optimal tree topologies.
Size = the storage size of lookup tables.
Time = (parallel) running time of generating the lookup table.

A. The size of Pareto frontiers

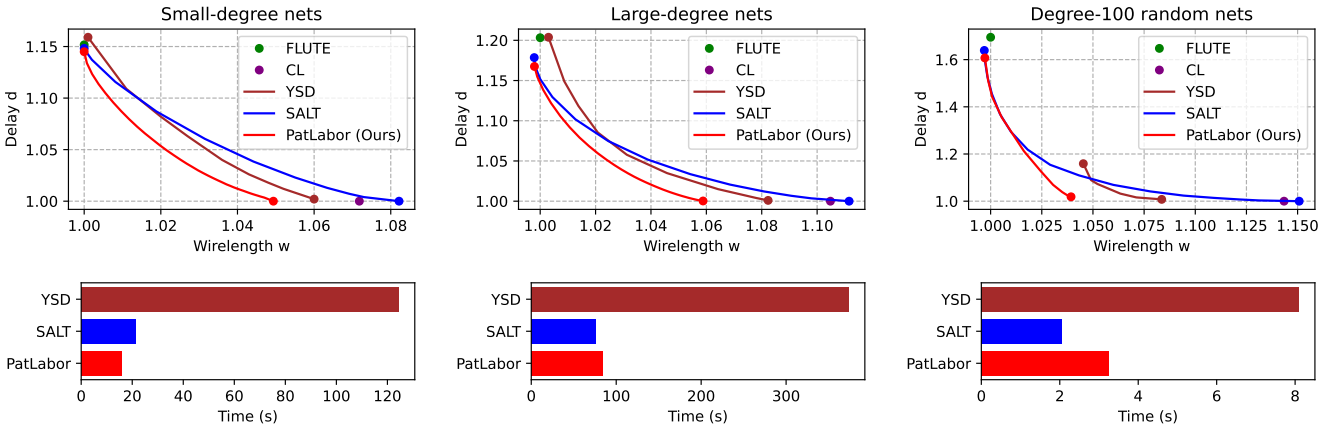
The analysis of Pareto frontiers in Section III is verified empirically. We obtain the number of solutions on the Pareto frontier. For all nets with degree $n \le 9$, the maximum size of the Pareto frontier is computed. The result is illustrated in Figure 6. Although the worst-case bound on the Pareto frontier size is exponential, the number of optimal solutions only grows roughly $\sim 2.85n$ on real-world testcases. For $n = 9$, the maximum Pareto frontier size is only 16. This verifies our analysis in Theorems 1 and 2, but also illustrates our bound is still slightly loose.

B. Generation of lookup tables

We present experimental results on generating lookup tables using the proposed Pareto-DW in Section V. Table II lists the statistics of our lookup tables. The lookup tables are generated on an Intel Xeon 2.30GHz CPU with 16 cores. The total number of tree topologies is about 1.7×10^8. Using 16 threads, we generate all lookup tables in less than 5 hours. The serial running time is 49.9 hours. As a rough comparison, FLUTE [4] only generates about 4.5×10^5 RSMT topologies[3] using 58.2 hours (on a 3.40GHz Intel Pentium 4 CPU). Our method is about $(1.7 \times 10^8)/(4.5 \times 10^5) \times (58.2/49.9) \approx 441\times$ on average faster than FLUTE for each topology.

Directly storing all topologies results in a huge table. We find that for a single set of pins with different sources, many topologies are the same. Moreover, some groups of pin sets have similar topologies with slight differences on the boundary pins. We group them into clusters and store only one topology for each cluster. This greatly reduces the table size. The tables can be read into memory in less than 5 seconds, occupying < 250MB memory space, which is negligible for routing millions of nets. It is also faster than loading neural network weights compared with YSD [6], which trains a model for each degree n and each weighted sum parameter.

[3]Our number of tree topologies is much larger than RSMT's since timing-driven routing trees (a) depend on the position of the source; (b) consider two optimization objectives.



(a) Small-degree nets ($n \leq 9$) (b) Large-degree nets ($n > 9$) (c) Degree-100 randomly generated nets

Fig. 7. Averaged Pareto curve and the total running time comparisons on (a) small-degree nets; (b) large-degree nets; (c) 100 randomly generated degree-100 nets. The green circle represents RSMT by FLUTE and the purple circle represents RSMA by CL. The wirelength w and the delay d are normalized by dividing $w(\text{FLUTE})$ and $d(\text{CL})$. Note that the Pareto curves for small-degree nets are averaged on non-optimal nets as in Table III.

TABLE III
THE RATIO OF NON-OPTIMAL NETS FOR $n \leq 9$.

n	#Net	PatLabor	YSD [6]	SALT [5]
4	364670	**0.0%**	**0.0%**	**0.0%**
5	256663	**0.0%**	0.3%	0.9%
6	103199	**0.0%**	7.8%	11.9%
7	75055	**0.0%**	23.3%	24.3%
8	42879	**0.0%**	36.0%	34.7%
9	62449	**0.0%**	49.5%	45.4%
Total	904915	**0.0%**	8.0%	8.4%

TABLE IV
THE TOTAL NUMBER OF SOLUTIONS ON THE PARETO FRONTIERS FOUND BY THREE METHODS FOR $n \leq 9$.

n	PatLabor	YSD [6]	SALT [5]
4	**364670**	**364670**	**364670**
5	**297636**	296634	295053
6	**137460**	128237	123590
7	**116941**	100012	92083
8	**77325**	49398	52767
9	**132487**	72105	77483
Total	**1.0**	0.898	0.893

C. Comparisons on Pareto curves

We implement PatLabor using C++ and compare it with two state-of-the-art timing-driven routing tree algorithms, SALT [5] and YSD [6]. As the open-source code of YSD is incomplete, we implement some components of YSD based on the paper. All experiments are executed on an Intel Xeon 2.30GHz CPU with 16 cores, except YSD, which uses a neural network model and is evaluated on an NVIDIA H100 GPU.

For **small-degree nets**, we first evaluate the ratio of non-optimal nets in Table III. An algorithm is non-optimal on a net if it cannot find at least one solution on the Pareto frontier. The usage of lookup tables ensures the optimality of PatLabor, while both YSD and SALT are non-optimal for $n \geq 5$. (We run YSD and SALT with different parameters to obtain Pareto sets.) Among 904,915 nets with degree $n \leq 9$, there are 72,393 (resp. 76,013) nets on which YSD (resp. SALT) cannot achieve the Pareto frontier. Specifically, for $n = 9$, there are 30,912 (resp. 28,352) out of 62,449 nets, on which YSD (resp. SALT) is non-optimal. In contrast, there are **zero** non-optimal nets for PatLabor. We also collect the total number of optimal solutions found by three methods for $n \leq 9$ in Table IV. There are 115,463 (resp. 120,873) out of 1,126,519 solutions on the Pareto frontier that YSD (resp. SALT) cannot find. Specifically, for $n = 9$, there are 60,382 (resp. 55,004) out of 132,487 solutions that YSD (resp. SALT) cannot find. PatLabor finds **all** Pareto-optimal solutions.

Figure 7(a) shows the Pareto curves of different methods for small-degree nets. Since the baselines achieve optimal Pareto curves on many small-degree nets, the curves are only averaged on **non-optimal** nets by SALT and YSD as in Table III (i.e., either SALT or YSD is non-optimal on these nets). PatLabor achieves the tightest Pareto curve and is the fastest method due to the use of lookup tables[4]. In total, PatLabor is about $1.35\times$ faster than SALT.

For **large-degree nets**, the result is presented in Figure 7(b). PatLabor also achieves the tightest Pareto curves. However, due to the running time of combining Pareto sets, PatLabor is about 11.6% slower than SALT, but still much faster than YSD. Note that most nets in the ICCAD-15 benchmark have less than 50 pins. To compare the performances of different methods comprehensively, we also randomly synthesize 100 nets with 100 pins, and evaluate all methods. The result is presented in Figure 7(c). The Pareto curve of PatLabor is almost the same as SALT's for low total wirelength w, but is tighter than SALT's for high total wirelength. Note that YSD uses a divide-and-conquer framework for large-degree nets, which performs poorly for wirelength minimization.

VII. CONCLUSION

In this work, we study the timing-driven routing tree problem. We first show the size of Pareto frontiers is only polynomial under the smoothed analysis setting. Then, we propose theoretical algorithms and a practical method, PatLabor, for timing-driven routing. Finally, experimental results show that PatLabor obtains tighter Pareto curves compared with state-of-the-art methods. A promising direction of future work is to extend our approach to other metrics of routing trees, such as congestion, multi-patterning lithography, and reliability. It would also be interesting if timing-driven routing trees could be integrated into global routing or even global placement engines.

[4]Note that YSD is much slower than SALT and PatLabor. Although it is claimed in the paper [6] that YSD is more efficient than SALT, their comparison is not fair. YSD uses batch inference on GPUs for the neural model and SALT is only evaluated serially. In fact, SALT (and PatLabor, of course) can be accelerated by multi-threading on the CPU.

REFERENCES

[1] I. Markov, "Limits on fundamental limits to computation," *Nature*, vol. 512, pp. 147–54, 2014.

[2] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu, and S. Venkatesh, "Prim-Dijkstra revisited: Achieving superior timing-driven routing trees," in *International Symposium on Physical Design*, 2018, p. 10–17.

[3] W. Li, R. Liang, A. Agnesina, H. Yang, C.-T. Ho, A. Rajaram, and H. Ren, "DGR: Differentiable global router," in *Design Automation Conference*, 2024.

[4] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2008.

[5] G. Chen and E. F. Y. Young, "SALT: Provably good routing topology by a novel Steiner shallow-light tree algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1217–1230, 2020.

[6] L. Yang, G. Sun, and H. Ding, "Towards timing-driven routing: An efficient learning based geometric approach," in *International Conference on Computer Aided Design*, 2023, pp. 1–9.

[7] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 30, no. 1, pp. 104–114, 1976.

[8] A. Kahng and G. Robins, "A new class of iterative Steiner tree heuristics with good performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 7, pp. 893–902, 1992.

[9] J. Liu, G. Chen, and E. F. Young, "REST: Constructing rectilinear Steiner minimum tree via reinforcement learning," in *58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 1135–1140.

[10] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, "The rectilinear Steiner arborescence problem," *Algorithmica*, vol. 7, no. 1–6, p. 277–288, 1992.

[11] J. Córdova and Y.-H. Lee, "A heuristic algorithm for the rectilinear Steiner arborescence problem," 1994.

[12] K.-S. Leung and J. Cong, "Fast optimal algorithms for the minimum rectilinear Steiner arborescence problem," in *1997 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 3, 1997, pp. 1568–1571.

[13] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-complete," *SIAM Journal of Applied Mathematics*, vol. 32, pp. 826–834, 1977.

[14] W. Shi and C. Su, "The rectilinear Steiner arborescence problem is NP-complete," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 729–740, 2005.

[15] M. Elkin and S. Solomon, "Steiner shallow-light trees are exponentially lighter than spanning ones," in *IEEE 52nd Annual Symposium on Foundations of Computer Science*, 2011, pp. 373–382.

[16] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh, and C. Wong, "Provably good performance-driven global routing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 6, pp. 739–752, 1992.

[17] S. Khuller, B. Raghavachari, and N. Young, "Balancing minimum spanning trees and shortest-path trees," *Algorithmica*, vol. 14, no. 4, p. 305–321, 1995.

[18] R. Scheifele, "Steiner trees with bounded RC-delay," *Algorithmica*, vol. 78, no. 1, p. 86–109, 2017.

[19] W. Li, Y. Qu, G. Chen, Y. Ma, and B. Yu, "TreeNet: Deep point cloud embedding for routing tree construction," in *Asia and South Pacific Design Automation Conference*, 2021, pp. 164–169.

[20] M. Hanan, "On Steiner's problem with rectilinear distance," *SIAM Journal on Applied Mathematics*, vol. 14, no. 2, pp. 255–265, 1966.

[21] A. Herzel, S. Ruzika, and C. Thielen, "Approximation methods for multiobjective optimization problems: A survey," *INFORMS Journal on Computing*, vol. 33, no. 4, pp. 1284–1299, 2021.

[22] D. A. Spielman and S.-H. Teng, "Smoothed analysis: an attempt to explain the behavior of algorithms in practice," *Communications of the ACM*, vol. 52, no. 10, p. 76–84, 2009.

[23] S. E. Dreyfus and R. A. Wagner, "The Steiner problem in graphs," *Networks*, vol. 1, no. 3, pp. 195–207, 1971.

[24] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, pp. 132–133, 1972.

[25] S. Arora, "Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems," *Journal of the ACM*, vol. 45, no. 5, p. 753–782, 1998.

[26] K. Kalpakis and A. T. Sherman, "Probabilistic analysis of an enhanced partitioning algorithm for the Steiner tree problem in $\mathbb{R}^d$," *Networks*, vol. 24, no. 3, pp. 147–159, 1994.

[27] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008, pp. 337–340.

[28] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*. Springer, 2015.

[29] D. Bertsekas, *Reinforcement Learning and Optimal Control*, 2019.

[30] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009, p. 41–48.

[31] R. Gupta and T. Roughgarden, "A PAC approach to application-specific algorithm selection," *SIAM Journal on Computing*, vol. 46, no. 3, pp. 992–1017, 2017.

FeKAN: Efficient Kolmogorov-Arnold Networks Accelerator Using FeFET-based CAM and LUT

Xuliang Yu, Yu Qian, Xunzhao Yin, Cheng Zhuo, and Liang Zhao*
College of Information Science and Electronic Engineering, Zhejiang University

***Abstract*—Kolmogorov-Arnold networks (KANs) have emerged as a promising alternative to MLP due to their adaptive learning capabilities for complex dependencies through B-spline basis activations (BBA). However, existing in-memory accelerators optimized for MLP-based DNNs are primarily designed for vector-matrix multiplication (VMM), making them inefficient for the dynamic and recursive B-spline interpolation (BSI) operations required by KANs. In this work, we propose FeKAN, an FeFET-based architecture designed to accelerate BBA operations. First, we develop a software-hardware co-optimized framework for mapping B-spline basis functions (BBF), leveraging a two-stage design space exploration (DSE) algorithm in combination with FeFET-based Look-Up Tables (LUT) and Content-Addressable Memory (CAM). This framework translated dynamic BSI operations into static codebook lookups, achieving a balanced trade-off between memory and computational efficiency. Second, we propose compress-sparsity-column (CSC) based encoding for B-spline basis function and grouped-computation strategy for memory and energy reduction. Third, we propose a grouped-pipeline optimization strategy to mitigate data dependencies, significantly enhancing computation efficiency. Experimental results demonstrate that FeKAN achieves up to $150.68\mathbf{K}\times$ and $4664\times$ higher throughput and up to $606.87\times$ and $11196\times$ greater energy efficiency over Intel Xeon Silver 4310 CPU and NVIDIA A6000 GPU, respectively.**

I. Introduction

The rapid advancement of artificial intelligence (AI) is profoundly transforming the world, with technologies like large language models (LLMs), autonomous driving and digital twins significantly redefining industries and enhancing user experience. Among the diverse deep learning architectures, MLP-based deep neural networks (DNNs [1], [2]) stand out for their ability to capture local features and extract high-level abstract information from sequences through their multi-layered structures. Despite their remarkable achievements, DNNs face significant limitations including fixed activation functions and large memory footprint, hindering their performance in high-dimensional data modeling and complex problem-solving. Recently, KAN [3] has emerged as a promising architecture for efficient data fitting through Bspline interpolation with trainable activation functions.

Research on accelerating KAN networks remains limited. In contrast, most efforts to accelerate MLP-based DNNs have primarily focused on sparse computation optimization and computational structure management to enhance computational efficiency. Sparse computation optimization involves pruning weakly-correlated neurons [4]–[6] and employing low-precision quantization [7], [8] to reduce redundant computation and memory footprint. Further improvements include designing sparse matrix multiplication units [9], [10] that compute only non-zero elements. In contrast, computational structure management focuses on balancing computational and memory resources through data-flow control mechanisms [11]–[13], multi-granular hardware parallelization [5], [14], and memory optimization techniques [15], [16]. Despite their impressive performance, these approaches are not directly applicable to KAN. This limitation arises as B-spline interpolation requires frequent retrieval of embeddings from off-chip DRAM to perform recursive computations, which are not well-suited for conventional DNN-based accelerators.

Most recently, compute-in-memory (CIM) has emerged as a promising non-von Neumann architectural approach to accelerate MLP-based DNNs with enhanced parallelism and energy efficiency. These architectures embed computations directly within the memory arrays, reducing the need for frequent off-chip data access and movement, thereby circumventing the memory bottleneck and improving both the computational and energy efficiency. CIM variants include SRAM-based digital CIM (DCIM) [17], [18], analog CIM (ACIM) [19], [20] and emerging non-volatile memory (eNVM) ACIM [21]–[25], among others. Currently, SRAM-DCIM and ACIM demonstrate higher accuracy than eNVM-ACIM in specific tasks. However, the larger area, higher standby power and inherent volatility of SRAM cells significantly limit their application in energy-constrained edge devices. In contrast, eNVMs such as RRAM, MRAM, PCM and FeFET can encode data into varying conductance states, enabling VMM operations to be mapped and processed in the analog domain through voltage/current transformations. For instance, ISSAC [26], PRIME [27], FELIX [28] and iCELIA [29] support energy-efficient VMM acceleration for DNNs using crossbar arrays of eNVM cells.

Although eNVM-ACIM designs offer substantial potential for accelerating DNNs, the dynamic order-associated interpolation operations within KAN are not well aligned with the static VMM nature of eNVM-ACIM. This misalignment poses significant challenges in terms of write endurance, cache bandwidth and read/write energy efficiency for KAN acceleration with eNVM. To illustrate this misalignment, Fig. 1(a) outlines the major challenge: **(1) BSI operations act as the bottleneck of KAN on existing hardwares.** The kth-order BBF require recursive and node-by-node interpolation, resulting in frequent memory access for intermediate results and excessive computation delays. Profiling reveals that BBA operations account for $91.31\% \sim 96.6\%$ of KAN inference time across various GPU platforms. In BBA computation, BSI contributes 65.6% of the total latency. To address the challenge of irregular and dynamic BSI operations, we transform the BBF into a static codebook and embed them into FeFET memory, leveraging the local support property within B-spline functions. Such practices transform the above challenge into two detailed challenges (Fig. 1 (b-c)): **(2) Multi-scale BSI curve mapping brings large memory footprint**. Different layers in KAN exhibit diverse data distributions, requiring the B-spline codebook mapping to adapt to multi-scale input ranges, sampling resolutions and quantization bit-widths for more precise approximation. Although the codebook memory footprint decreases with the reduced sampling points, the model precision also degrades significantly in extreme cases, necessitating a careful trade-off between the two. **(3) Non-local sparsity of B-spline functions causes redundant calculations and**

*Liang Zhao is the corresponding author (e-mail: lzhao2020@zju.edu.cn).
This work was supported by the National Science and Technology Major Project of the Ministry of Science and Technology of China (Grant No. 2024YFE0201100), the National Natural Science Foundation of China (Grant No. 62474162), and the Natural Science Foundation of Zhejiang Province (Grant No. LD25F040003).

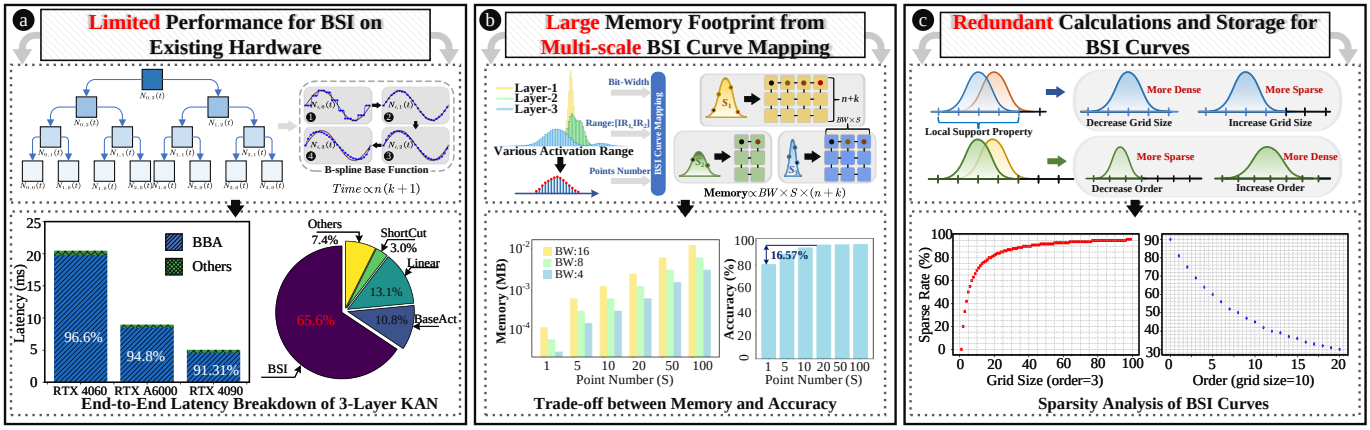


Fig. 1: Challenges in KAN inference: (a) Limited performance for BSI operations on existing hardware, (b) Large memory footprint from multi-scale BSI curve mapping and (c) Redundant calculations and storage for BSI curves.

storage. The BBF features non-local sparsity coupled with the order and control point number, which introduces computation redundancy in basis function weighting. When the order is fixed, the sparsity within the kth-order base function increases from zero to nearly 100% as the number of control points increases. Conversely, with a fixed number of control points, increasing order allows the kth-order base function to cover a broader range without sacrificing accuracy.

To resolve these challenges, we propose FeKAN, an FeFET-based architecture for KAN acceleration, which implements a software-hardware co-optimized B-spline mapping scheme to achieve memory-friendly and energy-efficient BBA computation. The key contributions of this work are summarized as follows:

- We propose a software-hardware co-optimized BBF mapping framework that combines the two-stage DSE algorithm and LUT-CAM mapping scheme, effectively balancing memory and computational efficiency.
- We propose a CSC-based encoding for BBF along with grouped computation strategy, effectively reducing the memory occupancy and energy consumption.
- we propose a grouped-pipeline optimization strategy to mitigate data dependencies within BSI computations, significantly enhancing computation efficiency.
- Experimental results confirm that FeKAN offers significant advantages over CPUs and GPUs for KAN acceleration, especially in terms of end-to-end latency, throughput and energy efficiency.

II. Background

Kolmogorov-Arnold Theorem: For arbitrary continuous multi-variable function $f(x_1, x_2, \ldots, x_n)$ on a bounded domain, there always exists a finite composition of continuous univariate functions ϕ_{ij} and ψ_i such that $f : [0,1]^n \to \mathbb{R}$ can be represented as a weighted sum of these functions, that is:

$$f(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{2n+1} \psi_i \left(\sum_{j=1}^{n} (\phi_{ij}(x_j)) \right) \tag{1}$$

Just as the universal approximation theorem underpins the development of artificial neural networks, this Kolmogorov-Arnold theorem serves as the theoretical foundation of the KAN algorithm. Moreover, unlike the universal approximation theorem, this theorem focuses on transforming high-dimensional functions into compositions of univariate functions to simplify the computation, providing significant advantages for the design of hierarchical network structures.

B-spline Interpolation (BSI): BSI is widely used in computational science as a flexible and efficient tool for approximating functions. BSI curves are generated by recursive linear combination of control point set and BBF, leveraging their local support property. This process typically involves two computational phases: BBF generation and BSI curve construction.

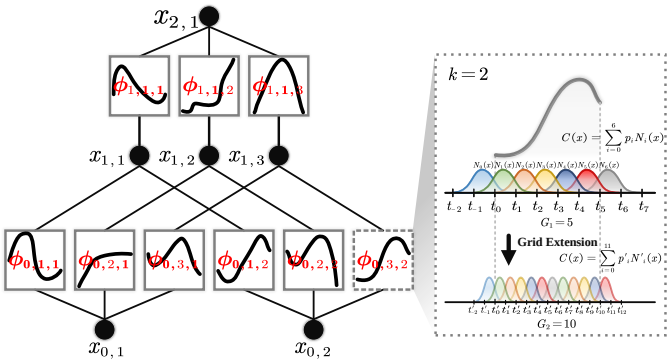


Fig. 2: KAN architecture and the mathematical principles of BSI.

• **B-spline Basis Function (BBF) Generation**. For a k-degree BBF with n control points, the k-degree basis functions are constructed iteratively, starting from zero-base function. Here, k determines the smoothness and polynomial degree of the basis functions, while the knot vector $\{t_0, t_1, \cdots, t_{n+k}\}$ defines the support interval and shape of the basis function curve. Specifically, the basis functions can be obtained through the following recursive procedure:

$$\begin{aligned} N_{i,0}(x) &= \begin{cases} 1, & \text{if } t_i \le x < t_{i+1} \\ 0, & \text{otherwise} \end{cases} \\ N_{i,k}(x) &= \frac{x - t_i}{t_{i+k} - t_i} N_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(x) \end{aligned} \tag{2}$$

where $N_{i,k}(x)$ represents the k-degree BBF associated with the ith control point for the input x.

• **B-spline Curve Construction**. Once the BBF are obtained, they can be linearly combined with the control points set $\mathbf{P} = \{p_1, \cdots, p_n\} \in \mathbb{R}^{n \times 1}$ to generate the B-spline interpolation results:

$$C(x_j) = \sum_{i=1}^{n} p_i N_{i,k}(x_j) \longleftrightarrow C = A \cdot P \tag{3}$$

where $\{x_j\}_{j=1}^{r}$ represent the interpolation points and $C(x_j) \subset C \in \mathbb{R}^{r \times 1}$ denotes the BSI values corresponding to $\{x_j\}_{j=1}^{r}$. $A \in \mathbb{R}^{r \times n}$ is the interpolation coefficient matrix, indicating the contribution of the ith control point to the jth interpolation point.

KAN Architecture: Inspired by the Kolmogorov-Arnold theorem, KAN incorporates a hierarchical activation mechanism by combining the SiLU$(\cdot)$ and B-Spline$(\cdot)$ functions. SiLU$(\cdot)$ serves as the base

activation (BaseAct), providing a smooth initial state, while the B-spline activation enhances adaptability with its flexible shape and local adjustment capability. As shown in Fig. 2, M-layer KAN can be recursively expanded into a composition of functions:

$$\begin{aligned} \text{KAN}(x) &= (\psi_{M-1} \circ \psi_{L-2} \circ \cdots \psi_0)(x), \psi = \{\phi_{i,j}\} \\ \phi_{i,j} &= \mathbf{w_b} b(x) + \mathbf{w_s}\text{spline}(x), b(x) = \text{SiLU}(x) \end{aligned} \tag{4}$$

where spline(x) can be parameterized as a linear combination of BBF with respect to the control points set $\mathbf{P}$ in Eq. 3. The weight $\mathbf{w_b}$, $\mathbf{w_s}$ and the control points $\mathbf{p_i}$s can be trained for different tasks to adjust the activation function's response over various input intervals.

FeFET and Its In-Memory Computing Applications: compared to other eNVMs [30]–[32], FeFET devices offer some unique advantages for CIM including the voltage-driven mechanisms, high ON/OFF ratio and very low OFF current associated with the three-terminal FET structure [33]. As a typical use case, FeFET-based CAM accelerates parallel associative search operations throughout the entire memory array, demonstrating promising speed and energy efficiency that is difficult to achieve with other two-terminal eNVMs [34], [35]. In addition, FeFET crossbar arrays can store weight matrices and perform parallel VMM to accelerate neural networks [36] and accelerate combinatorial optimization [37]–[39].

III. Algorithm

A. Two-stage Design Space Exploration

Translating the dynamic BBF into a static codebook offers an optimal space-time trade-off. Specifically, we store the input activations and their corresponding B-spline values in memory as dictionary pairs. For each query, the key is matched in the codebook, and the corresponding value is returned as BBF. Although this method effectively improves computational efficiency and reduces memory access overhead, the pursuit of higher inference performance and memory efficiency remains constrained by several factors, including the input range, number of sampling points and quantization precision. These factors are elaborated as follows:

- The distribution of input activations across different layers of KAN is affected by the network structure and the specific transduction tasks, making it difficult to establish a universal and widely applicable codebook that can approximate the true data distribution. This, in turn, impacts the inference accuracy of KAN based on the codebook.
- The number of sampling points and input range jointly affect the resolution of the codebook. Too few sampling points result in insufficient discretization accuracy in the input space, making it impossible to map the input data correctly and causing B-spline mapping distortion. Conversely, over-sampling increases memory and computational overhead during the query-key matching.
- Model quantization leads to flattening and overlapping effects in the values of BBF. Thus, neighboring B-spline data points lose the ability to distinguish differences in the original data, resulting in the same quantized value after mapping. This fails to accurately capture the nonlinear activation characteristics of the BBF, which in turn degrades the model accuracy.

To mitigate these risks, we propose a two-stage design space exploration algorithm (see Algorithm 1) to optimize the trade-off between model performance and memory load, which is influenced by parameters such as the input range, number of sampling points and quantization bit-width. The goal is to construct an efficient codebook that simultaneously achieves high fitting accuracy and low memory load. In the first stage, we update the knot vector $\{t_i\}_{i=0}^{n+k}$ using the predefined control points number n, spline degree k, and grid intervals gr. Based on the input range ir and the number of sampling points s, we generate a candidate input query set $\mathcal{Q}$. Then, we acquire the shared codebook using the BBF computation method (Eq. 2), which takes the candidate set $\mathcal{Q}$ and the knot vector$\{t_i\}_{i=0}^{n+k}$ as inputs. Afterward, we refine the parameter search space by evaluating the model task and applying a fitness function reward. This aims to determine the optimal set of parameters that balance both codebook storage requirements and model accuracy. Finally, a two-stage finer search further optimizes the parameters near the solution obtained from the previous coarse search, using a finer step size τ_{fine}.

Algorithm 1 Two-stage Design Space Exploration (DSE) Algorithm

Input: Base-model, Transduction-Task, ϵ, Input Range Set $\{IR\}$, Sampling Point Set $\{\mathcal{S}\}$, Bit-Width Set $\{BW\}$, Search Grid $\{\tau_{\text{coarse}}, \tau_{\text{fine}}\}$, BSI-degree k, knot-vector range $\{gr\}$ and control points n.
Output: Best Parameter Set $\{IR_{\text{best}}, \mathcal{S}_{\text{best}}, BW_{\text{best}}\}$, Shared BSI Codebook.

```
1: Initialize Best_Score ← 0
2: for layer in layers do
3:   IR_coarse, S_coarse, BW_coarse ← {None, None, None}
4:   for {bw, s, ir} in {BW_coarse, S_coarse, IR_coarse} step τ_coarse do
5:     knot_vector ← Generate_Knot_Vector(gr, k, n, bw)
6:     {Keys} ← Input_Sampling(ir, s, bw)
7:     BSI-Codebook ← Compute_Bspline(knot_vector, Keys, k, bw)
8:     Model ← Update(Base-model, BSI-Codebook, knot_vector)
9:     Accuracy ← Evaluate(Model, Transduction-Task)
10:    if abs(Accuracy − Base-accuracy) ≤ ε then
11:      Score ← Fitness(Accuracy, bw, s, k, n)
12:      if Score > Best_Score then
13:        IR_coarse, S_coarse, BW_coarse ← {s, bw, ir}
14:        Best_Score ← Score
15:      end if
16:    end if
17:  end for
18: end for
19: Repeat Lines 2-18 with {IR_coarse, S_coarse, BW_coarse} for finer search.
```

Fitness$(\cdot)$:= $\alpha \times$ Accuracy $+ \beta \times (bw \times s \times k \times n)$

B. GQA-LUT-based SiLU Approximation

The BaseAct (e.g. SiLU$(\cdot)$) is introduced as residual nonlinear term in KAN to enable the learning of more complex decision boundaries and data distributions. However, the computation of nonlinear functions involves complex data-flow transformations, which are not well-suited to the CIM-based integer computation architecture. Hence, an eNVM-friendly method for nonlinear computation is highly desirable.

Due to the linear inseparability of nonlinear functions, directly estimating scaling factors and quantizing nonlinear functions can distort the original data distribution, leading to a degradation of model accuracy and learning capability. Piecewise linear approximation (pwl) decomposes nonlinear functions into simpler data segments within specific intervals [40], making it a promising candidate for accelerating B-splines. Based on the $pwl(\cdot)$ method, we can convert the SiLU$(\cdot)$ calculation into a series of linear mapping expressions represented by N entries $pwl_i(x)_{i=0}^{N-1}$, as shown below:

$$pwl(x) = \begin{cases} k_0 x + b_0, & \text{if } x < p_0, \\ k_1 x + b_1, & \text{if } p_0 \le x < p_1, \\ \quad\vdots & \\ k_{N-1} x + b_{N-1}, & \text{if } x \ge p_{N-2}. \end{cases} \tag{5}$$

in which the breakpoints $p_i{}_{i=0}^{N-1}$ determine the interval location of each entry, while the slope k_i and intercept b_i define the mapping from the input to $pwl(x)$.

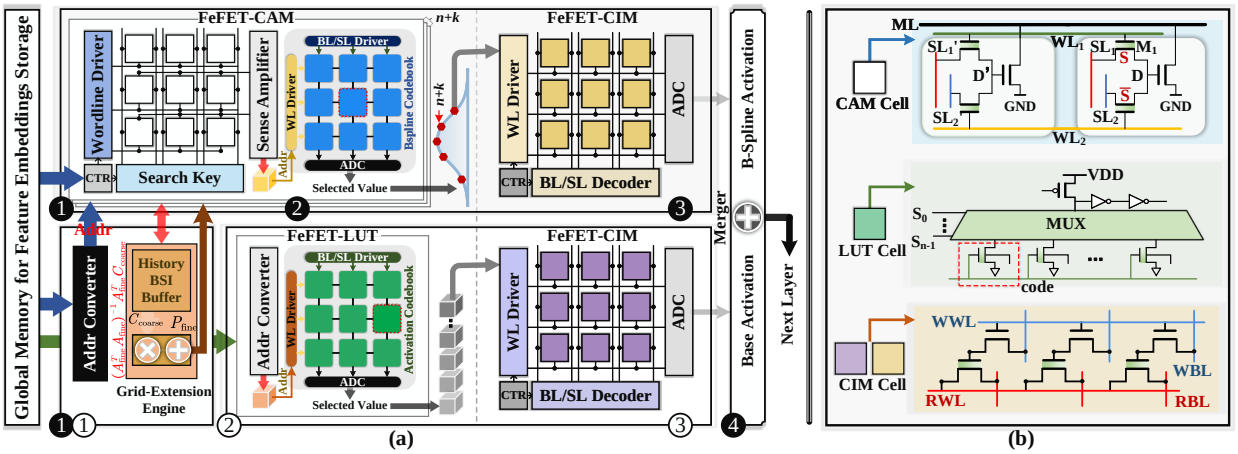


Fig. 3: (a)FeKAN accelerator architecture (BSI Dataflow: ❶–❹, BaseAct Dataflow: ①–④); (b) CAM/LUT/CIM cell structure.

To improve the nonlinear computation efficiency of SiLU on the FeFET-based CIM architecture, we adopt the genetic piecewise linear approximation method from Genetic Quantization-Aware LUT-Approximation (GQA-LUT) [41]. The breakpoint set $\mathcal{O} = \{\mathcal{P}_\prime, \mathcal{P}1, \cdots, \mathcal{P}_{N_p-1}\}$ is randomly initialized within the interval $[R_n, R_p]$. During each evolution phase of the genetic algorithm, the approximation result is evaluated by calculating the mean square error between the nonlinear function and the piecewise linear approximation. The crossover operation combines the breakpoints of two individuals to generate new candidates, while the mutation operation adjusts the breakpoint locations to explore new solution spaces. A 3-size tournament selection is applied to choose the individuals with higher fitness for the next generation. The final output of the best slope, intercept and breakpoint candidates can be used to construct the piecewise asymptotic scheme for SiLU, effectively approximating the non-asymptotic function.

IV. Architecture

A. Overall Architecture

Fig. 3 shows the overall architecture of FeKAN, which integrates FeFET-based CAM, LUT and CIM computation macros to support both BSI and BaseAct operations. To better elucidate the two dataflows within the architecture, we illustrate this by using the BSI computation as an example: ❶ Feature embeddings are first loaded on-chip. Then, the BBF mapping is implemented in either the CAM or LUT computation unit based on the configured mode. The CAM adopts a query-key matching approach to generate position-selection signals, while the LUT employs a content-address mapping list for address conversion. ❷ Upon receiving the matched signal from the CAM or LUT, the B-spline codebook group, with a depth of $n+k$, reads out the corresponding B-spline mapping values and column indices of the inputs, facilitating subsequent sparse computations. ❸ The BBF results are weighted by the control point matrix $\mathbb{P}$, which is embedded in the FeFET-CIM array, to generate the final BSI results. ❹ Finally, the BSI and BaseAct results are merged by the Merger engine, after which the generated activation information is forwarded to the next layer. The BaseAct follows the same computation manner as BSI. Besides, we incorporate an on-chip grid-extension engine to support fine-grained BSI operations.

B. FeFET-based CAM and LUT

Following the two-stage DSE algorithm, we obtain a shared static codebook for the input queries $\mathcal{Q}_{i=0}^{M}$ across all layers. This codebook can then be embedded into memory for BBF mapping of feature embeddings. In this work, we used FeFET-based CAM and LUT

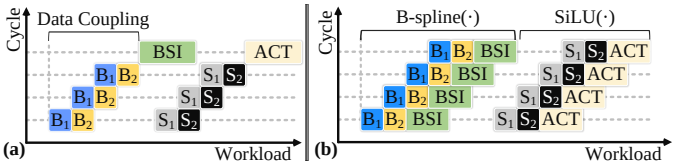


Fig. 4: (a) Raw computation stack and (b) pipeline optimization.

to implement the query-key primitive in both BBF and BaseAct, as depicted on the left top of Fig. 3(b).

We employed a 2FeFET-1T array [35] for CAM. Each row shares the same wordlines (WLs) and matchline (ML), and each column shares the same searchlines (SLs) within the array. The MLs are connected to a sense amplifier for output sensing. The schematic of the 2FeFET-1T CAM cell is illustrated on the top of Fig. 3(b), where each cell consists of a pair of FeFETs and an nMOS discharge transistor. Before search operation, each bit of the word in the codebook is encoded and written into the cell as the V_{th} of FeFETs. During search, each bit in the search key is encoded as voltages applied to SLs. The voltage of node D will be high/low enough to turn on/off the nMOS for the mismatch/match case, respectively. Upon a search, only the row where all cells are matched remains a high voltage of ML, which is sensed by the sense amplifier for further data fetch. For FeFET-LUT, we use a multiplexer formed by n-bit inputs to decode the input address into row/column select signals [42], which are then used to access and read specific FeFET cells.

C. CSC-based Encoding and Grouped-Computation

To address the non-local sparsity introduced by BBF, we propose a CSC-based encoding and grouped-computation strategy for compact BSI computation. As illustrated in Fig. 5(a), the raw B-spline codebook is represented as a structured 2D grid, where the column vectors correspond to the sampled points along the input distribution, and the row vectors represent the interpolation points derived from various BBFs. During encoding, arbitrary feature point x within row-vector can be sequentially extracted for zero-detection and unfolding. For each non-zero element, a column address is assigned to identify its exact location in the B-spline matrix, and the corresponding {column index, B-spline(x)} pair is generated. This pair is then written into the address and content buffers, providing efficient input support for subsequent computations. Based on this encoding scheme, the codebook memory footprint is reduced by a factor of $\frac{n+k}{k'}$, where k' is less than $k+1$.

Furthermore, we introduce a grouped-computation strategy, which partitions the B-spline set $\in \mathbb{R}^{F\times(n+k)}$ into F distinct groups,

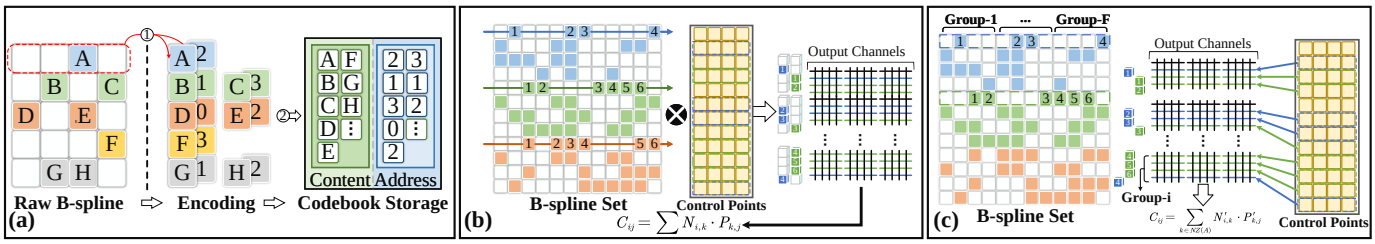


Fig. 5: (a) CSC-based sparse encoding, (b) conventional computation and (c) optimized sparse grouped-computation.

where F denotes feature dimension. Each group corresponds to the BBF associated with a specific feature point. Within each group, local weighted calculations are carried out using the interpolation coefficient matrix $N_{i,k}$ in conjunction with the control points $P_{k,j}$. By leveraging the encoding method for BBFs discussed earlier, we can efficiently compute only the non-zero entries, denoted as $NZ(A)$ using Eq. 6, thereby further optimizing computational efficiency.

$$C_{i,j} = \sum_{k} N_{i,k} \cdot P_{k,j} = \sum_{(k)\in NZ(A)} N'_{i,k} \cdot P'_{k,j}, \quad NZ(A) = \{k : N_{i,k} \neq 0 \text{ and } P_{k,j} \neq 0\} \tag{6}$$

D. Pipeline Optimization

In the BBA computation process, both the B-spline activation function and the basic nonlinear activation functions have parallelization potential. However, due to the data coupling in large array computation tasks, the parallelization capability has not been fully realized. For example, the generation of B-spline curves depends on the preparation and computation of BBFs in phases B1 and B2 (see Fig. 4 (a)). In this mode, the output of each phase must wait for all preceding tasks to complete, and this dependency causes resource idling, significantly limiting computational efficiency.

To address this issue, we propose an optimization method based on grouped pipelining. By tiling the control point matrix of the raw large array, we decompose the complex matrix operations into fine-grained single-element basis function weighting computations, achieving higher parallelism. At the same time, by grouping the basis functions and processing them layer by layer in a pipelined manner, we effectively reduce phase dependencies and decrease the waiting time between tasks. This method significantly improves overall computational efficiency and provides a new approach for parallel optimization in long feature sequence inference tasks.

E. Grid Extension Engine

Considering the local support and piecewise polynomial structure of BBFs, the B-spline function can be refined through grid extension. This allows it to more precisely to capture complex features and subtle variations. Suppose P_{coarse} and A_{coarse} represent the original control point matrix and interpolation coefficient matrix, these matrices will become P_{fine} and A_{fine} after grid refinement. To ensure alignment between the refined spline and the original, we can formulate a least-squares objective as follows:

$$\min_{\mathbb{P}_{\text{fine}}} \|A_{\text{fine}} \cdot P_{\text{fine}} - C_{\text{coarse}}\|_2^2 \tag{7}$$

where the solution to Eq. 7 can be expressed as follows:

$$P_{\text{fine}} = (A_{\text{fine}}^T A_{\text{fine}})^{-1} A_{\text{fine}}^T C_{\text{coarse}} \tag{8}$$

To achieve off-line grid refinement, an on-chip digital multiplier-accumulator circuit is used together with the history BSI buffer storing C_{coarse} to update the control point weights P_{fine}.

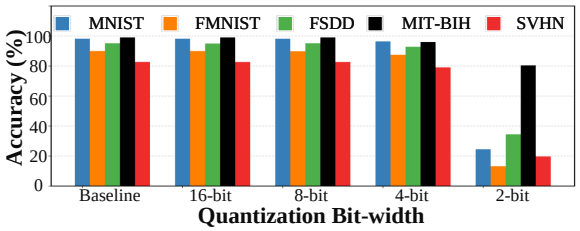


Fig. 6: Impact of Quantization Bit-width on Model Accuracy.

TABLE I: Parameter Configurations for KAN

Model	order k	control points n	layer configurations
KAN-I	3	10	[784,256,64,10]
KAN-II	5	10	[300, 64, 5]
KAN-III	5	10	[3072, 512, 128, 10]

V. Evaluation

A. Evaluation Methodology

We evaluated the performance of FeKAN using three different KAN models across open-source benchmark tasks spanning three modalities: image classification (MNIST, FMNIST, SVHN), speech recognition (FSDD), and electrocardiogram (ECG) sequence analysis (MIT-BIH). The specific model parameters are listed in Table I, where MNIST, FMNIST, and FSDD are evaluated using KAN-I. We employed a two-stage design space exploration algorithm with $\tau_{coarse} = 1$ and $\tau_{fine} = 0.1$ to identify the optimal trade-offs between accuracy and memory usage across different tasks. This process involved sampling the best points for bit-width quantization and input range sets.

The performances of FeFET-CAM and LUT were validated using Cadence Virtuoso based on a 45nm FeFET model calibrated with experimental data [35] [42]. Power consumption and latency of the designs were estimated through Spectre simulations. For the weighted operations of BBA, we simulated the FeFET-CIM performances in NeuroSim V1.4 [43] at the 32nm process node, which implements the BSI and residual activation function SiLU(·).

Additionally, to demonstrate the acceleration effectiveness of the BBA within FeKAN, we evaluated power consumption and latency on Intel Xeon Silver 4310 CPUs, and Nvidia RTX 4060, 4090, and A6000 GPUs. Latency for both GPU and CPU was measured using CUDA Events and Python Timers, while power consumption was monitored using Nvidia-smi and PyRAPL.

B. Algorithm Performance

Impact of Quantization Bit-Width. We first evaluated how varying the quantization precision from 2 to 16 bits impacts the performance of the KAN model across different benchmark tasks, as shown in Fig. 6. The KAN model demonstrates strong quantization tolerance across various tasks. Specifically, when the quantization precision is reduced to 8 bits, the model maintains performance comparable to the baseline. However, when the quantization precision drops below 4 bits, a sharp decline in performance is observed. This degradation

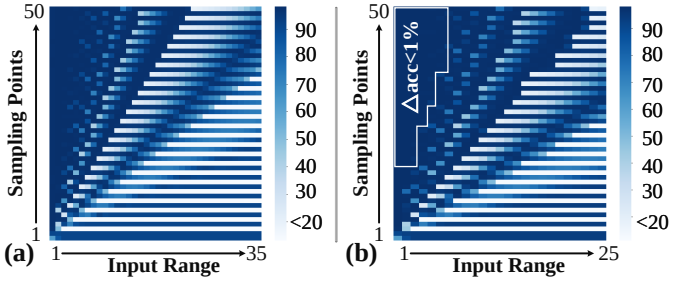


Fig. 7: Search results visualization: (a) BW = 32-bit, (b) BW = 8-bit.

is attributed to the inability of low-bit quantization to fully capture the data distribution of the dataset.

DSE Analysis. We visualized the accuracy distribution of the KAN-I model under quantization precisions of 16 and 8 bits, across different input ranges and sampling point numbers. The accuracy of the KAN-I model exhibits a block-like variation in the input range, sampling points parameter space. Specifically, when the number of sampling points is fixed and the input range is large, or when the input range is fixed and the number of sampling points is small, most real activation values are mapped to the same value range, resulting in the loss of fine-grained information and, consequently, a decrease in model performance. Therefore, increasing the number of sampling points within a given input range effectively maintains model accuracy. The parameter space with $\Delta_{acc} < 1\%$, indicating an inference accuracy loss relative to the baseline, is highlighted. Based on this analysis, the optimal parameter space for this task is identified as an input range of {-1, +1} and 10 sampling points.

C. Hardware Performance

End-to-End SpeedUp and Energy Efficiency. Fig. 8 presents the end-to-end speedup and energy efficiency improvements of FeKAN compared to CPUs across different computation platforms. FeKAN-CAM achieves up to 2.74× the speedup of FeKAN-LUT. FeKAN-CAM achieves a maximum speedup of 3620.41× and 2303.32× over the RTX 4090 and A6000, respectively, across five tasks, while FeKAN-LUT can achieve a maximum speedup of 54943× over the CPU. Two key factors contribute to these results: first, the B-spline interpolation mapping based on CAM and LUT effectively bypasses dynamic recursive interpolation calculations, shortening the computational path length associated with order coupling; second, grouped pipelined operations decouple the B-spline curve construction from the basis function generation process, enhancing parallelism.

Regarding energy efficiency, FeKAN-LUT achieves the highest performance improvement, with up to 606.87× better energy efficiency compared to the CPU, while FeKAN-CAM achieves a 304.17× improvement in energy efficiency. In comparison to A6000 and RTX 4060 GPUs, FeKAN-CAM demonstrates energy efficiency improvements of 207.19/5612.01× and 9.16/154.78×, respectively. The exceptional energy efficiency of FeKAN can be attributed to its CIM architecture which alleviates the von-Neumann bottleneck, as well as computational optimizations leveraging sparse encoding.

Ablation Study. Fig. 9 shows the latency and energy breakdown of three KAN models (KAN-I, KAN-II, and KAN-III) on FeKAN when sparse optimization and pipelining optimization strategies are applied. The experimental results indicate that the KAN-I and KAN-II models significantly reduce the B-spline CIM computation time through the pipelining optimization strategy, with a compression factor ranging from 6.86× to 7.43×. This demonstrates that the pipelining optimization strategy successfully avoids phase-dependent

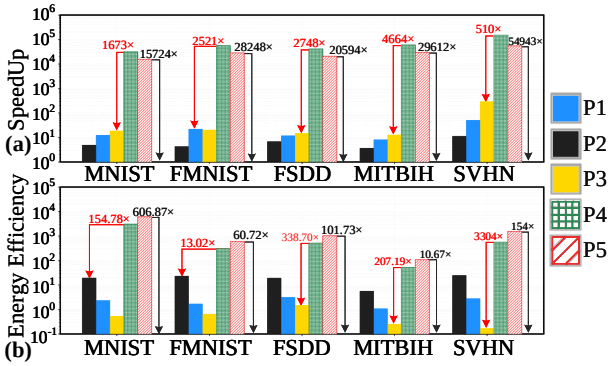


Fig. 8: Comparison of Latency and Energy Efficiency: FeKAN vs. RTX 4060/4090/A6000 GPUs (P1-P3) and Intel Xeon Silver 4310 CPU, with P4 and P5 Representing FeKAN-CAM and FeKAN-LUT.

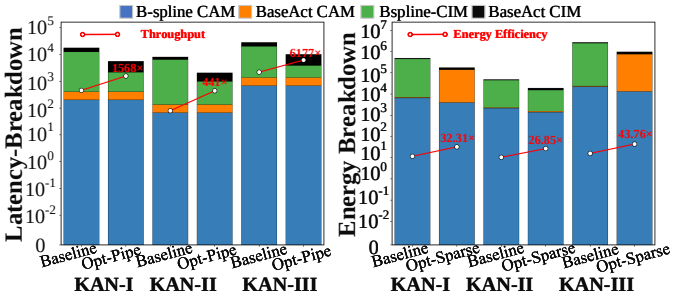


Fig. 9: Latency and energy breakdown of FeKAN with sparse and pipeline optimizations versus baseline.

data waiting by stacking data along the time dimension, thereby improving overall computational efficiency.

Furthermore, with the application of the grouped pipelining optimization strategy, KAN-III achieves a throughput of 6177 GOPS, representing a 2.82× improvement over the baseline model. Regarding energy breakdown, the sparse optimization strategy significantly reduces the energy consumption of CAM and CIM during query-key matching and BBF weighting. Specifically, KAN-III and KAN-I achieve energy efficiencies of 43.76 TOPS/W and 32.31 TOPS/W, respectively. This performance improvement can be attributed to the sparse encoding and grouped-computation strategy. By reducing redundant weighting calculations and storage requirements in the BBFs, this strategy greatly reduces computational complexity, leading to lower power consumption and higher energy efficiency.

VI. Conclusion

In this work, we propose FeKAN, an FeFET-based in-memory accelerator designed to address the unique challenges of accelerating Kolmogorov-Arnold Networks (KANs) through efficient and memory-friendly B-spline basis activation computations. FeKAN incorporates a software-hardware co-optimized framework that combines a two-stage design space exploration (DSE) algorithm with a LUT-CAM mapping scheme, effectively balancing memory overhead and interpolation efficiency. To further reduce memory usage and redundant computation, we introduce a CSC-based encoding for BBF compression and a grouped-computation strategy for compact and efficient processing. Additionally, we propose a grouped-pipeline optimization strategy to improve computational efficiency and throughput. Experimental results demonstrate FeKAN's superiority over conventional CPUs and GPUs, achieving significant reductions in end-to-end latency while delivering substantial gains in throughput and energy efficiency for KAN acceleration.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[2] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2016.

[3] Z. Liu, Y. Wang, S. Vaidya *et al.*, "KAN: Kolmogorov-Arnold Networks," 2024, arXiv:2404.19756.

[4] S. Han, X. Liu, H. Mao *et al.*, "EIE: Efficient inference engine on compressed deep neural network," in *Proc. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 243–254.

[5] S. Zhang, Z. Du, L. Zhang *et al.*, "Cambricon-X: An Accelerator for Sparse Neural Networks," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2016, pp. 1–12.

[6] J.-F. Zhang, C.-E. Lee, C. Liu *et al.*, "SNAP: An efficient sparse neural acceleration processor for unstructured sparse deep neural network inference," *IEEE J. Solid-State Circuits (JSSC)*, vol. 56, no. 2, pp. 636–647, 2021.

[7] K. Wang, Z. Liu, Y. Lin *et al.*, "HAQ: Hardware-Aware Automated Quantization With Mixed Precision," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 8604–8612.

[8] D. Zhang, J. Yang, D. Ye *et al.*, "Lq-Nets: Learned Quantization for Highly Accurate and Compact Deep Neural Networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 365–382.

[9] N. Srivastava, H. Jin, J. Liu *et al.*, "MatRaptor: A Sparse-Sparse Matrix Multiplication Accelerator Based on Row-Wise Product," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2020, pp. 766–780.

[10] S. Pal, J. Beaumont, D.-H. Park *et al.*, "OuterSPACE: An Outer Product Based Sparse Matrix Multiplication Accelerator," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2018, pp. 724–736.

[11] Y.-H. Chen, T. Krishna, J. Emer *et al.*, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

[12] A. Parashar, M. Rhu, A. Mukkara *et al.*, "SCNN: An accelerator for compressed-sparse convolutional neural networks," *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 27–40, 2017.

[13] W. Lu, G. Yan, J. Li *et al.*, "FlexFlow: A flexible dataflow accelerator architecture for convolutional neural networks," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2017, pp. 553–564.

[14] Y. Chen, T. Luo, S. Liu *et al.*, "DaDianNao: A Machine-Learning Supercomputer," in *Proc. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2014, pp. 609–622.

[15] J. Li, S. Jiang, S. Gong *et al.*, "SqueezeFlow: A sparse CNN accelerator exploiting concise convolution rules," *IEEE Trans. Comput. (TC)*, vol. 68, no. 11, pp. 1663–1677, 2019.

[16] V. Sze, Y.-H. Chen, T.-J. Yang *et al.*, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[17] M.-E. Shih, S.-W. Hsieh, P.-Y. Tsai *et al.*, "NVE: A 3nm 23.2 TOPS/W 12b-digital-cim-based neural engine for high-resolution visual-quality enhancement on smart devices," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 67, 2024, pp. 360–362.

[18] H. Fujiwara, H. Mori, W.-C. Zhao *et al.*, "A 5-nm 254-TOPS/W 221-TOPS/mm2 fully-digital computing-in-memory macro supporting wide-range dynamic-voltage-frequency scaling and simultaneous MAC and write operations," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, vol. 65, 2022, pp. 1–3.

[19] X. Si, Y.-N. Tu, W.-H. Huang *et al.*, "A 28nm 64Kb 6T SRAM computing-in-memory macro with 8b MAC operation for AI edge chips," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 246–248.

[20] J.-W. Su, X. Si, Y.-C. Chou *et al.*, "A 28nm 64Kb inference-training two-way transpose multibit 6T SRAM compute-in-memory macro for AI edge chips," in *IEEE Int. Solid-State Circuits Conf. (ISSCC)*, 2020, pp. 240–242.

[21] C. Matsui, K. Toprasertpong, S. Takagi *et al.*, "FeFET local multiply and global accumulate voltage-sensing computation-in-memory circuit design for neuromorphic computing," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 32, no. 3, pp. 468–479, 2024.

[22] B. Li, P. Gu, Y. Shan *et al.*, "RRAM-based analog approximate computing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 34, no. 12, pp. 1905–1917, 2015.

[23] S. Angizi, Z. He, A. Awad *et al.*, "MRIMA: An MRAM-based in-memory accelerator," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 39, no. 5, pp. 1123–1136, 2020.

[24] V. Joshi, M. Le Gallo, S. Haefeli *et al.*, "Accurate deep neural network inference using computational phase-change memory," *Nat. Commun.*, vol. 11, no. 1, p. 2473, 2020.

[25] X. Yin, X. Chen, M. Niemier *et al.*, "Ferroelectric FETs-based nonvolatile logic-in-memory circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, pp. 159–172, 2019.

[26] A. Shafiee, A. Nag, N. Muralimanohar *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 14–26.

[27] P. Chi, S. Li, C. Xu *et al.*, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. ACM/IEEE Annu. Int. Symp. Comput. Archit. (ISCA)*, 2016, pp. 27–39.

[28] T. Soliman, N. Laleni, T. Kirchner *et al.*, "FELIX: A ferroelectric FET based low power mixed-signal in-memory architecture for DNN acceleration," *ACM Trans. Embed. Comput. Syst. (TECS)*, vol. 21, no. 6, pp. 84:1–84:25, 2022.

[29] H. Yan, H. R. Cherian, E. C. Ahn *et al.*, "iCELIA: A full-stack framework for STT-MRAM-based deep learning acceleration," *IEEE Trans. Parallel Distrib. Syst. (TPDS)*, vol. 31, no. 2, pp. 408–422, 2020.

[30] Y. Qian, L. Zhao, F. Meng *et al.*, "Enhancing convnets with convfifo: A crossbar pim architecture based on kernel-stationary first-in-first-out dataflow," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, no. 01, pp. 1640–1651, 2024.

[31] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nat. Electron.*, vol. 1, pp. 333–343, 2018.

[32] C. Zhuo, Z. Yang, K. Ni *et al.*, "Design of ultra-compact content addressable memory exploiting 1T-1MTJ cell," *Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, 2022.

[33] K. Ni, X. Yin, A. F. Laguna *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nat. Electron.*, vol. 2, no. 11, pp. 521–529, 2019.

[34] X. Yin, Y. Qian, M. Imani *et al.*, "Ferroelectric ternary content addressable memories for energy-efficient associative search," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. (TCAD)*, vol. 42, no. 4, pp. 1099–1112, 2022.

[35] Y. Qian, Z. Fan, H. Wang *et al.*, "Energy-aware designs of ferroelectric ternary content addressable memory," in *Proc. Des. Autom. Test Eur. Conf. Exhib. (DATE)*, 2021, pp. 1090–1095.

[36] Z. Yang, Q. Huang, Y. Qian *et al.*, "Energy efficient dual designs of FeFET-based analog in-memory computing with inherent shift-add capability," in *Proc. 61st ACM/IEEE Des. Autom. Conf. (DAC)*, 2024, pp. 1–6.

[37] X. Yin, Y. Qian, A. Vardar *et al.*, "Ferroelectric compute-in-memory annealer for combinatorial optimization problems," *Nat. Commun.*, vol. 15, no. 1, p. 2419, 2024.

[38] Y. Qian, K. Ni, T. Kampfe *et al.*, "C-Nash: A novel ferroelectric computing-in-memory architecture for solving mixed strategy Nash equilibrium," in *Proc. 61st ACM/IEEE Des. Autom. Conf. (DAC)*, 2024, pp. 1–6.

[39] Y. Qian, Z. Yang, K. Ni *et al.*, "HyCiM: A hybrid computing-in-memory qubo solver for general combinatorial optimization problems with inequality constraints," in *Proc. 61st ACM/IEEE Des. Autom. Conf. (DAC)*, 2024, pp. 1–6.

[40] J. Yu, J. Park, S. Park *et al.*, "NN-LUT: Neural approximation of non-linear operations for efficient transformer inference," in *Proc. 59th ACM/IEEE Des. Autom. Conf.*, 2022.

[41] P. Dong, Y. Tan, D. Zhang *et al.*, "Genetic quantization-aware approximation for non-linear operations in transformers," in *Proc. 61st ACM/IEEE Des. Autom. Conf.*, 2024.

[42] Y. Xu, Z. Zhao, Y. Xiao *et al.*, "Ferroelectric FET-based context-switching fpga enabling dynamic reconfiguration for adaptive deep learning machines," *Sci. Adv.*, vol. 10, no. 3, 2024.

[43] X. Peng, S. Huang, Y. Luo *et al.*, "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," in *IEEE International Electron Devices Meeting (IEDM)*, 2019.

DANN: Diffractive Acoustic Neural Network for in-sensor computing system target at multi-biomarker diagnosis

Lewei He[1,2], Ning Lin[1], Binbin Cui[1], Xinran Zhang[1], Shiming Zhang[1,†], Zhongrui Wang[2,†]
[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China;
[2]School of Microelectronics, Southern University of Science and Technology, Shenzhen, China;
[†]Corresponding author: wangzr@sustech.edu.cn, szhang@eee.hku.hk.

***Abstract*—Analog machine learning hardware platforms, such as those using wave physics, present potential for edge artificial intelligence (AI) applications due to in-sensor computing architecture, offering superior energy efficiency compared to digital circuits. While the diffractive neural network has been implemented in optical systems, its deployment on integrated acoustic systems has not been achieved due to the challenges associated with hardware optimization. In this paper, we propose the Diffractive Acoustic Neural Network (DANN), a novel approach that applies diffractive neural network algorithms to surface acoustic wave (SAW) systems for in-sensor multi-biomarker diagnosis. To address optimization challenges, we introduce a novel training methodology that combines Finite Element Analysis (FEA) with gradient descent. We validate our method on Major Depressive Disorder (MDD) and prostate cancer, achieving accuracies of 74.07% and 86.0%, respectively, nearly reaching the accuracy levels of clinical diagnoses. By comparing the co-training method with the traditional gradient descent training method and direct training on the FEA model, the co-training method demonstrates its advantages in balancing training efficiency and accuracy. Furthermore, a comparison of power consumption is conducted between the traditional method and the in-sensor computing system, indicating 66% energy savings attributed to its high level of integration.**

***Index Terms*—Diffractive neuaral network; in-sensor computing; FEA method; SAW biosensor; multi-biomarker approach;**

I. Introduction

The convergence of artificial intelligence (AI) and biosensor technology has opened new frontiers in healthcare, promising improved diagnostic accuracy and reduced diagnostic time [1]. By integrating AI algorithms with analog processors directly into biosensors, in-sensor computing systems enable real-time analysis and decision-making at the point of data collection [2]. This eliminates the need for frequent analog-to-digital conversions and mitigates the von Neumann bottleneck caused by physically separated sensing, memory, and processing units [3], [4]. A prominent example is the diffractive optical neural network (DONN), an analog processor capable of in-sensor image recognition, which has gained considerable attention for its efficiency [5]–[7]. However, adapting DONNs for biosensing presents significant challenges, particularly in the context of microfluidic applications. One of the primary obstacles is the miniaturization of optical components to fit within the confined dimensions of microfluidic channels without compromising performance. High-quality optical components, such as lasers and detectors, are often difficult to miniaturize without a loss in efficiency or a concomitant increase in costs [8], [9]. This limits their practical applicability to chip-scale systems for biomolecular diagnostics.

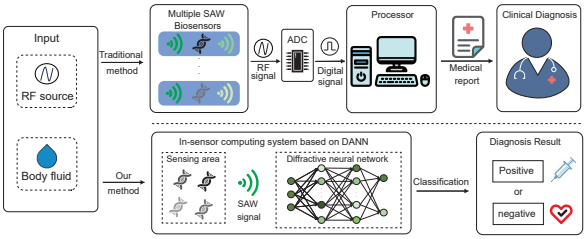


Fig. 1. Comparison of traditional diagnosis method with in-sensor computing system diagnosis

Surface acoustic wave (SAW) systems provide an alternative physical platform for realizing in-sensor computing by enabling a diffractive acoustic neural network (DANN). SAW-based biosensors exhibit high sensitivity in solution, as absorption of the target molecule induces distinct phase changes [10], [11]. Moreover, efficient coupling of SAWs into the microfluid compared to visible light provides DANNs more sensitive and a higher signal-to-noise ratio than DONNs for microfluidic biosensing applications [10], [12]. Like their optical counterparts, SAW-based diffractive neural networks benefit from reduced analog-to-digital conversions and colocated sensing, memory, and processing, resulting in ultra-low energy consumption (see Fig. 1).

Despite their potential, training DANNs poses unique hardware optimization challenges. Like DONNs, DANN training requires differentiable models, necessitating the extraction of propagation matrices that describe the wave behavior within the hardware. However, real-world SAW systems introduce complexities such as mechanical damping, substrate anisotropy, and nonlinear wave propagation [13], making analytical modeling infeasible. To address these issues, a physics-aware training methodology involving finite element analysis (FEA) is essential to accurately capture the dynamics of SAW-based systems [14], [15].

Our contributions are concluded as follows:

- **First Demonstration of SAW-based DANN**: We introduce SAW-based in-sensor DANN for biosensing ap-

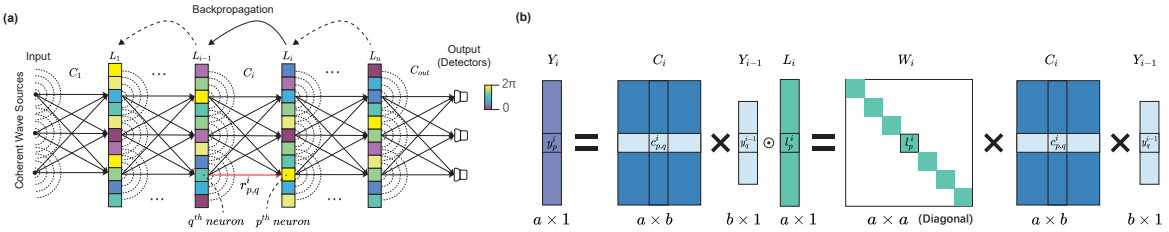


Fig. 2. (a) Diffractive neural network architecture; (b) Calculation carried out by adjacent layers.

plications with a focus on diagnosing major depressive disorder and prostate cancer.

- **Novel Training Methodology**: We propose a comprehensive training pipeline that minimizes errors between the algorithmic model and hardware, including physical parameter extraction, FEA-assisted model training, and model refinement.
- **Benchmarking Performance**: Our DANN achieves diagnostic accuracies of **74.07% for major depressive disorder** and **86.0% for prostate cancer**, approaching clinical standards and outperforming alternative approaches.
- **Energy Efficiency**: We demonstrate that in-sensor DANNs significantly reduce power consumption to 33% of that used by traditional hardware, highlighting their potential for low-power, high-integration biosensing solutions.

II. Preliminaries

A. Diffractive Neural Network

Diffractive neural network represents an algorithm deployed on hardware architecture which relies on diffraction process of wave to achieve neural network tasks [5]. Since diffraction is a fundamental property of waves, diffractive neural networks extend beyond optical applications and have been implemented in electromagnetic wave systems [16] and ultrasound systems [17] to accomplish various tasks. However, the deployment of diffractive neural network on SAW system has faced considerable challenges. The complexity of wave modes [12] and the anisotropic nature of substrates [13] have hindered the integration of diffractive neural network with SAW system.

A comparative analysis between the structure of diffractive neural networks and the conventional neural network reveals similarities and distinctions in their structural compositions and operational mechanisms. As shown in Fig. 2(a), both the diffractive neural network and the conventional neural network share similar training and testing methodologies, relying on forward propagation by matrix multiplication and backpropagation by gradient descent [5].

Differing from the weight matrices in the conventional neural network, the matrices in a diffractive neural network consist of two types shown in Fig. 2(b): weight matrices (W) which are inherently diagonal and constant propagation matrices (C) which are determined by physical structure of the system. The process of signal propagation through the ith diffraction layer can be represented by

$$Y^i = C_i \cdot Y^{i-1} \odot L_i = W_i \cdot C_i \cdot Y^{i-1} \tag{1}$$

where C^i is the propagation matrix of ith layer, L_i stores the parameter of ith layer in vector form and W_i is the diagonal matrix that consist of L_i as diagonal elements.

B. Phase Control

The physical counterpart of neural weights in diffractive neural networks is achieved through the phase velocity alteration of waves as they propagate through different materials. Considering a one-dimensional case, when a wave propagates through an artificial neuron, the wave function can be expressed as [18]:

$$\Psi(x,t) = A \cdot exp\left[j \cdot \left(\omega t - \overline{k} x_0 + \Phi\right)\right] \tag{2}$$

$$\overline{k} = \frac{2\pi f}{x_0}\left(\frac{x_1}{v_1} + \frac{x_2}{v_2}\right) \tag{3}$$

where x_1, v_1 represent the length and phase velocity of the special material, respectively, and x_2, v_2 represent the length and phase velocity of the original material. x_0 denotes the total length of the artificial neuron. Considering an increment (Δx) in the length of the special material within the neuron area, the phase shift induced by this increment is given by:

$$\Delta\Phi = 2\pi f \Delta x \cdot \left(\frac{1}{v_1} - \frac{1}{v_2}\right) \tag{4}$$

As a result, by modulating regions with different phase velocities, the phase properties of the wave are precisely controlled, thereby encoding the intended neural weights within the diffractive neural network.

C. Propagation Matrix

Beyond the parameters of artificial neurons controlled through phase shifts, the propagation matrices, as illustrated in Fig. 2(a), are also critical parameters for diffractive neural networks. Unlike weight matrices, propagation matrices are untrainable, depending solely on the spatial positions of neurons and the distances between diffractive layers. These matrices characterize the wave propagation process between layers. Typically, each element within the propagation matrix can be calculated using:

$$c^i_{p,q} = A(\vec{r}_0) \cdot exp\left[j \cdot \vec{r}^{\,i}_{p,q} \cdot \vec{k}(\vec{r}_0)\right] \tag{5}$$

where $c^i_{p,q}$ denotes the element in the p-th row q-th column of i-th propagation matrix. Here, $\vec{r}^{\,i}_{p,q}$ represents displacement vector from the q-th neuron in the previous layer to the p-th neuron in the subsequent layer. $\vec{r}_0$ is the unit vector that repersents the propagation direction of wave. The term $A(\vec{r}_0)$ and $\vec{k}(\vec{r}_0)$ denotes the wave amplitude decay ratio and wave vector on the propagation direction.

For an optical diffraction system in air, calculating propagation matrices is relatively straightforward by assuming

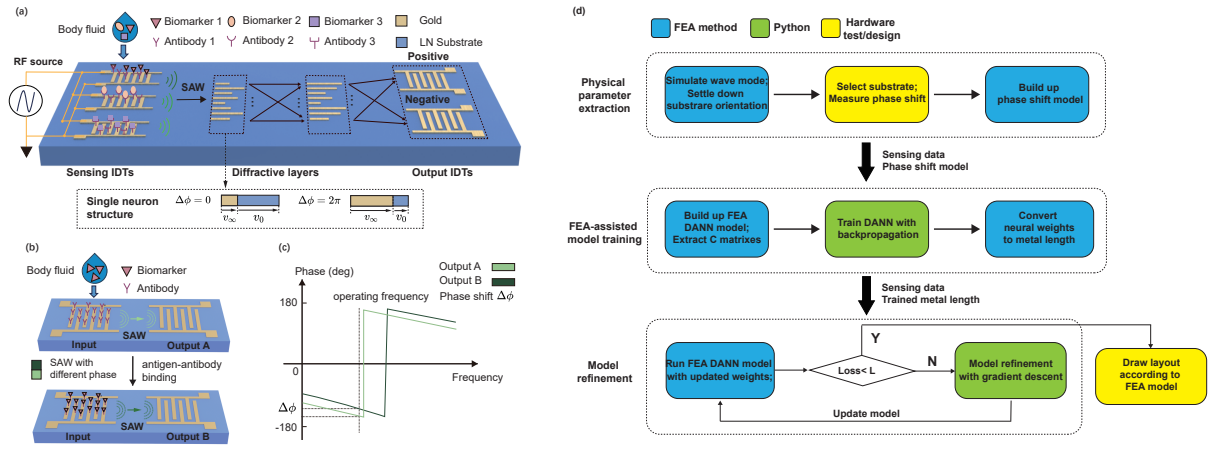


Fig. 3. Overview of in-sensor computing system based on DANN. (a) Hardware architecture overview; (b) SAW biosensor mechanism; (c) Sensing signal of SAW biosensor; (d) Design pipeline overview.

$A(\vec{r}_0) = 1$ and simplifying vector $\vec{k}(\vec{r}_0)$ to scalar k_{air}, due to isotropy and low-loss character of the media [7]. Under the approximation, Eq. 5 can be simplified to:

$$c^i_{p,q} = 1 \cdot exp\left[j \cdot r^i_{p,q} \cdot k_{air}\right] \tag{6}$$

However, in a SAW diffraction system, the anisotropy characteristic and mechanical loss of the substrate prevent such simplifications; applying these approximations would induce substantial system errors, leading to algorithm failure in the DANN. Additionally, obtaining analytical expressions for $A(\vec{r}_0)$ and $\vec{k}(\vec{r}_0)$ is challenging due to their pronounced nonlinearity in response to the complex behavior of the substrate. To address these challenges, we propose using Finite Element Analysis (FEA) [19] to extract the propagation matrix for the DANN system, thereby ensuring accurate modeling of wave propagation under real conditions.

III. DANN for In-sensor Computing System

In this section, we propose an in-sensor computing system architecture based on DANN, illustrated in Fig. 3(a). We firstly introduce the hardware part of the system which includes the physical structure of neural weights and the sensing part of the system. The following discussions related to the design pipeline are divided into three parts corresponding to three stages: parameter extraction, FEA-assisted model training, and model refinement, shown in Fig. 3(d).

A. In-sensor DANN Hardware

Phase control. In DANN system, the phase shift of neural weights is controlled by the metal length of neurons. SAW velocity can be modulated by adjusting the area of the metal-covered substrate shown in Fig. 3(a). As SAW propagates through the metal-covered region, its velocity decreases due to the effect of electromechanical coupling. The electromechanical coupling coefficient (K^2) is a critical parameter that describes the ratio of wave velocity change which can be expressed as: [18]:

$$K^2 = \frac{v_0^2 - v_\infty^2}{v_0^2} \cong 2\frac{v_0 - v_\infty}{v_0} \equiv 2\frac{\Delta v}{v_0}, \quad (v_0 > v_\infty) \tag{7}$$

where v_0 refers to the phase velocity of SAW in piezoelectric substrate with free surface condition and v_∞ refers to the phase velocity of SAW in piezoelectric substrate covered with material of metal. Refering to Eq.4, the relation ship of phase shift ($\Delta\Phi$) with K^2 can be expressed in:

$$\Delta\Phi = \frac{2\pi\Delta x}{\lambda_0} \cdot \frac{\Delta v}{v_\infty} = \frac{2\pi\Delta x}{\lambda_0} \cdot \frac{K^2}{2 + K^2} \tag{8}$$

where λ_0 is the wavelength of SAW with free surface condition.

Biosensing area. SAW biosensing area leverages the interaction between surface acoustic waves and biological analytes to detect and quantify biological substances with high sensitivity and specificity. As shown in Fig. 3(b), the sensing ability of SAW device is facilitated by the development of sensitive surface coatings on the SAW device that selectively bind to target analytes. When target molecules bind to these coatings, they alter the mass load on the surface, modulating the phase of the SAW signal. The phase change of SAW signal is converted to electric signal as demonstrated in Fig. 3(c). By integrating DANN and biosensing area, the phase shift of the sensor can be processed directly through diffractive layers. This integration enables the system to analyze multiple biomarkers simultaneously, achieving a comprehensive diagnosis of diseases.

B. Physical Parameter Extraction

The initial stage in training an in-sensor computing based on DANN involves physical parameter extraction, which settles down the fundamental physical parameters of FEA model by subsystem testing of SAW hardware. The first critical parameter is the SAW mode, determined by the material properties and substrate orientation. As noted in previous works [20], the shear horizontal (SH) mode is preferable for SAW biosensors and can be stimulated on a 41° Y-cut LN substrate. To ensure the accuracy of the FEA model's physical behavior, we begin by simulating the wave mode for this specific substrate orientation. Fig. 4(a) presents a 3D simulation of an IDT section generating SAW in the 41° Y-cut direction. From the top view of model, the SAW particle

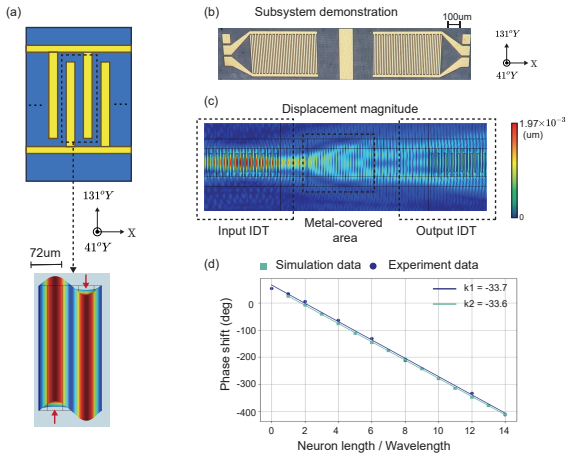


Fig. 4. (a) Top view of 3-dimensional FEA simulation of wave mode; (b) Photo of subsystem; (c) FEA model of subsystem; (d) Phase shift comparision of simulation and hardware experiment.

motion follows a zigzag pattern, confirming the presence of SH wave mode.

The next parameter to address is K^2, which links the phase shift to metal length. By settling down K^2, we can ensure that the DANN FEA simulation accurately reflects the hardware performance. To obtain K^2, the phase shifts of subsystems are tested, demonstrated in Fig. 4(b), where a pair of IDTs with a fixed distance and varying metal lengths on the substrate are used. Refering to Eq. 8, the experiment data, presented in Fig. 4(d), allows for the calculation of K^2 by following equation:

$$k = \frac{\Delta\Phi}{\frac{\Delta x}{\lambda_0}} = 2\pi \cdot \frac{K^2}{2 + K^2} \tag{9}$$

where k is the slope calculate by linear regression of experiment data.

To verify the accuracy of the FEA model in representing the physical system, we input the calculated K^2 into the FEA model and construct models based on the subsystem geometry to simulate the phase shift, as shown in Fig. 4(c). By comparing the experimental results with the simulation results in Fig. 4(d), we confirm that the performance of the FEA model, incorporating the calculated K^2 aligns closely with the hardware structure. Upon validation, the substrate orientation, along with the parameters for K^2, is passed to the next stage for FEA-assisted model training of the DANN system.

C. FEA-assisted Model Training

The primary objective of FEA-assisted model training is to develop a differentiable model of the DANN system that accurately represents the diffraction process with minimal error. As discussed in Section 2.3, using approximations similar to those in DONN systems introduces significant errors into the SAW-based system, while calculating the propagation matrix analytically poses substantial challenges. To address this, the propagation matrix is extracted numerically using the FEA model of the DANN system. By integrating these numerically derived propagation matrices into the diffractive neural network, a differentiable physical model of the DANN

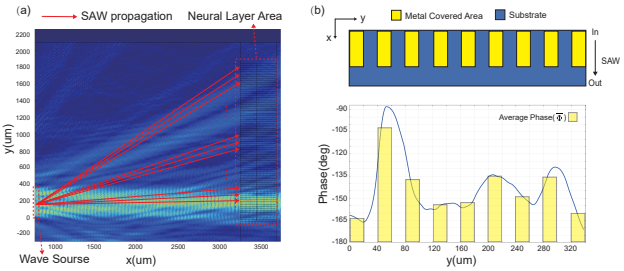


Fig. 5. (a) FEA simulation of propagation matrix; (b) Average phase shift of surface displacement at neruon positions;

system is established, which achieves a low representation error and supports training via backpropagation.

In the FEA-assisted model training stage, we first construct the FEA model based on the system's geometry. Material parameters and crystal orientations are assigned to the substrate, while the neuron areas, covered with metal, are initialized and assigned material properties, including the K^2 parameter obtained from the previous stage, to achieve the desired phase velocities of the SAW. To extract the propagation matrix, we apply Huygens' principle and exploit the symmetry of the system. By conducting FEA simulations with a single wave source excitation in the frequency domain,as shown in Fig. 5(a), the displacement at each point in the system can be represented as a complex number. The elements of the propagation matrix are then computed using the following equations:

$$c^i_{p,q} = \frac{\overline{A^i_p}}{A^{i-1}_q} exp\left[j \cdot \left(\overline{\Phi^i_p} - \overline{\Phi^{i-1}_q}\right)\right] \tag{10}$$

where $\overline{A^i_p}$ and $\overline{\Phi^i_p}$ represents the average amplitude and phase of the output points of q-th neuron in the i-th layer. $\overline{A^{i-1}_q}$ and $\overline{\Phi^{i-1}_q}$ represent the average amplitude and phase of the input points of p-th neuron in the $(i-1)$-th layer. The process of taking the average phase shift is demonstrated in Fig. 5(b).

After determining the propagation matrices, we input the matrices into DANN system and train it by backpropagation. Specifically, each weight $w^i[m]$ $(m \in \{1, 2, \cdots, a\})$in diagno matrix $\mathbf{W}^i$ is updated by

$$w^i_t[m] \leftarrow w^i_t[m] - \alpha \cdot \frac{\partial \mathcal{L}}{\partial y^i} \cdot \frac{\partial y^i}{\partial w^i_t[m]}, \tag{11}$$

$$w^i[m] \leftarrow \frac{w^i_t[m]}{\left\| w^i_t[m] \right\| \|} \tag{12}$$

where α is the learning rate. $\mathcal{L}$ denotes the loss function which is calculated by negative log likelihood loss. y^i is element in the output matrix Y^i. $w^i_t[m]$ is the intermediate weight of the m-th element in layer i, updated with a full gradient of the complex number. As the trained weights of the DANN correspond to the phase shifts of the hardware structure, the updated weight is normalized by Eq. (12). In the final step, well-trained neural weights are converted into metal lengths using Eq. (8).

D. Model Refinement

The final stage involves testing and refining the FEA model, which directly represents the geometry of the system layout.

Algorithm 1 Stochastic area tunning

Input: FEA model $F(\cdot)$, loss function calculated by Negative Log-Likelihood Loss $\mathcal{L}(\cdot)$, training dataset D, total iterations T, initial position vector P^0, initial tuning step size δp^0, cardinality of finetuned position N, and decay factor α;

for $t \leftarrow 1, ..., T$ **do**
  $O = F(d^t, P^t)$, $d^t \sim D$ ▷ Run FEA model with dataset and position
  $\mathcal{L}(O, d^t)$ ▷ Calculate loss of FEA model
  Randomly select a set of position $\{p_i^t\}_{i=1}^N \subseteq P^t$
  for each position $p_i^t \in \{p_i^t\}_{i=1}^N$ **do**
    $O' = F(d^t, p_i^t + \delta p_i^t)$ ▷ Run FEA model with dataset and tuned position
    if $\mathcal{L}(O, d^t) < \mathcal{L}(O', d^t)$ **then**
      $p_i^{t+1} \leftarrow p_i^t - \delta p_i^t$
    else
      $p_i^{t+1} \leftarrow p_i^t + \delta p_i^t$
    end if
  end for
  $\delta p^{t+1} = \alpha \cdot \delta p^t$
end for

Output: Converged position vector P^T

To validate the system before hardware fabrication, the training dataset is used as input to the FEA model, with the trained metal lengths applied. The output is measured as an electrical signal at the output electrode and the classification accuracy of the FEA model is calculated.

The performance of the FEA model often degrades compared to the differentiable model developed in Stage 2 due to the averaging process involved in extracting the propagation matrix, as discussed in Eq. (10). To mitigate the error introduced by this approximation, a stochastic tuning step is performed on the positions of the diffractive layer areas and output electrodes in the FEA model, denoted as y_i $(i = 1, \ldots, n)$ in Fig.5(a). The tuning algorithm is detailed in Alg.1, introduces stochastic perturbations to these areas to optimize their placement, compensating for errors introduced during propagation matrix averaging and significantly improving system performance.

Since running the FEA model is computationally intensive compared to Python-based training, it is crucial to identify the positions requiring the most adjustment to optimize the tuning process. Errors are most pronounced in the output propagation matrix, where averaging displacement over wider output electrodes causes larger inaccuracies. Therefore, prioritizing the adjustment of output electrode positions effectively reduces computational overhead while enhancing the overall accuracy of the system.

IV. EVALUATION

A. Experiment Setup

Hardware and Training Setup. To design a DANN insensor computing system, we select the 41^oY cut LN as our substrate with working frequency at 120MHz. The material parameters are adopted from the database in COMSOL which is a commercial FEA modeling software. We design two network structures: 3-36-36-2 and 2-36-2, in order to complete Major Depressive Disorder (MDD) and prostate cancer diagnosis, respectively. In each task, to evaluate the training performance, we compared the inference accuracy of the system with 1) clinical diagnosis accuracy, 2) trained by Python only, 3) FEA-assisted model training, and 4)FEA-assisted model training with model refinement. All software experiments are carried out with an Intel Xeon Platinum 8375C CPU and an NVIDIA GTX 2070 GPU.

Dataset. As introduced in Section 3, the response of SAW biosensing signals is related to phase shift. For simplicity, we assume that the response range of the SAW sensors is consistent across all cases. In the context of Major Depressive Disorder (MDD) diagnosis, patients are considered positive when all three biomarkers—cortisol, interleukin-6 (IL-6), and C-reactive protein (CRP)—are positive [21]. Accordingly, positive cases in the MDD dataset are generated using a normal distribution with high sensor response ($\mu = [10^o, 10^o, 10^o], \sigma = 3.5^o$);, while negative cases are generated with combinations of low sensor responses, as shown in Fig.6(a). In the case of prostate cancer diagnosis, patients are considered positive when the prostate-specific antigen (PSA) is detected as positive, while its membrane isoform (PSMA), is detected as negative. [22], [23]. Consequently, positive cases are generated by a normal distribution with a high response in PSA but a low response in PSMA ($\mu = [10^o, 2^o], \sigma = 2^o$), shown in Fig.6(b).

B. Training Method Comparison

To evaluate the performance of FEA-assisted model training, we compare the classification accuracy of models trained by 1) traditional method, 2) FEA-assisted model training without model refinement process, and 3) complete FEA-assisted model training. The traditional method involves training a diffractive neural network with propagation matrices estimated by Eq. 6. All models are tested through FEA simulations.

Fig.6(c)-(f) illustrates the displacement field differences between the positive and negative cases. In a positive case, the displacement amplitude increases at the top electrode, while in a negative case, the displacement amplitude increases at the bottom electrode. By subtracting the negative case with positive case, the displacement field around the top electrode should remain high, and the displacement field around the bottom electrode should become negative, if the model performs the classification task correctly, as shown in Fig.6(e), (f). In the MDD application, the displacement field of the traditionally trained model fails to maintain a high value around the top output IDT, leading to incorrect classification results. In the prostate cancer application, the displacement field of the traditionally trained model remains high at both top and bottom IDTs, indicating poor accuracy in the classification task.

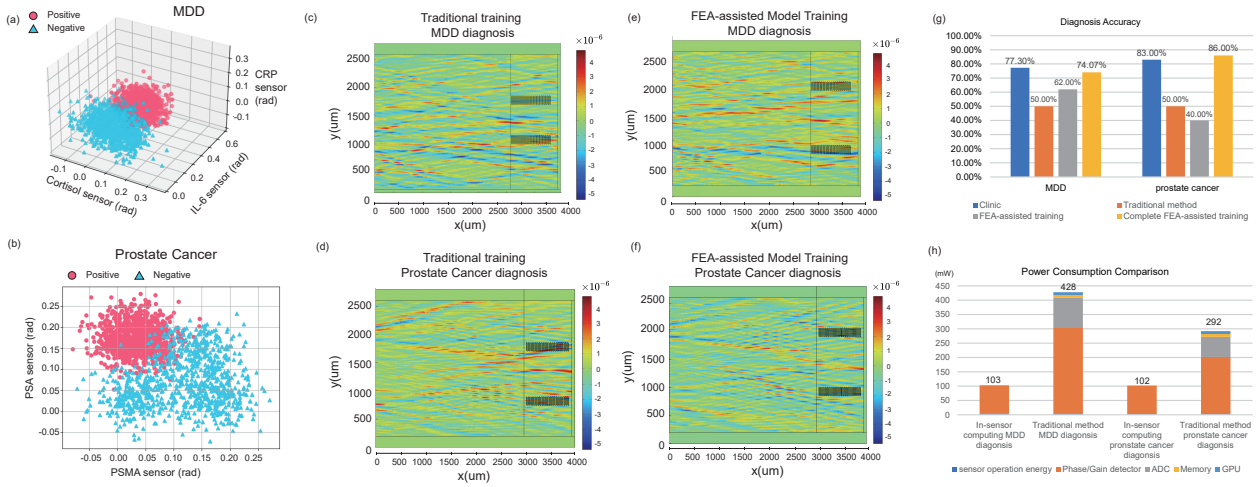


Fig. 6. (a) Dataset of MDD; (b) Dataset of prostate cancer; (c) ,(d) are the displacement difference of traditional trained models target at MDD and prostate cancer diagnosis respectively; (e), (f) are the displacement difference of FEA-assisted trained models target at MDD and prostate cancer diagnosis respectively; (g) The diagnosis accuracy of MDD and prostate cancer;(h) Power consumption comparison.

TABLE I
TIME CONSUMPTION COMPARISON.

Model type	Time consumption per epoch
Python model	3×10^{-3}s
FEA model	330s

Fig.6(g) shows the inference accuracy of models trained by traditional method, FEA-assisted training without model refinement and complete FEA-assisted training. Models trained by traditional method achieves 50% inference accuracy, equivalent to random selection, indicating no classification ability. In prostate cancer diagnosis, the model trained by FEA-assisted training without model refinement performs worse than 50%, due to errors from the approximation during propagation matrix extraction, especially in the output layer with wide IDTs. After model refinement, accuracy improves to 74.07% and 86.00% in MDD and prostate cancer diagnosis, which almost reaches clinical diagnostic levels.

In addition to training the DANN system using backpropagation, it is also possible to train it directly with the FEA model. We explored the application of on-chip training methods, such as brute-force phase tuning (BFT) [24], [25], to directly adjust the metal lengths in the diffractive layer. However, the on-chip training method requires running multiple epochs on the FEA model, with each epoch taking approximately 5 minutes, which makes it approximately 10^5 times slower than Python-based training, as shown in tableI. Therefore, after considering both computational costs and performance, the FEA-assisted model training method emerges as the most effective approach to train DANN, combining the precision of the FEA model with the efficiency of backpropagation.

C. In-sensor Computing System Evaluation

To demonstrate the energy efficiency of the in-sensor computing system achieved by DANN, we compared its power consumption with the traditional method, as shown in Fig.1. The power consumption of DANN arises from the radio frequency (RF) input of the IDTs, which is around 1 mW. In contrast, the traditional method requires additional components, such as a phase detector module to detect sensor responses, an analog-to-digital converter (ADC) to convert the RF signal from the SAW sensor into the digital signal, and a digital processor to carry out neural network algorithm. Integrating these modules with the SAW biosensor is challenging due to its fabrication on a piezoelectric substrate. An alternative is to use a separate test bench to house the necessary modules. Commercial ADC and phase detector chips typically consume between 10 mW and 100 mW, which is 10 times higher than the power consumption of the SAW sensor alone. The detailed comparison of power consumption is illustrated in Fig6(g).When accounting for the comparator at the output of the DANN system, the total power consumption is reduced by at least 33% compared to the traditional method. Thus, DANN offers a significant advantage in terms of high integration and low power consumption.

V. CONCLUSION

In this work, we propose the DANN system trained by FEA-assisted model training methods. By comparing different training methods on MDD and prostate cancer diagnosis task, we demonstrated the effectiveness and efficiency of FEA-assisted model training methods. Compared to conventional medical testing methods, the DANN-based in-sensor computing system offers distinct advantages in terms of simplicity and intelligence.

ACKNOWLEDGMENT

This research is supported by Beijing Natural Science Foundation (Grant No. Z210006), the National Key R&D Program of China (Grant No. 2023YFB2806300), Shenzhen Science and Technology Innovation Commission (Grant No. SGDX20220530111405040), Hong Kong Research Grant Council (Grant Nos. 27209621, 17205922, 17212923, 17208623). This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

REFERENCES

[1] Mingrui Chen, Daxiang Cui, Hossam Haick, and Ning Tang. Artificial intelligence-based medical sensors for healthcare system. *Advanced Sensor Research*, 3(3):2300009, 2024.

[2] Byungjoon Bae, Yongmin Baek, Jeongyong Yang, Heesung Lee, Charana SS Sonnadara, Sangeun Jung, Minseong Park, Doeon Lee, Sihwan Kim, Gaurav Giri, et al. Near-sensor computing-assisted simultaneous viral antigen and antibody detection via integrated label-free biosensors with microfluidics. *InfoMat*, 5(10):e12471, 2023.

[3] Hossam Haick and Ning Tang. Artificial intelligence in medical sensors for clinical decisions. *ACS nano*, 15(3):3557–3567, 2021.

[4] Chan Wang, Tianyiyi He, Hong Zhou, Zixuan Zhang, and Chengkuo Lee. Artificial intelligence enhanced sensors-enabling technologies to next-generation healthcare and biomedical platform. *Bioelectronic Medicine*, 9(1):17, 2023.

[5] Xing Lin, Yair Rivenson, Nezih T Yardimci, Muhammed Veli, Yi Luo, Mona Jarrahi, and Aydogan Ozcan. All-optical machine learning using diffractive deep neural networks. *Science*, 361(6406):1004–1008, 2018.

[6] Tianyu Wang, Mandar M Sohoni, Logan G Wright, Martin M Stein, Shi-Yuan Ma, Tatsuhiro Onodera, Maxwell G Anderson, and Peter L McMahon. Image sensing with multilayer nonlinear optical neural networks. *Nature Photonics*, 17(5):408–415, 2023.

[7] Tingzhao Fu, Yubin Zang, Yuyao Huang, Zhenmin Du, Honghao Huang, Chengyang Hu, Minghua Chen, Sigang Yang, and Hongwei Chen. Photonic machine learning with on-chip diffractive optics. *Nature Communications*, 14(1):70, 2023.

[8] Cun-Zheng Ning. Semiconductor nanolasers and the size-energy-efficiency challenge: a review. *Advanced Photonics*, 1(1):014002–014002, 2019.

[9] S Nicoletti, P Barritault, S Boutami, M Brun, A Gliere, P Labeye, J Rouxel, J Czarny, H Lhermet, M Carras, et al. Challenges in the realization of a fully integrated optical lab-on-chip. In *SENSORS, 2014 IEEE*, pages 649–652. IEEE, 2014.

[10] Kathrin Baumgartner and Christoph Westerhausen. Recent advances of surface acoustic wave-based sensors for noninvasive cell analysis. *Current Opinion in Biotechnology*, 79:102879, 2023.

[11] Xiaoyun Ding, Peng Li, Sz-Chin Steven Lin, Zackary S Stratton, Nitesh Nama, Feng Guo, Daniel Slotcavage, Xiaole Mao, Jinjie Shi, Francesco Costanzo, et al. Surface acoustic wave microfluidics. *Lab on a Chip*, 13(18):3626–3649, 2013.

[12] Kerstin Länge, Bastian E Rapp, and Michael Rapp. Surface acoustic wave biosensors: a review. *Analytical and bioanalytical chemistry*, 391:1509–1519, 2008.

[13] Shitian Huang, Yao Shuai, Lu Lv, Zijie Wei, Wei Fan, Yuedong Wang, Dailei Zhu, Xinqiang Pan, Wenbo Luo, Chuangui Wu, et al. Linbo3 surface acoustic wave resonators with large effective electromechanical coupling. *Electronics*, 12(13):2964, 2023.

[14] Tingzhao Fu, Yubin Zang, Honghao Huang, Zhenmin Du, Chengyang Hu, Minghua Chen, Sigang Yang, and Hongwei Chen. On-chip photonic diffractive optical neural network based on a spatial domain electromagnetic propagation model. *Optics Express*, 29(20):31924–31940, 2021.

[15] Jennifer Brookes, Rory Bufacchi, Jun Kondoh, Dorothy M Duffy, and Rachel A McKendry. Determining biosensing modes in sh-saw device using 3d finite element analysis. *Sensors and Actuators B: Chemical*, 234:412–419, 2016.

[16] Che Liu, Qian Ma, Zhang Jie Luo, Qiao Ru Hong, Qiang Xiao, Hao Chi Zhang, Long Miao, Wen Ming Yu, Qiang Cheng, Lianlin Li, et al. A programmable diffractive deep neural network based on a digital-coding metasurface array. *Nature Electronics*, 5(2):113–122, 2022.

[17] Jingkai Weng, Yujiang Ding, Chengbo Hu, Xue-Feng Zhu, Bin Liang, Jing Yang, and Jianchun Cheng. Meta-neural-network for real-time and passive deep-learning-based object recognition. *Nature communications*, 11(1):6309, 2020.

[18] Ken-ya Hashimoto and Ken-Ya Hashimoto. *Surface acoustic wave devices in telecommunications*, volume 116. Springer, 2000.

[19] Olek C Zienkiewicz and Robert L Taylor. *The finite element method set*. Elsevier, 2005.

[20] Debdyuti Mandal and Sourav Banerjee. Surface acoustic wave (saw) sensors: Physics, materials, and applications. *Sensors*, 22(3):820, 2022.

[21] Yinghui Li, Yingying Yue, Suzhen Chen, Wenhao Jiang, Zhi Xu, Gang Chen, Zixin Zhu, Liangliang Tan, and Yonggui Yuan. Combined serum il-6, c-reactive protein, and cortisol may distinguish patients with anhedonia in major depressive disorder. *Frontiers in Molecular Neuroscience*, 15:935031, 2022.

[22] David Heim, Gian Luca Barbruni, and Sandro Carrara. A novel approach in edge computing: In-memory sensing of cancer markers. In *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 306–310. IEEE, 2022.

[23] Shikha Sharma, Julia Zapatero-Rodriguez, and Richard O'Kennedy. Prostate cancer diagnostics: Clinical challenges and the ongoing need for disruptive and effective diagnostic tools. *Biotechnology Advances*, 35(2):135–149, 2017.

[24] Jiaqi Gu, Zheng Zhao, Chenghao Feng, Wuxi Li, Ray T Chen, and David Z Pan. Flops: Efficient on-chip learning for optical neural networks through stochastic zeroth-order optimization. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[25] Yichen Shen, Nicholas C Harris, Scott Skirlo, Mihika Prabhu, Tom Baehr-Jones, Michael Hochberg, Xin Sun, Shijie Zhao, Hugo Larochelle, Dirk Englund, et al. Deep learning with coherent nanophotonic circuits. *Nature photonics*, 11(7):441–446, 2017.

Learning-Aided Safe Controller Synthesis with Formal Guarantees via Vector Barrier Certificates

1st Xia Zeng
Southwest University
xzeng0712@swu.edu.cn

2nd Mengxin Ren
East China Normal University
mxren@stu.ecnu.edu.cn

3st Zhiming Liu
Southwest University
zhimingliu88@swu.edu.cn

4nd Zhengfeng Yang*
East China Normal University
zfyang@sei.ecnu.edu.cn

***Abstract*—The design of controllers for safety-critical systems is an important research issue. Especially, the generation of controllers with formal safety guarantees is a challenging problem. Recently, for safety objectives of various system control tasks, machine learning technologies have been used to achieve ideal training and simulation performance, but formal guarantees are still lacking. This paper takes advantages of learning technology to assist safe controller synthesis with formal guarantees. On the one hand, the generation of verifiable safe controllers is aided by reinforcement learning; on the other hand, a set of barrier certificates (BC), i.e. a vector BC, is synthesized with the aid of deep learning to certify the safety of synthesized controllers. Vector BCs are more expressive than the conventional single BCs for safety verification. Compared with the existing work on vector BC generation, our method has two advantages: first, our method verifies a learned candidate vector BC, rather than directly generating a verified one, and thus has low computational complexity; second, the existing method has made relaxations to the non-convex vector BC constraints, which reduced the feasible region of solutions, while our method can deal with the original constraints. Furthermore, experiments fully demonstrate the effectiveness of our method on a series of benchmarks.**

***Index Terms*—controller synthesis, safety verification, vector barrier certificate, reinforcement learning, sum-of-squares**

I. Introduction

The design and synthesis of controllers are fundamental challenges in control theory. With the rapid advancement of deep learning, there has been a surge in research exploring the application of neural networks (NNs) for controlling nonlinear systems [1], [2]. NNs possess a robust capacity to represent complex nonlinear mappings and offer efficient computational performance, making them well-suited for sophisticated control tasks [3]. However, learned control policies often lack formal guarantees, limiting their applicability in practical scenarios, especially within safety-critical control systems. To ensure system safety under a designed controller over an indefinite time horizon, a crucial verification method is the generation of barrier certificates, which are employed to demonstrate that the trajectories of an autonomous dynamical system remain within a specified safe region. Since barrier certificates provide sufficient conditions for safety assurance, researchers are continually exploring various forms of these functions to expand the range of verifiable problems.

The foundational work by Prajna et al. introduced the use of barrier certificates for the safety verification of hybrid systems [4]. Building upon this, subsequent research has focused on exploring various forms of barrier certificates to enrich and enhance their expressiveness. Kong et al. proposed barrier certificates based on exponential conditions for semi-algebraic systems [5]. Dai et al. discussed general relaxations of barrier certificate conditions that maintain convexity, broadening the scope of systems amenable to safety verification [6]. Sloth et al. introduced barrier certificates tailored for hybrid systems comprising multiple interconnected subsystems, facilitating compositional safety analysis [7]. Additionally, Zeng et al. explored Darboux-type barrier certificates, which define algebraic curves that system trajectories cannot cross once they intersect [8].

In fact, previous mainstream research on barrier certificates has predominantly utilized scalar functions. Recently, Sogokon et al. introduced a composite barrier function condition termed the vector barrier certificate (vector BC) [9]. When the combination number is set to one, it simplifies to a scalar barrier certificate, indicating that vector BCs offer broader expressiveness compared to conventional scalar barrier certificates. However, the theoretical advantages of vector BCs are accompanied by practical challenges. The existence conditions for vector BCs involve non-convex constraints, complicating the parameter-solving process. To facilitate computation, existing approaches have modified these non-convex constraints by strengthening the disjunctive conditions of multiple barrier certificates to one of the disjuncts. This modification, while simplifying the solving process, may result in the loss of feasible solutions to the original vector BC conditions. Besides, the so-called essentially non-negative matrix in the vector BC conditions needs to be handcrafted, which complicates its application.

In this paper, we focus on designing controllers with formal safety guarantees. We propose a novel method that leverages learning techniques to synthesize safe controllers for nonlinear systems using vector barrier certificates. Initially, we employ safe reinforcement learning combined with Probably Approximately Correct (PAC) approximation to obtain an ideal and verifiable safe controller candidate. Subsequently, we develop a new safety verification approach that utilizes

* Corresponding author

This work was supported in part by the National Key Research and Development Project, China under Grant 2022YFA1005101, the National Natural Science Foundation of China under Grants (No.62472362, 62272397, 12171159, 62032019), "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100, and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.

learning techniques to assist in generating vector BCs. Unlike existing methods, our approach does not require modifying or strengthening the disjunctive constraints in the original vector BC conditions, and can learn the essentially non-negative matrix automatically, thereby avoiding solution loss and enhancing expressiveness and flexibility. In summary, this paper makes the following contributions: We leverage advanced learning techniques to propose a novel method for synthesizing safe controllers with formal guarantees; We have developed a novel approach that employs learning techniques to assist in the generation of vector BCs, through learning we avoid the drawbacks of conventional vector BC generation methods that strengthen constraints, causing the loss of potential feasible solutions for vector BCs; We propose an incremental SOS-encoding method to transform the formal verification of vector BC conditions into a series of semi-definite programs, enabling highly efficient posterior verification after learning; We conduct extensive comparative experiments on state-of-the-art benchmarks to demonstrate the effectiveness and robustness of our proposed method.

II. Preliminaries

Notations. Let $\mathbb{R}$ be the field of real numbers, $\mathbb{R}[\mathbf{x}]$ be the ring of polynomials with coefficients in $\mathbb{R}$ over n-dimensional variables $\mathbf{x} = (x_1, x_1, \ldots, x_n)^\top$, $\mathbb{R}[\mathbf{x}]^n$ be the space of n-dimensional vector polynomials, $\Sigma[\mathbf{x}] \subset \mathbb{R}[\mathbf{x}]$ be the space of Sum-of-Squares (SOS) polynomials, and $A_{i:}$ be the elements in the i-th row of a matrix A.

A. Safe Control Problem

A controlled continuous dynamical system is modeled by first-order ordinary differential equations

$$\begin{cases} \dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \\ \mathbf{u} = \pi(\mathbf{x}) \end{cases}, \tag{1}$$

where $\mathbf{x} \in D \subseteq \mathbb{R}^n$ is the system state, $\mathbf{u} \in U \subseteq \mathbb{R}^m$ is the control input, and $\mathbf{f} \in \mathbb{R}[\mathbf{x}]^n$ is the vector field defined on the state space $D \subseteq \mathbb{R}^n$. Assume $\mathbf{f}$ satisfies the local Lipschitz condition, which ensures (1) has a unique solution $\mathbf{x}(t, \mathbf{x}_0)$ in D for every initial state $\mathbf{x}_0 \in D$ at $t = 0$.

In many contexts, a dynamical system is equipped with a domain $\Psi \subset D$ and an initial set $\Theta \subset \Psi$, represented as a triple $\mathcal{C} \doteq (\mathbf{f}, \Psi, \Theta)$. Given a prespecified unsafe region $X_u \subset D$, we say that the system $\mathcal{C}$ is *safe* if all system trajectories starting from Θ can not evolve into any state specified by X_u, which has been widely investigated in safety critical applications.

Definition 1 (Safety): For *a* controlled constrained continuous dynamical system (CCDS) $\mathcal{C} = (\mathbf{f}, \Psi, \Theta)$ and a given unsafe region X_u, the system is safe if for all $\mathbf{x}_0 \in \Theta$, there does not exist $t_1 > 0$ s.t. $\mathbf{x}(t_1, \mathbf{x}_0) \in X_u \wedge \forall t \in [0, t_1).\mathbf{x}(t, \mathbf{x}_0) \in \Psi$.

Definition 2 (Safe Controller Synthesis): Given a controlled CCDS $\mathcal{C} = (\mathbf{f}, \Psi, \Theta)$ with $\mathbf{f}$ defined by (1) with an unsafe set X_u, design a locally Lipschitz continuous feedback control law π such that the closed-loop system $\mathcal{C}$ with $\mathbf{f} = \mathbf{f}(\mathbf{x}, \pi(\mathbf{x}))$ is safe as per Definition 1.

In this paper, we consider the above safe controller synthesis problem for nonlinear systems with formal guarantee.

B. Vector Barrier Certificates

The concept of *barrier certificates* (BCs) plays an important role in safety verification of continuous systems. The essential idea is to use the zero level set of a function $B(\mathbf{x})$ as a barrier to separate all the reachable states from the unsafe region. Since the existence of BCs serves as a sufficient condition for system safety, efforts have been made to explore certificate functions that meet less rigid criteria. The concept of vector BCs is proposed in [9]. Compared to traditional scalar BCs, it represents a more expressive barrier certificate family.

Definition 3 (Vector Barrier Certificate): Let the system (1) and sets Ψ, Θ, X_u be given. An m-dimensional vector of continuously differentiable function $\mathrm{B}(\mathbf{x}) = (B_1(\mathbf{x}), B_2(\mathbf{x}), \cdots, B_m(\mathbf{x}))^\top : \Psi \to \mathbb{R}^m$ is called a vector barrier certificate if it satisfies

VBC1. $\forall \mathbf{x} \in \mathbb{R}^n.(\mathbf{x} \in \Theta \Rightarrow \wedge_{i=1}^m B_i(\mathbf{x}) \leq 0)$,
VBC2. $\forall \mathbf{x} \in \mathbb{R}^n.(\mathbf{x} \in X_u \Rightarrow \vee_{i=1}^m B_i(\mathbf{x}) > 0)$,
VBC3. $\forall \mathbf{x} \in \mathbb{R}^n.(\mathbf{x} \in \Psi \Rightarrow \mathcal{L}_\mathbf{f}\mathrm{B}(\mathbf{x}) \leq A\mathrm{B}(\mathbf{x}))$,

where $\mathcal{L}_\mathbf{f}\mathrm{B}(\mathbf{x})$ denotes the element-wise Lie-derivative of $\mathrm{B}(\mathbf{x})$ along the vector field $\mathbf{f}(\mathbf{x})$, i.e., $\mathcal{L}_\mathbf{f}\mathrm{B}(\mathbf{x}) = (\mathcal{L}_\mathbf{f}B_1(\mathbf{x}), \mathcal{L}_\mathbf{f}B_2(\mathbf{x}), \cdots, \mathcal{L}_\mathbf{f}B_m(\mathbf{x}))^\top$, and $A \in \mathbb{R}^{m \times m}$ is some *essentially non-negative* matrix.

Remark 1: When $m = 1$, the vector BC in Definition 3 reduces to a scalar function, which corresponds to the traditional single BC condition.

Definition 3 is developed by applying the comparison principle in the theory of ODEs to safety verification. An important prerequisite for this theorem is that the $m \times m$ real matrix A is essentially non-negative, i.e., all the off-diagonal entries of A are non-negative [10].

The following result from [9] can be used to guarantee the safety of a given controlled CCDS.

Theorem 1: Given a controlled CCDS $\mathcal{C} = (\mathbf{f}, \Psi, \Theta)$, with $\mathbf{f}$ defined by (1), a feedback control law $\mathbf{u} = \pi(\mathbf{x})$, and an unsafe region X_u, if an m-dimensional vector of continuously differentiable function $\mathrm{B}(\mathbf{x})$ as per Definition 3 exists, then the safety property of the system (1) is guaranteed.

Throughout this paper, we assume that the domain Ψ, the initial set Θ, and the unsafe set X_u are compact semi-algebraic sets, represented by polynomial equations and inequalities, i.e.,

$$\begin{cases} \Psi := \{\mathbf{x} \in \mathbb{R}^n \mid h_i(\mathbf{x}) \geq 0, 1 \leq i \leq m_1\}, \\ \Theta := \{\mathbf{x} \in \mathbb{R}^n \mid g_j(\mathbf{x}) \geq 0, 1 \leq j \leq m_2\}, \\ X_u := \{\mathbf{x} \in \mathbb{R}^n \mid q_k(\mathbf{x}) \geq 0, 1 \leq k \leq m_3\}, \end{cases} \tag{2}$$

for some polynomials $g_i, h_j, q_k \in \mathbb{R}[\mathbf{x}]$.

III. Verifiable Safe Controller synthesis via Reinforcement Learning

In this section, we propose to synthesize an easily verifiable control policy by combining reinforcement learning (RL) and scenario optimization. RL is a promising tool for designing controllers, especially for complex systems. However, the intricate structure of deep neural network (DNN)

controllers trained through RL often makes their verification over unbounded time horizons challenging, limiting their use in safety-critical applications. To address this issue, we leverage well-performing learned controllers as a guide to design polynomial-type controllers based on Probably Approximately Correct (PAC) principles. Our approach probabilistically characterizes approximation errors and guides the design of polynomial templates for controlled approximation performance. Additionally, the polynomial forms of these controllers enable numerical methods for solving polynomial constraints, facilitating subsequent safety verification of the control system.

A. Training well-performing DNN controllers using RL

Here, we focus on how to train a well-performing controller by RL. Here *well-performing* means that the controller could lead the system to avoid the obstacles within the simulation time bound. Training with RL requires the construction of four components: the state space, the action space, the environment, and the reward function, respectively. For a control task, we take the system state $\mathbf{x}$ which is a vector of n dimensions as the state, the output of the controller u which is of dimension m as the action, and the system dynamics as the environment. Furthermore, the design of the reward function for the safety requirement is explained as follows.

We hope that RL helps to synthesize an ideal controller using the designed reward, and all the trajectories of the closed-loop system starting from the initial set Θ cannot evolve into the unsafe region X_u under the trained DNN controller. Thus, the reward function is preliminarily defined as $\hat{r}_t = \beta_1 \cdot \text{dist}(X_u, \mathbf{x}_t)$, where $\beta_1 > 0$ is the scale factor, and $\text{dist}(X_u, \mathbf{x}_t)$ denotes the distance between the state $\mathbf{x}_t$ and the unsafe region X_u.

In addition, once the trajectory approaches and nearly hits the boundary of the unsafe set, the system behavior should be penalized. For this purpose, the reward function is updated as

$$r_t = \begin{cases} \hat{r}(\mathbf{x}_t) - \min(\beta_2 \cdot \frac{1}{\text{dist}(X_u, \mathbf{x}_t)}, \Delta r_{\min}), & \mathbf{x} \in S_{belt} \\ \hat{r}(\mathbf{x}_t), & \text{otherwise} \end{cases}$$

where $S_{belt} = \{\mathbf{x} \,|\, \text{dist}(X_u, \mathbf{x}) < \delta\}$ with δ a small positive value, $\beta_2 > 0$ is the scale factor and $\Delta r_{\min} > 0$ is the threshold avoiding too large fluctuations of reward value. In this work, we set $\beta_1 = 1.0$, $\beta_2 = 5.0$, $\delta = 0.1$. The r_t set in this way can be kept within a certain range, making the convergence effect better.

The remaining problem is to train the controller via RL. Here we use Deep Deterministic Policy Gradient (DDPG) [1] which is a popular RL approach suited for continuous control applications. The DDPG algorithm combines the value-based and policy-based methods, and is made up of two neural networks: the critic network and actor network. Concretely, the critic network $Q : \mathbb{R}^{n+m} \to \mathbb{R}$ takes the state $\mathbf{x}_t$ and the action $\mathbf{u}_t$ as input, and outputs a scalar Q-value $Q(\mathbf{x}_t, \mathbf{u}_t|\theta^Q)$, which evaluates the current action at the given state $\mathbf{x}_t$ The parameter θ^Q is updated by minimizing L^Q as

$$L^Q = \frac{1}{N} \sum_t (y_t - Q(\mathbf{x}_t, \mathbf{u}_t|\theta^Q))^2, \tag{3}$$

where $y_t = r_t + \gamma Q'(\mathbf{x}_{t+1}, \mathbf{u}'_{RL}(\mathbf{x}_{t+1}))$. The actor network $\mathbf{u}_{RL}$: $\mathbb{R}^n \to \mathbb{R}^m$ receives a state $\mathbf{x}_t$ at time step t, and directly outputs a continuous action $\mathbf{u}_t = \mathbf{u}_{RL}(\mathbf{x}_t|\theta^A)$. The parameter θ^A is updated by minimizing J as

$$J = -\frac{1}{N} \sum_t Q(\mathbf{x}_t, \mathbf{u}_{RL}(\mathbf{x}_t|\theta^A)). \tag{4}$$

Note that in (3) and (4) the Q' and $\mathbf{u}'_{RL}$ are target networks that have the same structures as Q and $\mathbf{u}_{RL}$ respectively; in DDPG, the parameters of Q' and $\mathbf{u}'_{RL}$ are not learned but slowly synchronized with Q and $\mathbf{u}_{RL}$; besides, N is the training batch size, and $\gamma \in (0, 1)$ is the reward decay factor.

To train the desired controller, we first generate a set of initial states from Θ. For each sampled initial state $\mathbf{x}_0$, with the help of $\mathbf{u}_{RL}$, one may yield the associated trajectory as a discrete time state sequence $\{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_t, \cdots, \mathbf{x}_T\}$ which does not enter the unsafe area, and then collect the transition tuples $(\mathbf{x}_t, \mathbf{x}_{t+1}, \mathbf{u}_t, r_t)$ to form a replay buffer. Every few time steps, a batch size of data is sampled from the replay buffer to update the parameters θ^Q and θ^A, and then the new controller is used to simulate the trajectory to collect new data until the controller behaves well.

B. Polynomial controller synthesis based on PAC-model

Based on the trained DNN controller $\mathbf{u}_{RL}(\mathbf{x})$ through RL, we construct an easily verifiable controller with a polynomial form based on Probably Approximately Correct (PAC) principles [11]. The error bound between the PAC-based polynomial controller and the original auxiliary controller $\mathbf{u}_{RL}(\mathbf{x})$ can be evaluated in a statistical way given a pre-defined error rate ϵ and a significance level η.

We first seek to compute an approximate polynomial $p(\mathbf{x}, \mathbf{c}) \in \mathbb{R}[\mathbf{x}]$ with a pre-assigned degree d. Let $[\mathbf{x}]_d$ be the vector consisting of monomials whose degree is at most d, ordered in the graded lexicographic ordering, i.e.,

$$[\mathbf{x}]_d = (1, x_1, x_2, \cdots, x_n, x_1^2, x_1x_2, \cdots, x_{n-1}x_n^{d-1}, x_n^d)^\top.$$

Let the dimension of $[\mathbf{x}]_d$ be ν, i.e., $\nu = \dim([\mathbf{x}]_d) = \binom{n+d}{d}$. Define $\mathbb{N}_d^n$ as $\{\alpha = (\alpha_1, \cdots, \alpha_n) \in \mathbb{N}^n \mid \sum_1^n \alpha_i \le d\}$ and the monomial $\mathbf{x}^\alpha = x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$. Thus, $p(\mathbf{x}, \mathbf{c})$ can be written as

$$p(\mathbf{x}, \mathbf{c}) = \mathbf{c}^T [\mathbf{x}]_d = \sum_{\alpha \in \mathbb{N}_d^n} \mathbf{c}_\alpha \mathbf{x}^\alpha.$$

Essentially, we aim to find a polynomial $p(\mathbf{x}, \mathbf{c})$ that approximates $\mathbf{u}_{RL}(\mathbf{x})$ as closely as possible, which amounts to solve

$$\min_{\mathbf{c}} e \quad s.t.\ |\mathbf{u}_{RL}(\mathbf{x}) - p(\mathbf{x}, \mathbf{c})| \le e, \ \ \forall \mathbf{x} \in \Psi. \tag{5}$$

However, it is impossible for a polynomial of a specified degree to be an arbitrarily close approximation to a DNN within a specified region. In practice, we seek for a polynomial $p(\mathbf{x}, \mathbf{c})$ that approximates $\mathbf{u}_{RL}(\mathbf{x})$ to a high degree of accuracy, that is, with an approximation error less than a threshold τ.

Theoretically, solving (5) is a challenge due to the presence of infinitely many constraints. Calafiore et al. introduced a scenario approach to address this issue, providing a solution with a PAC guarantee in [12].

The scenario approach relaxes the infinitely many constrains in (5) by only considering a finite subset of K constrains. Meanwhile, a PAC guarantee depending on K, between the original solution in (5) is proved in [12] and the following scenario solution in (6).

For pre-specified ϵ and η, we seek a polynomial $p(\mathbf{x}, \mathbf{c})$ with degree d by solving the transformed linear programming from (5) using the scenario optimization approach, i.e., by solving:

$$\min_{\mathbf{c}} e \quad s.t. \ |\mathbf{u}_{RL}(\mathbf{x}_i) - p(\mathbf{x}_i, \mathbf{c})| \leq e, \ \ \forall \mathbf{x}_i \in \widetilde{\Psi}, \qquad (6)$$

where $\widetilde{\Psi}$ is constructed by randomly sampling from Ψ. According to PAC principles [11], there are requirements for the number of sampling points, i.e., $K = |\widetilde{\Psi}|$, to achieve credible approximation error $\epsilon \geq \frac{2}{K}(\ln \frac{1}{\eta} + \kappa)$ where $\kappa = \nu + 1$.

Therefore, through solving the linear programming (6), we obtain a polynomial $p(\mathbf{x}, \mathbf{c}^*)$ which is an approximation of $\mathbf{u}_{RL}(\mathbf{x})$ with a statistical error bound e^* at a given confidence level. Furthermore, if e^* exceeds the specified empirical threshold τ, the error bound between the approximate polynomial and the auxiliary controller $\mathbf{u}_{RL}(\mathbf{x})$ can be further improved by increasing the degree of the polynomial template and re-solving the problem (6), which increases the dimensionality of the feasible domain for the polynomial coefficients.

IV. Learning-aided Safety Verification with Vector Barrier Certificates Generation

We have synthesized a polynomial controller for a controlled CCDS subject to the safety requirement with the aid of RL learning and PAC-model approximation. In this section, we propose to generate a vector BC as stated in Definition 3 to ensure the safety property in formal reasoning. For conventional scalar BC generation, the SOS relaxation technique is often used to effectively encode barrier certificate conditions, transforming them into solvable problems such as LMIs or SDPs. However, for the extended vector BC conditions, directly using SOS encoding causes the VBC2 condition to lead to non-convexity, making subsequent computations unsolvable. To address this difficulty, we design a learning-assisted method for vector BC generation, followed by SOS-based post-verification.

A. Vector Barrier Certificate Learning

The safety verification of polynomial controllers obtained through RL combined with PAC approximation is based on synthesis of the vector BC in Definition 3. Existing methods consider using SOS encoding of the conditions in Definition 3 to solve for the polynomial-form vector BC and verify the system's safety [9]. Due to the non-convexity of the VBC2 constraint in Definition 3, this method requires further strengthening and modifying the constraint as

$$\forall \mathbf{x} \in \mathbb{R}^n.(\mathbf{x} \in X_u \Rightarrow B_{i^\star}(\mathbf{x}) > 0)$$

for some $i^\star \in \{1, \cdots, m\}$, to recover convexity. The introduction of such stronger conditions reduces the feasibility of the constraints with respect to the existence of vector BCs.

We leverage learning techniques to directly obtain vector BCs based on the original conditions and then design an SOS-based post-verification method tailored to disjunctive expressions, thereby avoiding the solution loss caused by strengthened conditions.

1) vector BC network design: To facilitate the use of numerical SOS verification methods in subsequent studies, we consider constructing a polynomial-form vector BC. A straightforward approach is to build a polynomial network by weighting neurons, followed by applying a commonly used *square* activation function. Instead of the polynomial network of with square activation functions, we make use of the DNN jump connection proposed in [13] to construct a Skip polynomial network.

Let $\mathbf{x}^{(0)} \in \mathbb{R}^n$ denote the input of the Square/Skip network, $W_1^{(k+1)}$ and $b_1^{(k+1)}$ be the weight matrix and the bias from the k-th layer to the $(k+1)$-th layer respectively, and $W_2^{(k+1)}$ and $b_2^{(k+1)}$ be the weight matrix and the bias from the 0-th layer to the $(k+1)$-th layer respectively. Square network multiplies the output of the current layer with itself as the input of the next layer, i.e., the recursive relationship two adjacent layers of the Square network can be formally expressed as:

$$\mathbf{x}^{(k+1)} = (W_1^{(k+1)}\mathbf{x}^{(k)} + b_1^{(k+1)})^2.$$

As can be seen from the module of the *skip polynomial network* in Fig. 1, the Skip network directly multiplies the output of the current layer with the input layer as the input for the next layer. Thus the relationship between two adjacent layers of the Skip network can be expressed as:

$$\mathbf{x}^{(k+1)} = (W_1^{(k+1)}\mathbf{x}^{(k)} + b_1^{(k+1)}) \otimes (W_2^{(k+1)}\mathbf{x}^{(0)} + b_2^{(k+1)}),$$

where $\otimes$ is the Hadamard product.

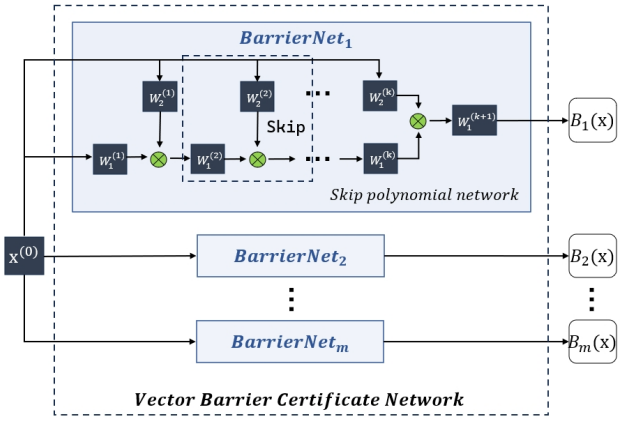


Fig. 1. Structure of vector BC Network.

Compared with Square network, the Skip network is more expressive by introducing two different weights and bias parameters on each layer. In this paper, we make use of Skip polynomial network as a basic module to construct a Skip

polynomial vector network (see Fig. 1) and Skip polynomial matrix network for vector BC and multiplier A respectively.

2) loss function design: To learn a vector BC that satisfies the three constraints in Definition 3, the designed vector BC network structure is utilized to learn $\mathrm{B}(\mathbf{x},\theta) = (B_1(\mathbf{x},\theta_1),\cdots,B_m(\mathbf{x},\theta_m))^\top$ and the auxiliary multiplier matrix $A(\Lambda)$, thereby obtaining parameters θ and Λ that satisfy the safety constraints. The vector BC parameters are learned by minimizing the loss function:

$$\mathcal{L}(\theta,\Lambda) = \mathcal{L}_{init}(\theta) + \mathcal{L}_{Unsafe}(\theta) + \mathcal{L}_d(\theta,\Lambda),$$

where $\mathcal{L}_{init}(\cdot)$, $\mathcal{L}_{Unsafe}(\cdot)$ and $\mathcal{L}_d(\cdot)$ are loss sub-functions designed to satisfy VBC1, VBC2, and VBC3, respectively.

The training dataset is constructed by randomly sampling from the system's initial set Θ, unsafe set X_u and system domain Ψ, denoted as $\hat{\Theta}$, $\hat{X}_u$ and $\hat{\Psi}$, respectively. *For condition VBC1*, it requires that $\forall \mathbf{x}\in\Theta.B_i(\mathbf{x})\le 0, 1\le i\le m$. To encode this condition, we design the loss sub-function $\mathcal{L}_{init}$ as follows: $\sum_{x_i\in\hat{\Theta}} Relu(\max\{B_1(x_i),...,B_m(x_i)\})$. *For condition VBC2*, it requires that $\forall \mathbf{x}\in X_u$, $B_i(\mathbf{x})>0$ for some $i\in\{1,\cdots,m\}$. Then, we design the loss sub-function $\mathcal{L}_{Unsafe}$ as follows: $\sum_{x_i\in\hat{X}_u} Relu(-\max\{B_1(x_i),...,B_m(x_i)\})$. *For condition VBC3*, it requires that $\mathcal{L}_{\mathbf{f}}\mathrm{B}(\mathbf{x}) \le A\mathrm{B}(\mathbf{x})$ on the system domain Ψ, i.e., $\mathcal{L}_{\mathbf{f}}B_i(\mathbf{x}) - A_{i:}\mathrm{B}(\mathbf{x}) \le 0$ for $\mathbf{x}\in\Psi$, and thus we design the loss sub-function $\mathcal{L}_d$ as: $\sum_{x_i\in\hat{\Psi}} Relu(\max_{1\le j\le m}\{\mathcal{L}_{\mathbf{f}}B_j(x_i) - A_{j:}\mathrm{B}(x_i)\})$. Then, the *AdamW optimizer* is applied to minimize $\mathcal{L}(\theta,\Lambda)$ subject to the constraints that all the off-diagonal entries of A are non-negative to generate a vector barrier function network candidate $\mathrm{B}(\mathbf{x})$.

Example 1: Consider the following closed-loop system

$$\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}x_1^2+x_2^2-1\\ 5(x_1x_2-1)\end{bmatrix},$$

with $\Psi = \{x_1,x_2\in\mathbb{R}\,|\,-4\le x_1\le 4,-4\le x_2\le 4\}$, $\Theta = \{x_1,x_2\in\mathbb{R}\,|(x_1-0.6)^2+(x_2-0.2)^2\le 0.5^2\}$, and $X_u = \{x_1,x_2\in\mathbb{R}\,|-1.8\le x_1\le 2, 1\le x_2\le 2\}$, for which we learned a vector BC $(B_1,B_2)^\top$ with $B_1(\mathbf{x}) = -1.284x_1 + 0.519x_2 - 0.665$ and $B_2(\mathbf{x}) = 0.149x_1 + 0.619x_2 - 0.542$. Please see Fig. 2 for an illustration.

B. Posterior Verification

In this subsection, we verify that the learned vector barrier certificate candidate $\mathrm{B}(\mathbf{x}) = (B_1(\mathbf{x}),B_2(\mathbf{x}),\cdots,B_m(\mathbf{x}))^\top$ strictly satisfies the conditions VBC1, VBC2, and VBC3. This is achieved by transforming the verification problems into global optimization problems over polynomials, which can be solved efficiently via SOS-encoding and semidefinite program solving.

Given a compact basic semi-algebraic set $\mathbb{K}$ defined by $\mathbb{K} = \{\mathbf{x}\in\mathbb{R}^n\,|\,\gamma_1(\mathbf{x})\ge 0,\ldots,\gamma_\ell(\mathbf{x})\ge 0\}$, where $\gamma_j(\mathbf{x})\in\mathbb{R}[\mathbf{x}]$ for $1\le j\le\ell$, the global optimization problem $\min_{\mathbf{x}\in\mathbb{K}} f(\mathbf{x})$ is well defined for any polynomial $f(\mathbf{x})\in\mathbb{R}[\mathbf{x}]$. It is not difficult to check that

$$\forall\mathbf{x}\in\mathbb{R}^n.(\mathbf{x}\in\mathbb{K}\Rightarrow f(\mathbf{x})\triangleright 0) \equiv \min_{\mathbf{x}\in\mathbb{K}} f(\mathbf{x})\triangleright 0 \quad , \qquad (7)$$

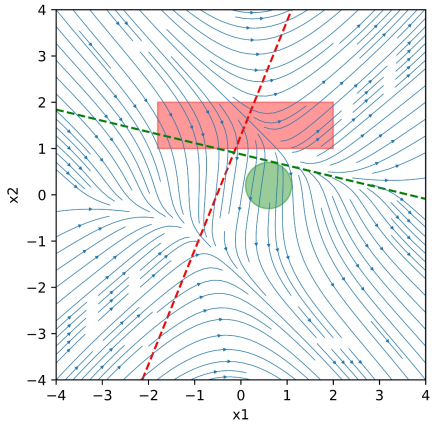


Fig. 2. An illustrative example with the learned vector barrier certificate $(B_1,B_2)^\top$. The outer white square, the green circle, and the red rectangle represent Ψ,Θ and X_u, respectively. The red and green dotted lines represents the level sets of B_1 and B_2, respectively, which are formally verified by SOS programming. The blue curves with arrows represent the system trajectories.

where $\triangleright\in\{>,\ge\}$, and $\equiv$ means the two constraints on both sides are equivalent. By [14], $\min_{\mathbf{x}\in\mathbb{K}} f(\mathbf{x})$ can be well approximated from below as closely as possible by solving a hierarchy of SOS-programs

$$\max_{r\in\mathbb{R},\,\sigma_i\in\Sigma[\mathbf{x}]} r \quad \text{s.t. } \big(f(\mathbf{x})-r-\sum_{i=1}^{\ell}\sigma_i(\mathbf{x})\gamma_i(\mathbf{x})\big) \text{ is SOS}, \quad (8)$$

with increasing degree bounds for σ_i.

Combining Definition 3 and formulas (2) and (7), we can see that VBC1 is equivalent to

$$\min_{\mathbf{x}\in\Theta} -B_i(\mathbf{x}) \ge 0, \quad 1\le i\le m,$$

which can be checked by solving m optimizations as (8); similarly, VBC3 is equivalent to

$$\min_{\mathbf{x}\in\Psi}(A_{j:}\mathrm{B}(\mathbf{x}) - \mathcal{L}_{\mathbf{f}}B_j(\mathbf{x})) \ge 0, \quad 1\le j\le m,$$

which can again be checked via m optimizations as (8).

Verification of VBC2 is a bit more complicated, which will be conducted iteratively as the following four steps:

- Initially set $i=1$ and $X = X_u$;
- Compute $r_i = \min_{\mathbf{x}\in X} B_i(\mathbf{x})$ by (8);
- If $r_i > 0$ then we are done; otherwise, update X by $X\wedge B_i(\mathbf{x})\le 0$ and increment i by 1;
- If $i\le m$, go to the second step; otherwise we fail.

Therefore at most m optimizations as (8) need to be solved.

We have successfully applied the SOS-based approach to formally verify the learned vector BC of Example 1.

V. Experiments

In this section, we conduct and report an experimental evaluation of our method over a set of benchmark examples. We further conduct experiments to compare our method with a mainstream SOS-based BC generation approach in terms of safety verification performance. All experiments are conducted

on a machine with Ubuntu 22.04, 256GB RAM, dual AMD EPYC 7302 CPUs, and three NVIDIA RTX A6000 GPUs.

The original conditions for the vector BCs we adopted were first proposed in [9]. Compared to [9], their method is not applicable to the safe controller generation problem. To facilitate computation, they strengthened the original vector BC conditions. Furthermore, when addressing the existence of vector BCs, their approach requires manually fixing a key multiplier matrix A in advance. However, as they discussed in their paper, there is no well-established guidance for A's setting. Since the setting of A remains unclear, we do not conduct a detailed comparison with [9]. Moreover, they only reported a case study on one 3-dimensional verification problem.

We present a detailed experimental evaluation on a set of widely used benchmarks C_1 to C_{10} in Table I, using our proposed Learning-Aided Safe Controller Synthesis with vector BC method (*Lasac_VBC*). To further demonstrate the verification effectiveness of the learning-assisted vector BC generation in our *Lasac_VBC*, we compared it with the conventional approach of scalar BC generating using *SOSTOOLS* [15] for safety verification.

The origins of these 10 widely used cases are provided in the first column; $n_{\mathbf{x}}$ and $d_{\mathbf{f}}$ denote the number of state variables and the maximal degree of the polynomials (or the polynomial abstractions for non-polynomial systems) in the vector fields, and the examples are with dimension up to 12.

For each control task, *Lasac_VBC* first obtains a well-performing controller through RL and PAC approximation. The auxiliary network structure information and the corresponding polynomial degree of the controller are recorded in the 'NN $struc.$' and '$d_{contr.}$' columns, respectively. Here, all our used NNs are with ReLU activation functions. Secondly, the results of the learned vector BC, including its dimension (combination parameter m) and the degree of each B_i, are recorded in the 'm_{VBC}' and 'd_{VBC}' columns, respectively. Finally, the learned vector BC is subjected to post-verification, and the verification time cost is recorded in the '$T_{VBC}(s)$' column if successful. Meanwhile, in the comparison of safety verification performance, *SOSTOOLS* uses the safe controller learned by *Lasac_VBC* for each control case. *SOSTOOLS* starts the search with a low-degree polynomial template, incrementally increasing the polynomial degree until a barrier certificate is found. The resulting degree and computation time are recorded in the 'd_B' and '$T_B(s)$' columns, respectively. If the search fails to find a barrier certificate after attempting up to a degree 6 polynomial template, it is reported as a verification failure by '$\times$'.

Observing Table I, our *Lasac_VBC* successfully synthesizes safe controllers with vector BC guarantees for all examples including a complex 12-dimensional system. In contrast, *SOSTOOLS* serves only as a safety verification method and fails to verify C_8, C_9 and C_{10}. Comparing the safety verification performance: for examples C_1-C_7, the time efficiency of the two methods can be compared by observing the '$T_{VBC}(s)$' and '$T_B(s)$' columns. It is evident that for cases where *SOSTOOLS* can also perform verification, *Lasac_VBC* consistently achieves better time efficiency than *SOSTOOLS*.

TABLE I
PERFORMANCE EVALUATION OF OUR METHOD

Benchs.	$n_{\mathbf{x}}$	$d_{\mathbf{f}}$	*Lasac_VBC*					*SOSTOOLS*	
			NN $struc.$	$d_{contr.}$	m_{VBC}	d_{VBC}	$T_{VBC}(s)$	d_B	$T_B(s)$
C_1 [16]	2	2	2-64(2)-1	2	2	1,1	0.377	2	3.241
C_2 [16]	2	2	2-64(2)-1	2	2	1,1	0.343	2	3.997
C_3 [17]	3	3	3-64(2)-1	2	2	1,1	1.258	2	4.867
C_4 [18]	3	2	3-64(2)-1	2	2	1,1	0.766	2	4.652
C_5 [19]	4	3	4-64(2)-1	2	2	2,2	1.900	2	6.584
C_6 [20]	5	2	5-64(2)-1	2	2	1,1	3.574	2	8.810
C_7 [17]	6	3	6-64(2)-1	2	2	2,2	11.702	2	12.833
C_8 [21]	7	2	7-64(2)-1	2	2	1,1	10.626	$\times$	$\times$
C_9 [21]	9	2	9-64(2)-1	2	2	1,2	31.454	$\times$	$\times$
C_{10} [22]	12	1	12-64(2)-1	2	2	1,1	107.999	$\times$	$\times$

Further examining the 'd_{VBC}' and 'd_B' columns in Table I, in all examples, the vector BCs successfully generated by *Lasac_VBC* are combinations of low-degree BCs, with their degrees not exceeding those obtained by *SOSTOOLS*. Additionally, in most examples, the degrees of our results are lower than those obtained by *SOSTOOLS*.

This indicates that the vector BCs generated by *Lasac_VBC*, through the vector form as a combination of multiple BCs, can achieve the same expressiveness that conventional single BCs require higher degrees to attain. The use of multiple low-degree functions for verification, compared to high-degree complex functions, offers potential advantages in terms of verification computational efficiency and the scale of problems that can be handled. The superior performance of *Lasac_VBC* over *SOSTOOLS* in terms of experimental verification efficiency and the scale of verification problems precisely demonstrates this point.

To summarize, Table I shows that our proposed *Lasac_VBC* demonstrates potential advantages in learning-assisted safe controller generation, verification computational efficiency, and the scale of problems it can handle.

VI. CONCLUSION

This paper addresses the problem of controller generation for safety-critical systems by skillfully leveraging RL, PAC techniques, and supervised learning to assist in the generation of formally guaranteed safe controllers. Benefiting from a controller synthesis method that combines RL and PAC approximation techniques, our approach yields system control policies with good behavioral performance. Additionally, learning techniques are utilized to aid in generating vector BCs with higher expressiveness, reducing verification costs while avoiding the drawbacks of conventional methods that strengthen constraints, causing the loss of potential feasible solutions for vector BCs. Especially, we propose an incremental SOS-encoding method to transform the formal verification of vector BC conditions into a series of semi-definite programs, enabling highly efficient solving. The experimental results demonstrate the advantages of the proposed method in learning-assisted controller synthesis and verification.

REFERENCES

[1] T. P. Lillicrap, J. J. Hunt, A. Pritzel, and et al., "Continuous control with deep reinforcement learning," *Computer science*, 2015.

[2] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International conference on machine learning*. PMLR, 2016, pp. 1329–1338.

[3] A. S. Poznyak, E. N. Sanchez, and W. Yu, *Differential neural networks for robust nonlinear control: identification, state estimation and trajectory tracking*. World Scientific, 2001.

[4] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2004, pp. 477–492.

[5] H. Kong, F. He, X. Song, W. N. Hung, and M. Gu, "Exponential-condition-based barrier certificate generation for safety verification of hybrid systems," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 242–257.

[6] L. Dai, T. Gan, B. Xia, and N. Zhan, "Barrier certificates revisited," *Journal of Symbolic Computation*, vol. 80, pp. 62–86, 2017.

[7] C. Sloth, G. J. Pappas, and R. Wisniewski, "Compositional safety analysis using barrier certificates," in *Proceedings of the 15th ACM international conference on Hybrid Systems: Computation and Control*, 2012, pp. 15–24.

[8] X. Zeng, W. Lin, Z. Yang, X. Chen, and L. Wang, "Darboux-type barrier certificates for safety verification of nonlinear hybrid systems," in *Proceedings of the 13th International Conference on Embedded Software*, 2016, pp. 1–10.

[9] A. Sogokon, K. Ghorbal, Y. K. Tan, and A. Platzer, "Vector barrier certificates and comparison systems," in *Formal Methods - 22nd International Symposium, FM 2018*, ser. Lecture Notes in Computer Science, vol. 10951. Springer, 2018, pp. 418–437.

[10] W. Walter, "Differential inequalities and maximum principles: theory, new methods, and applications," *Nonlinear Analysis: Theory, Methods & Applications*, vol. 30, no. 9, pp. 4695–4711, 1997.

[11] G. Calafiore and M.Campi, "The scenario approach to robust control design," *IEEE Transactions on automatic control*, vol. 51, no. 5, pp. 742–753, 2006.

[12] M. Campi, S. Garatti, and M. Prandini, "The scenario approach for systems and control design," *Annual Reviews in Control*, vol. 33, no. 2, pp. 149–157, 2009.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2016, pp. 770–778.

[14] J. B. Lasserre, *An Introduction to Polynomial and Semi-Algebraic Optimization*, ser. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2015, vol. 52.

[15] S. Prajna, A. Papachristodoulou, and P. A. Parrilo, "Sostools: Sum of squares optimization toolbox for matlab," 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:56539680

[16] A. Sogokon, K. Ghorbal, and T. T. Johnson, "Non-linear continuous systems for safety verification (benchmark proposal)," in *ARCH@ CPSWeek 2016-3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, vol. 43. EasyChair, 2016, pp. 42–51.

[17] X. Chen, C. Peng, W. Lin, Z. Yang, Y. Zhang, and X. Li, "A novel approach for solving the bmi problem in barrier certificates generation," in *International Conference on Computer Aided Verification*. Springer, 2020, pp. 582–603.

[18] J. V. Deshmukh, J. P. Kapinski, T. Yamaguchi, and D. Prokhorov, "Learning deep neural network controllers for dynamical systems with safety guarantees," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[19] G. Chesi, "Computing output feedback controllers to enlarge the domain of attraction in polynomial systems," *IEEE Transactions on Automatic Control*, vol. 49, no. 10, pp. 1846–1853, 2004.

[20] M. A. B. Sassi and S. Sankaranarayanan, "Stabilization of polynomial dynamical systems using linear programming based on bernstein polynomials," *arXiv preprint arXiv:1501.04578*, 2015.

[21] E. Klipp, R. Herwig, A. Kowald, C. Wierling, and H. Lehrach, *Systems biology in practice: concepts, implementation and application*. John Wiley & Sons, 2005.

[22] S. Gao, "Quadcopter model," *[online] https://github.com/dreal/benchmarks*, 2016.

Secondary-Power-Cell-Aware Detailed Placement in Multiple Power Domain Designs

Yu-Wei Chang[1], Shao-Yun Fang[1], Kai-Chuan Yang[2], and Min-Ching Lin[2]

[1]Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan
[2] Synopsys, Inc., Taipei 106, Taiwan
{M11107415, syfang}@mail.ntust.edu.tw

***Abstract*—Using multiple power domains is a common technique to achieve significant power savings in modern designs. However, providing power to *cross-domain cells* is challenging at advanced technology nodes, due to strict IR drop constraints and competition for routing resources between secondary power routing and signal routing. Despite its importance, this issue has received limited attention in recent literature. In this paper, we propose a detailed placement method that accounts for secondary power routing. Our approach employs an integer linear programming (ILP) model, complemented by targeted adjustments to the placement of cross-domain cells, to optimize the efficiency of secondary power routing. We validate our results with commercial design tools, showing that our approach effectively improves secondary power routing performance by reducing the wirelength of the secondary power distribution network (PDN) and decreasing violations due to secondary PDN constraints.**

I. Introduction

Low power has become one of the most important features in many electronic products, with multiple supply voltages (MSVs) and power domains being widely used to achieve reduced power consumption. MSVs control circuit sections independently to reduce dynamic and leakage power, while power gating boosts efficiency by shutting off idle domains. In multiple power domain designs, cross-domain cells, such as always-on buffers/inverters, retention flip-flops, and isolation cells, are essential for enabling power domain shutdowns to achieve power savings. These cells serve crucial roles, such as level shifting, transmitting default values, and ensuring that the design continues to function correctly even when certain power domains are turned off. Through these functions, cross-domain cells support the seamless operation of multiple power domain designs under varying power conditions.

These cells have both a *primary power pin*, shown in light blue of left cell in Fig. 1, and a *secondary power pin*, shown in gray. The primary power pin connects to the *primary power delivery network (primary PDN)*, supplying power from the primary power source during regular operation, shown in pink net. Meanwhile, the secondary power pin connects to the *secondary power delivery network (secondary PDN)* to provide power when the primary power domain is shut down, enabling continuous operation of the cross-domain cells, shown in dark blue nets. Each cross-domain cell belongs to one power domain but also connects to a secondary PDN, ensuring the independence between the two power domains, both of which supply power to the cross-domain cell as needed. In this paper, we refer to the ordinary power-supplying power domain of a cross-domain cell as the *primary power domain (PPD)*, while the secondary PDN and all cells connected to it are collectively referred to as the *secondary power domain (SPD)*. The term *power domain* will refer broadly to the combined set of both primary and secondary power domain.

In multiple power domain designs, when cells from different power domains or cross-domain cells are present within the same block, it is necessary to interleave power stripes for each power domain [11]. This arrangement allows each cell to quickly connect to the appropriate power stripe for its domain. Additionally, a design can contain multiple SPDs, with each SPD having its own independent secondary PDN that does not connect to others. Each cross-domain cell connects to only one secondary PDN, thus belonging to a single SPD as a backup power source. As shown in Fig. 2(a), the orange section represents the primary power stripe for the area, while the blue and green stripes represent power stripes for two different SPDs. This interleaved configuration helps cells connect more efficiently to their designated power stripes, reducing wire length and connection complexity. In addition, Fig. 2(b) shows the conventional way of connecting the cells to the primary PDN. Since all cells in this area share the same primary power domain, regular power and ground rails are used. Figs. 2(c) and (d) respectively illustrate the secondary PDN and cross-domain cells belonging to SPD 1 and 2, respectively. The secondary PDNs connect the gray secondary power pins of cross-domain cells to the secondary power stripes. Note that the trunk-based secondary PDN structures are preferred, which enhance IR drop performance and power stability and decrease routing complexity and the likelihood of routing violations. Each orange pin represents a signal pin, which is not connected during the secondary power routing phase but may suffer from violations if covered by some secondary PDN.

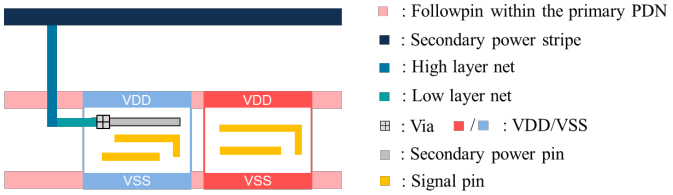


Fig. 1. The architecture of the secondary PDN and cross-domain cells. The blue cell represents a cross-domain cell connected to the secondary PDN, while the red cell represents a general cell.

Multiple power domain designs have been widely explored in the literature. Studies such as [9], [10], [11] have researched PDN optimization for multiple power domain designs. More recent work has also examined various power-aware considerations in placement and floorplanning domains [12], [13]. However, these studies primarily focus on primary PDN and do not extend their considerations to secondary PDN. In contrast, research addressing cross-domain cells has largely focused on power gating transitions [7], [8], buffering within cross-domain designs [6], [5], and rule-checking [4]. However, they lack emphasis on placement or routing for PDN optimization. To the best of our knowledge, the only existing study specifically addressing secondary power mesh improvements is [14]. This work develops an IR drop prediction model and a design rule violation prediction model to guide the construction of sub-meshes for cross-domain cells. However, the approach only considers mesh-based secondary PDNs, consuming more routing resources for the low-density regions. In addition, [14] optimizes secondary PDNs for a given cell placement result; however, the placement of cross-domain cells could greatly affect both the PDN complexity and power performance.

This work was partially supported by TSMC, Synopsys, MPI, and NSTC of Taiwan under Grant No's NSTC 113-2927-I-011-502, 113-2640-E-002-001, 113-2640-E-006-001, 113-2222-E-011-005-MY3, and 114-2927-I-011-501.

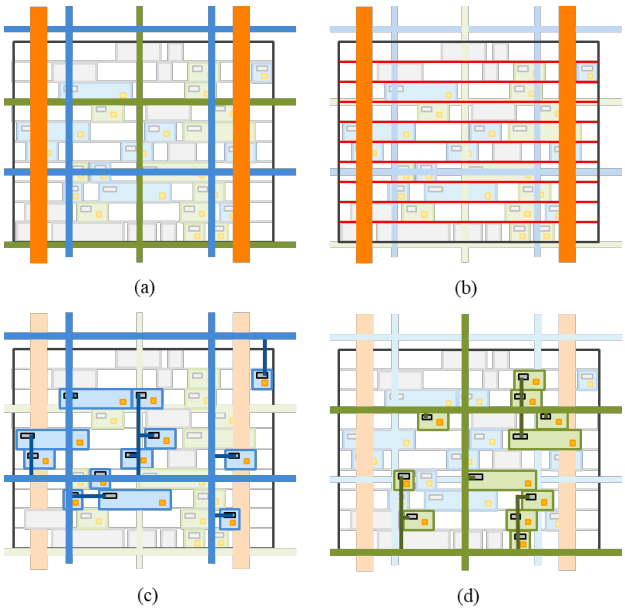


Fig. 2. An example of cross-domain cells and secondary PDN architecture. (a) The structure of power stripes in the design, where the orange stripes represent the primary PDN, and the blue and green stripes represent two different SPDs. (b) Primary PDN within this power domain. (c) The PDN and cells in SPD 1. (d) The PDN and cells in SPD 2.

In this paper, we propose a secondary power routing-aware detailed placement to refine the positions of cross-domain cells, facilitating better secondary power routing with a commercial tool. The main contributions of this paper are as follows:

- This is the first study to address reducing violations caused by secondary PDNs and minimizing wirelengths of tree-based secondary PDNs in the placement stage.
- We propose an integer linear programming (ILP) secondary-power-routing-aware placement refinement algorithm, followed by a trapped cell relocation technique and a pin alignment procedure.
- The proposed placement algorithm is validated by using a commercial tool, and the results demonstrate that the optimized placements reduce the critical secondary PDN wirelength by 11%, the total secondary PDN wirelength by 30%, and the number of violations by over 66%.

The remainder of this paper is structured as follows: Section II introduces the key challenges of secondary power routing and defines the problem. Section III details the proposed solution and algorithm. Section IV presents the reductions in wirelength and design rule violations achieved through our placement approach, and Section V concludes the study.

II. Potential Issues and Solution Concepts for Secondary Power Routing

In the following, Section II-A will discuss the negative effects that secondary PDNs introduce to the design, along with the physical factors that contribute to these issues. In Section II-B, we will present conceptual solutions and strategies to address these problems, and clearly define the problem at the end of the section.

A. Causes of Violations due to Secondary PDNs

Fig. 3(a) provides an example showing how design rule violations can be caused due to the connection patterns of secondary PDNs. In the figure, the blue and green sections represent two independent SPDs, each with its secondary PDN and cross-domain cells. The gray part within each cell indicates the secondary power pin, while the orange part represents the signal pin. Because the typical routing order begins with primary PDN, followed by secondary power routing and then signal routing, the routing of secondary PDNs can directly impact signal routing, resulting in violations without the awareness of signal nets.

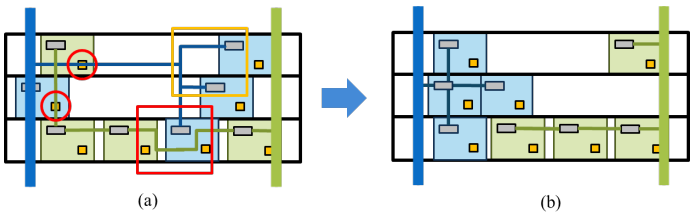


Fig. 3. Examples of potential non-ideal scenarios involving multiple secondary PDNs. (a) A given placement instance. (b) The optimized placement after cell rearrangement.

The resulting violations fall into two main types: the first occurs when connections in secondary power routing cannot be completed, and the second arises when secondary PDNs interfere with signal routing. Connections for secondary PDNs typically do not present issues due to the availability of complete routing resources before signal routing. In contrast, signal routing can greatly be affected by secondary PDNs due to its lower priority. The primary reasons for signal routing failure due to secondary PDN are as follows. First, if a secondary PDN directly covers a signal pin, it prevents any connection to the signal pin, resulting in a violation, as shown by the two red circles in Fig. 3(a). The second reason is that if the secondary PDN occupies too many routing resources, the signal network may lack sufficient resources for routing. This situation can lead to an increase in signal network wire length and even cause incomplete routing with violations.

To address this problem, rearranging cross-domain cells associated with different secondary PDNs can serve as a practical solution. If cells belonging to the same secondary domain are spaced apart, detours can be caused during secondary power routing. For instance, as illustrated in Fig. 3(a), the green cells within the red box are separated by the blue cell, leading to a detour in the green secondary PDN. In addition, if a cross-domain cell is surrounded by cells from different secondary PDNs, it could be obstructed from connecting to its own secondary power stripes, resulting in violations. Therefore, by grouping cells belonging to the same secondary domain together and placing each cell close to its *target secondary power stripe (TSPS)*, detours and obstructions could be effectively reduced, thereby improving both the difficulty and quality of secondary power routing. The TSPS of each cell is simply the closest secondary power stripe to its secondary power pin. Moreover, the arrangement of secondary power pins also plays a crucial role. It is generally desirable to minimize the number of wire bends in PDN routing to facilitate easier connections and power performance. For instance, as the blue cells within the orange box in Fig. 3(a), aligning their secondary power pins as shown in Fig. 3(b) can significantly reduce the secondary PDN wirelength and detours, thereby enhancing the likelihood of successful routing.

In addition to violations, secondary PDN wirelength significantly affects design performance, and both *the critical secondary PDN wirelength (WL^c_{sec})* and *the total secondary PDN wirelength (WL^t_{sec})* should be minimized. WL^c_{sec} refers to the wirelength of the longest secondary PDN path, impacting IR drop and performance, while WL^t_{sec} is the total wirelength of all secondary PDNs. Excessive WL^t_{sec} can degrade power performance and consume resources needed for subsequent signal routing, leading to longer signal wirelengths, increased routing difficulty, and more violations. Based on these factors, the objective function for improving secondary PDNs can be expressed as follows:

$$\text{Minimize} \qquad \#vios + \alpha \cdot WL^t_{sec} + \beta \cdot WL^c_{sec} + \gamma \cdot WL_{sig}, \quad (1)$$

where $\#vios$ represents the number of violations, and WL_{sig} represents the total wirelength of signal nets. α, β, and γ are user-defined

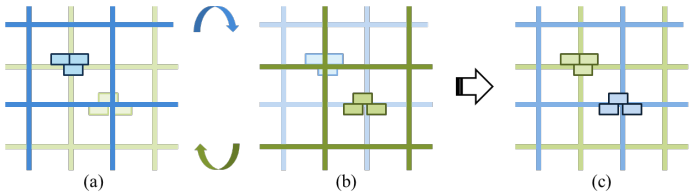


Fig. 4. Diagram of critical cell placement toward its TSPS. (a) Critical cells and power stripe of SPD 1. (b) Critical cells and power stripes of SPD 2. (c) Ideal placement after swapping the locations of the critical cells in (a) and (b).

constants, typically much less than one ($\alpha, \beta, \gamma \ll 1$), to prioritize violation reduction.

B. Operations for Placement Optimization

Equation (1) represents the objective function for the entire design, but it cannot be directly applied to the placement algorithm, as the number of violations and wirelengths cannot be directly derived during placement. Therefore, we explain how to achieve approach this goal using a placement algorithm. As mentioned above, the primary way to eliminate violations is to move cross-domain cells closer to their TSPSs. This approach has two main benefits: it reduces secondary PDN wirelength and aggregates the cells of the same secondary power domain closer together. In addition, after examining multipe designs, we found that connections that can greatest impact signal routing are usually the cross-domain cells farthest from their TSPSs. We define such cells as *critical cross-domain cells* (referred to as *critical cells* hereafter). Critical cells are identified as the top-x cells with the longest distances to their TSPSs.

Three operations can be used to rearrange the cross-domain cells for secondary PDNs. First, it is straightforward to directly move a critical cell c_i with large ζ_i toward its TSPS. Besides, since each cross-domain cell has a different TSPS, using cell swapping for placement has the potential to yield a win-win result. Fig. 4 shows an example, where Figs. 4(a) and (b) represent two independent SPDs, and their critical cells lie near the power stripes of the other secondary power domain. Directly swapping the cells from different secondary power domains results in a much better placement result, allowing both to be positioned closer to their respective TSPSs, as illustrated in Fig. 4(c). Furthermore, in addition to reducing the distance between a cell and its TSPS, secondary power routing difficulty can also be minimized by aligning secondary power pins or relocating cells that are surrounded by cells from other SPD. The detailed explanations of adopting these operations in our methodologies will be provided in Section III.

C. Problem Formulation

Given a multiple power domain design consisting of primary PDN, a set of secondary PDNs, a set of standard cells, a set of cross-domain cells, a legalized placement result, and a netlist containing all connection relationships, the objective of this work is to perform secondary power routing-aware placement refinement for the design to minimize the total secondary PDN wirelength, the critical secondary PDN wirelength, the total number of design rule v iolations, and the overhead in signal wirelength.

III. Proposed Flow

In the following section, we will provide a detailed description of how we use placement strategies to facilitate secondary power routing and signal routing and explain how we validate the effectiveness of the entire approach.

A. Secondary Power Optimization Framework

Fig. 5 illustrates the flowchart of our algorithm, divided into two main parts: the secondary-power-routing-aware placement stage (placement stage) and the routing and verification stage (routing stage). The

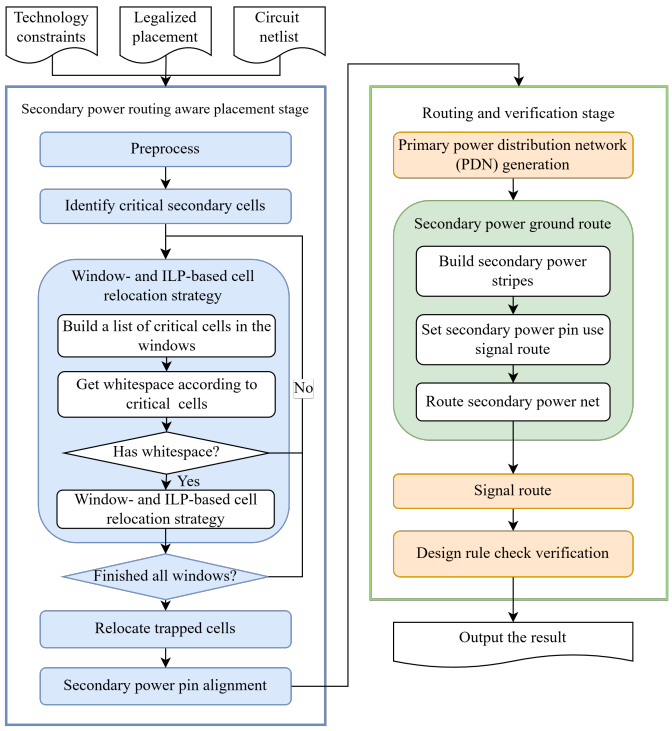


Fig. 5. The overall flow of the proposed framework

placement stage focuses on detailed placement to address violations caused by secondary PDNs, while the routing stage uses a commercial tool for secondary power routing and design rule check (DRC) to validate the effectiveness and feasibility of the placement algorithm.

The flow begins with data loading and preprocessing to establish data structures, followed by identifying critical cells among cross-domain cells based on their distances to their TSPSs. These cells undergo placement with a window-based secondary-power-routing-aware placement refinement procedure with an ILP formulation. Next, all cross-domain cells are processed to relocate trapped cells, and secondary power pins are aligned to ensure low-complexity routing patterns. In the routing stage, we use a commercial tool to generate the primary PDN, followed by secondary PDN generation. Signal routing is then performed to evaluate the impact of cross-domain cell replacement and secondary power routing. The final step is DRC verification, assessing the effectiveness of the proposed methodologies in resolving secondary power routing issues and providing quantitative comparison data. We detail each major invention of this work on placement refinement in the following subsections.

B. Window- and ILP-based Cell Relocation Strategy

In this stage, our primary goal is to place critical cells near their respective TSPSs. The ILP-based algorithm operates in an iterative manner, identifying critical cells and available white spaces within each window and pairing them for re-placement in a fill-in-the-blank style until all iterations are complete. Each iteration has a set time limit for solving to avoid runtime explosion. Details of each component are explained below.

1) Critical Cell and White Space Identification: The definition of critical cells is based on the distance between cross-domain cells and their TSPSs. In this study, we identify the top 25% of cells with the longest distances as critical cells for subsequent placement refinement. We define a *white space* as an empty row segment, which includes unoccupied areas and the original positions of critical cells. One of the advantages of considering the original positions of critical cells is to enable multi-way cell swapping. In addition, this characteristic allows the ILP-based algorithm to ensure that the solution space encompasses the original design, guaranteeing that the resulting performance is not worse than that of the initial design.

TABLE I
SYMBOLS USED IN THE WINDOW- AND ILP-BASED CELL RELOCATION STRATEGY

Variable	Definition
C^{cri}	The set of critical cells.
c_i	The i-th cell in C^{cri}.
N	The set of nets.
N_i	The set of nets connected to c_i.
n_{ij}	The j-th net in N_i.
$HPWL_{inc}$	Total increased HPWL of signal nets connected to critical cells.
Z	Total distance of all critical cells to their TSPS.
α	Ratio number of $HPWL_{inc}$ and Z.
x_i/y_i	The coordinate of a location candidate of c_i.
$\mathrm{M_{xij}}/\mathrm{M_{yij}}$	The maximum x/y coordinate values among cells connected to n_{ij}.
$\mathrm{m_{xij}}/\mathrm{m_{yij}}$	The minimum x/y coordinate values among cells connected to n_{ij}.
$incX_i/incY_i$	The increase in HPWL in the x/y direction caused by c_i.
$d_i^t/d_i^b/d_i^l/d_i^r$	Distance from secondary power pin of c_i to the top/bottom/left/right secondary power stripe.
$b_i^t/b_i^b/b_i^l/b_i^r$	A Boolean variable used to determine whether c_i connects to the top/bottom/left/right secondary power stripe.
ζ_i	The distance between the critical cell i and the TSPS.
$p_i^t/p_i^b/p_i^l/p_i^r$	The y- or x-coordinate of the top/bottom/left/right secondary power stripe of c_i.
x_i^{sec}/y_i^{sec}	The location of cell c_i's secondary power pin.
M	A large constant.
S	The set of white spaces.
s_k	The k-th white space in S.
$e_{i,k}$	A Boolean variable to determine whether c_i is in s_k.
row_i^c	The row of c_i.
$\mathrm{row_k^s}$	The row of s_k.
$\mathrm{minX_k^s}$	Minimum x-values of s_k.
$\mathrm{maxX_k^s}$	Maximum x-values of s_k.
w_i^c	The width of c_i.
w_k^s	The width of s_k.
$d_{i,k}$	A Boolean variable. $d_{i,k}=0$ indicates row_i^c and $\mathrm{row_k^s}$ is equal; $d_{i,k}=1$, otherwise.
$g_{i,j}$	A Boolean variable. $g_{i,j}=0$ indicates c_i is on the right side of c_j; $g_{i,j}=1$, otherwise.

2) ILP Objective Function: After identifying the critical cell list and the white space list, we can proceed to match them using the ILP algorithm. Table I presents the notations and descriptions used in the ILP formulation, and Equation (2) shows the objective function:

$$\text{Minimize} \qquad \alpha \times HPWL_{inc} + (1-\alpha) \times Z. \tag{2}$$

In the placement process, we utilize HPWL (Half-Perimeter Wire Length) to model the wirelength of signal nets, and we denote the total increase in signal wirelength due to placement refinement as $HPWL_{inc}$. Z is calculated as the total straight-line distance from the center of the secondary power pin of each critical cell to the center of its TSPS, representing the secondary PDN wirelength in this stage. The $HPWL_{inc}$ can be expressed by the following:

$$HPWL_{inc} = \sum_{c_i \in C^{cri}} (incX_i + incY_i), \tag{3}$$

$$incX_i = \sum_{n_{ij} \in N_i} \left(\frac{|x_i - \mathrm{M_{xij}}| + |x_i - \mathrm{m_{xij}}| - \mathrm{M_{xij}} + \mathrm{m_{xij}}}{2}\right), \tag{4}$$

$$incY_i = \sum_{n_{ij} \in N_i} \left(\frac{|y_i - \mathrm{M_{yij}}| + |y_i - \mathrm{m_{yij}}| - \mathrm{M_{yij}} + \mathrm{m_{yij}}}{2}\right). \tag{5}$$

Equations (4) and (5) respectively compute the overall HPWL increase in the x- and y-direction of all the nets connected to a cell c_i when moving c_i from its original position to a location candidate at (x_i, y_i). Note that no matter whether c_i is moved to a location inside or outside the original net bounding box, the increase in HPWL can be accurately computed using the equations. Next, the total straight-line distance of all secondary power pins to their TSPSs Z can be computed by the following equation:

$$Z = \sum_{c_i \in C^{cri}} \zeta_i, \tag{6}$$

and ζ_i can be expressed as follows:

$$\zeta_i = b_i^t \times d_i^t + b_i^b \times d_i^b + b_i^l \times d_i^l + b_i^r \times d_i^r, \tag{7}$$

$$b_i^t + b_i^b + b_i^l + b_i^r = 1. \tag{8}$$

Equation (7) is used to compute distance from c_i to its TSPS. Only one of b_i^t, b_i^b, b_i^l, and b_i^r will be 1, indicating that this power stripe is the TSPS for c_i, while the others will be 0. Therefore, we constrain this with Equation (8). It is important to note that, since this objective function is meant for use by an ILP solver, variable multiplications are not allowed. Therefore, we need to reformulate the above mathematical expression in an equivalent manner.

The primary task in the following is to express $d \cdot b = \zeta$ in a way that does not involve the multiplications of variables, and then to apply this mathematical expression to replace Equation (7). We try to express $d \cdot b = \zeta$ as inequalities, where $d, \zeta \in \mathbb{Z}$ and b is a binary variable, ζ represents the distance between the cell and the power stripe. This implies that when $b = 0$, ζ must equal 0, and when $b = 1$, ζ must equal d. We can utilize the big-M method to resolve this issue, and the transformed inequality is as follows:

$$\zeta \leq d, \tag{9}$$

$$\zeta \leq \mathrm{M} \times b, \tag{10}$$

$$\zeta \geq 0, \tag{11}$$

$$\zeta \geq d - \mathrm{M}(1-b), \tag{12}$$

where M is a large constant much greater than ζ and d, and $M \in \mathbb{Z}$. The above equations guarantee that when $b = 0$, ζ can be constrained to 0 by Equation (10) and Equation (11). Conversely, when $b = 1$, ζ can be constrained to d by Equation (9) and Equation (12). Given that our objective function seeks to minimize the value, it is unnecessary to impose upper bounds from Equation (9) and Equation (10); it suffices to restrict the lower bounds using Equation (11) and Equation (12) only. Based on the above derivations, we can incorporate this concept into Equation (7) to obtain the following inequalities that satisfy ILP requirements:

$$\zeta_i \geq |p_i^t - y_i^{sec}| - \mathrm{M}(1-b_i^t), \tag{13}$$

$$\zeta_i \geq |y_i^{sec} - p_i^b| - \mathrm{M}(1-b_i^b), \tag{14}$$

$$\zeta_i \geq |x_i^{sec} - p_i^l| - \mathrm{M}(1-b_i^l), \tag{15}$$

$$\zeta_i \geq |p_i^r - x_i^{sec}| - \mathrm{M}(1-b_i^r), \tag{16}$$

$$\zeta_i \geq 0. \tag{17}$$

The distance d between the cell and the power stripe will vary based on their relative position. Specifically, if the power stripe is located above, below, left, or right of the cell, the distance can be calculated as $|p_i^t - y_i^{sec}|$, $|y_i^{sec} - p_i^b|$, $|x_i^{sec} - p_i^l|$, or $|p_i^r - x_i^{sec}|$, respectively. Only one of the right-hand sides of Equations (13)–(16) will be greater than zero, while the others will be greater than a negative value, making them negligible. Equation (17) can be omitted by defining ζ_i to be no less than 0 in the implementation.

3) ILP Constrain Function: In the following, we consider the fundamental row-based design rule constraints to ensure that cells are placed reasonably within the available white spaces. Specifically, cells must not overlap, must be placed within rows, and must be aligned with sites. The first set of constraints consists of five equations, primarily used for matching critical cells to available white spaces:

$$\sum_{\forall s_k \in W} e_{i,k} = 1, \qquad \forall c_i \in C^{cri}, \tag{18}$$

$$\sum_{\forall s_k \in S} (e_{i,k} \times \mathrm{row}_k^s) = row_i^c, \qquad \forall c_i \in C^{cri}, \tag{19}$$

$$\sum_{\forall s_k \in S} (e_{i,k} \times \mathrm{minX}_k^s) \leq x_i, \qquad \forall c_i \in C^{cri}, \tag{20}$$

$$\sum_{\forall s_k \in S} (e_{i,k} \times \mathrm{maxX}_k^s) \geq x_i + w_i^c, \qquad \forall c_i \in C^{cri}, \tag{21}$$

$$\sum_{\forall c_i \in C^{cri}} (e_{i,k} \times \mathrm{w}_i^c) \leq w_k^s, \qquad \forall s_k \in S. \tag{22}$$

If $e_{i,k}$ equals 1, it indicates that c_i is placed in s_k; otherwise, it indicates that c_i is placed in another white space. Equation (18) states that a cell must and can only be placed in one white space, and thus the sum of variables $e_{i,k}$ equals 1. Equation (19) ensures that c_i is placed in the same row as its target white space. In this equation, row_i^c is a variable, while row_k^s is a constant, ensuring that row_i^c is set to row_k^s. Equation (20) and Equation (21) are used to place c_i within the horizontal span of s_k. minX_j^w, maxX_j^w, w_i^c, and w_k^s are given constants. The constraints above ensure that cells are accurately matched to their white spaces, placed in the same row, and positioned within the white space. However, if a white space is large enough, it may be able to accommodate more than one cell. Therefore, we use Equation (22) to limit the total width of cells that can be placed within a white space, ensuring that it does not exceed the width of the white space itself, preventing each white space from overflowing. This ensures that all cells paired with s_k can fit within it without exceeding its capacity.

The second set of constraints primarily aims to prevent overlaps between cells, as formulated below:

$$|row_i^c - row_j^c| \leq \mathrm{M} \cdot d_{i,j}, \qquad \forall c_i, c_j \in C^{cri}, \tag{23}$$

$$|row_i^c - row_j^c| \geq d_{i,j}, \qquad \forall c_i, c_j \in C^{cri}, \tag{24}$$

$$x_i + w_i^c \leq x_j + \mathrm{M} \cdot (1 - g_{i,j}) + \mathrm{M} \cdot d_{i,j}, \qquad \forall c_i, c_j \in C^{cri}, \tag{25}$$

$$x_j + w_j^c \leq x_i + \mathrm{M} \cdot g_{i,j} + \mathrm{M} \cdot d_{i,j}, \qquad \forall c_i, c_j \in C^{cri}. \tag{26}$$

Equation (23) and (24) are primarily used to determine whether the assigned rows of c_i and c_j are the same. If they are the same, the boolean variable $d_{i,j}$ will equal 0; otherwise, it will equal 1. In Equation (23), when $row_i^c \neq row_j^c$, $|row_i^c - row_j^c|$ will be greater than 0, meaning that $d_{i,j}$ must be 1 for the inequality to hold. Conversely, when $row_i^c = row_j^c$, $d_{i,j}$ can be either 0 or 1, hence Equation (24) ensures that $d_{i,j}$ must be 0. Equation (25) and Equation (26) ensure that c_i and c_j do not overlap when they are in the same row. Equation (25) applies when c_i is to the left of c_j, while Equation (26) applies in the opposite situation. The boolean variable g_{ij} is used to distinguish between these two conditions. For instance, if $g_{ij} = 0$, Equation (26) simplifies to:

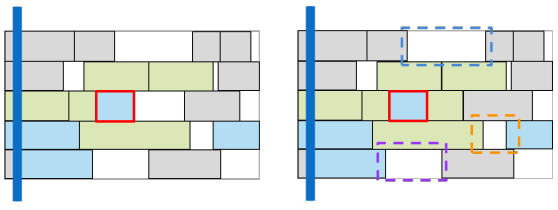
$$x_i + \mathrm{w_i^c} \leq x_j + M. \tag{27}$$

Since M is a constant much larger than all other numbers, Equation (25) becomes ineffective in restricting any variable. The term $M \cdot d_{ij}$ in both equations ensures that Equations (25) and (26) only impose constraints when $d = 0$, meaning the cells are in the same row. Otherwise, the expressions will always hold.

By combining the constraint functions with the objective functions, the ILP formulation can theoretically find the optimal solution to the problem. This ensures that critical cells stay as close as possible to the TSPS, without significantly increasing the signal wire length, while meeting all design constraints.

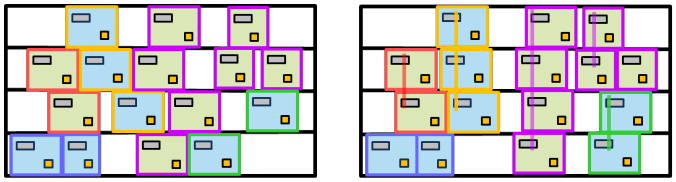
C. Trapped Cell Relocation

The ILP formulation performs replacement such that cross-domain cells are relocated to be close to their TSPSs. To perform secondary power routing, the nearby cross-domain cells belonging to the same SPD are usually connected to each other, forming a local routing tree and then connecting to their common TSPS. This routing nature reduces wire length and conserves routing resources, but if a cell of another domain is surrounded by these cells, it may be obstructed and suffer from routing difficulty, which will be labeled as a "trapped cell". To avoid an excessively conservative mechanism, a cell can only be marked as trapped if there is no available space larger than its width in its vicinity. If there is a space larger than the cell width nearby. As shown in Fig. 6, the cell highlighted by red in Fig. 6(a) is not classified as trapped, and that in Fig. 6(b) is trapped.



(a) Non-trapped cell (b) Trapped cell and its candidate white space

Fig. 6. Examples of trapped and non-trapped cells



(a) (b)

Fig. 7. Examples of pin alignment and clustering. (a) The cell clustering result before performing pin alignment, divided into four groups: blue, red, orange, and purple. (b) The cells after performing pin alignment, with line segments representing the planned routing paths

Trapped cells often struggle with accessing secondary power nets, resulting in limited routing paths and potential connection violations. To address this, we propose an algorithm for relocating these cells. The algorithm employs a window-based approach to identify suitable spaces, starting with a square window equal to five rows in height. If no suitable space is found, the window expands until an appropriate location is identified. the found candidates are prioritized as follows: a space adjacent to a cell of the same SPD and is wider than the trapped cell (as the purple box in Fig. 6(b)) has the highest priority. The second choice is a space adjacent to a cell of the same SPD but narrower than the trapped cell (as the orange box), and the displacement of other cells to be shifted to accommodate the trapped cell should be smaller than a user-defined threshold. Finally, we consider a space that has no adjacent cells from the same SPD and is wider than the trapped cell (as the blue box).

D. Secondary Power Pin Alignment

The final stage involves a placement algorithm applied to all cross-domain cells collectively. After relocating the trapped cells, we aim to align the secondary power pins of all cross-domain cells belonging to the same SPD to a certain degree. This secondary power pin alignment step can contribute to routing bend and complexity minimization during secondary power routing.

The alignment process consists of two phases. The first phase involves cell clustering, as shown in Fig. 7(a), and the second phase focuses on aligning pins within each cluster, with the result shown in Fig. 7(b). Each cluster can contain up to ten cells. To group cells, we start with the bottom-left cell and look for other cells within the same SPD, prioritizing those closest to the secondary power pin. The cell closest to this pin is added to the cluster. After each addition, the centroid of the secondary power pins within the cluster is updated, and the search continues for the next nearest secondary power pin relative to the centroid. This process repeats until no suitable candidates remain or the cluster reaches its maximum size. It is crucial to note that this search has spatial restrictions. If there is any cell from a different SPD between a candidate cell and the cluster, or if the distance exceeds

TABLE II
EXPERIMENTAL RESULT OF WIRE LENGTH AND VIOLATIONS

Case2							Case3						
Situation	Dis	CriSecWL	SecWL	SignalWL	#Vios	CPU (s)	Situation	Dis	CriSecWL	SecWL	SignalWL	#Vios	CPU (s)
Original		310.66	15598	1324738	1340	0	Original		294.53	13817	1493950	1324	0
Only ILP	10	269.65	11604	1360978	591	2800	Only ILP	10	256.24	10059	1539067	605	2369
All techniques		270.31	11317	1373733	468	2803	All techniques		248.16	9974	1538907	498	2375
Original		406.86	16724	1399103	1338	0	Original		430.43	19764	1493825	1629	0
Only ILP	20	408.6	15409	1399640	671	2533	Only ILP	20	344.64	14035	1567163	675	2224
All techniques		409.69	15704	1405561	531	2536	All techniques		344.68	13827	1568370	524	2229
Original		599.81	28554	1324735	1937	0	Original		612.87	25627	1493803	1845	0
Only ILP	30	509.16	19564	1425908	806	1542	Only ILP	30	532.15	19684	1625694	763	1253
All techniques		515.23	20246	1441374	681	1547	All techniques		529.95	17487	1608271	632	1255
Original		789.94	40284	1326071	2105	0	Original		769.82	33112	1494328	2256	0
Only ILP	40	729.64	26487	1490347	768	1044	Only ILP	40	709.67	22654	1635195	703	1013
All techniques		732.96	26682	1461772	735	1047	All techniques		704.49	21867	1626657	698	1017
Average							**Ratio**						
Original		526.87	24185	1418819	1722	0	Original		1	1	1	1	-
Only ILP		469.96	17437	1505499	698	1847.25	Only ILP		0.89	0.72	1.06	0.4	1
All techniques		469.43	17138	1503080	596	1851.12	All techniques		0.89	0.7	1.05	0.34	1

one row vertically or two rows horizontally, the candidate will not be merged.

Once the clusters are determined, the secondary pin alignment algorithm is applied to each cluster, focusing on aligning the secondary power pins along the horizontal axis. The number of centroids used for alignment is based on the maximum number of cells in the same row within a cluster. For example, in the purple cluster in Fig. 7(b), the cells are divided into three groups, each aligning to a different centroid. The goal of this alignment is to ensure that the secondary power pins overlap near the center by at least half the width of a secondary power pin, making routing easier.

During the alignment process, if a cell movement is obstructed by adjacent cells, specific rules are followed. When only one cell blocks the movement, the adjustment is made but reduced to half the intended distance. However, if multiple cells block the movement, no adjustment is made for the cells in that row. After completing the alignment for all clusters, the centroids are recalculated, and the process is repeated. In this second alignment phase, any displacement of adjacent cells is again reduced by half. If the resulting displacement falls below a certain threshold, further movement is stopped.

IV. EXPERIMENTAL RESULTS

Our algorithm for the placement stage was implemented in C++ programming language. The routing stage is performed using the commercial tool INNOVUS [2] for routing and DRC verification. The complete framework was executed on a Linux machine equipped with a 3.5 GHz CPU and 72 GB RAM, running CentOS. We use IBM's CPLEX [3] as the ILP solver. To verify the effectiveness of the proposed methodologies, we utilize the benchmarks of the ISPD 2018 Contest on Initial Detailed Routing [1], which provides a legally placed design, including cell placement, circuit netlist, and technology constraints. Each design consists of 9 metal layers: the primary power stripes are placed on M9 and M8, and the secondary power stripes on M7 and M6. To meet advanced process requirements, layers M1 to M3 are set as 1D routing layers, while the remaining layers are available for 2D routing.

Regarding pin patterns, the original benchmark lacks secondary power pins, so we modify pin patterns to generate cross-domain cells. In our design, the cells within a window of 10×10 row heights are designated as cross-domain cells for secondary power routing, comprising approximately 2% to 12% of all cells. Each design includes two secondary PDNs. The secondary power routing follows a trunk-based pattern with a fanout limit of 10. After routing the benchmark circuits (case1 through case10) in the routing stage, we observed that case2 and case3 experienced significant numbers of additional violations due to secondary power routing. Consequently, these two cases are used as primary targets for presenting experimental results, as they feature higher pin densities and a greater proportion of cross-domain cells compared to other designs.

Empirically, the distances between cross-domain cells and their TSPSs can significantly impact overall design quality. In designs with larger secondary power stripe spacings, moving critical secondary cells closer to their target stripe becomes a priority. Conversely, in designs with smaller stripe spacings, the focus shifts to fine-tuning pin alignment. To demonstrate the effectiveness of our algorithm across various secondary power stripe spacings, we conduct experiments with different stripe distances as a variable. In addition, we limit each ILP iteration to 100 seconds, and the parameter α in Equation (2) is set to 0.99, prioritizing aggressive reduction of secondary PDN wire length over preservation of signal wirelength.

The experimental results are shown in Table II. In the table, "Dis" refers to the spacing between secondary power stripes, measured by the number of rows in between. "CriSecWL" is the critical secondary PDN wire length, calculated by summing the longest 10 secondary power nets to provide stable data without large fluctuations. "SecWL" is the total wire length of the secondary PDNs, and "SignalWL" is the total wire length of signal nets. "#Vios" counts the number of design rule violations, and "CPU" represents the total runtime during the placement stage. In addition, we compare three scenarios: without using the placement algorithm ("Original"), using only the ILP formulation for placement refinement ("Only ILP"), and using the complete placement algorithm ("All techniques"). This comparison tests the effectiveness of the trapped cell relocation and pin alignment components through an ablation study.

The experimental results show that the ILP-based placement refinement approach averagely reduces the critical secondary PDN wire length by 11% and the total secondary PDN wire length by 28%, with only a 6% increase in signal wire length. Most importantly, the replacement significantly reduces the violations by 60% on average. Furthermore, incorporating trapped cell relocation and pin alignment into the optimization flow can further achieve 2% and 1% reductions in the PDN wire length and the signal wire length, and a 6% reduction in the number of violations, with minimal impact on both critical secondary wire length and signal wire length, further enhancing the overall algorithm performance.

V. CONCLUSIONS

This paper presents a secondary-power-routing-aware detailed placement algorithm that facilitates the repositioning of cross-domain cells, aiding multiple power domain designs in executing secondary power routing smoothly. The algorithm significantly reduces the secondary PDN wire length and mitigates open circuit issues caused by routing failures. As a result, the total secondary PDN wire length decreased by 30%, the critical secondary PDN wire length was reduced by 11%, and the average number of violations decreased by 66%.

REFERENCES

[1] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "ISPD 2018 initial detailed routing contest and benchmarks," *Proc. International Symposium on Physical Design*, 2018.

[2] Cadence Innovus Implementation System, version 20.10. www.cadence.com

[3] IBM, "CPLEX Optimizer 20.1," https://www.ibm.com/products/ilog-cplex-optimization-studio/cplex-optimizer

[4] C.-P. Lu and I.H.-R. Jiang, "Fast low power rule checking for multiple power domain design," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1745-1750, 2017.

[5] C.-P. Lu and I.H.-R Jiang, "COSAT: congestion, obstacle, and slew aware tree construction for multiple power domain design," *In Proceedings of Design Automation Conference (DAC)*, no. 47, pp. 1–6, 2018.

[6] C.-P. Lu, I.H.-R Jiang and C.-H. Hsu, "GasStation: Power and area efficient buffering for multiple power domain design," *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015

[7] K.-W. Chiu, Y.-G. Chen, and C. Lin, "An efficient NBTI-aware wake-up strategy for power-gated designs," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018.

[8] Y.-T. Shyu, J.-M. Lin, C.-C. Lin, C.-P. Huang, and S.-J. Chang, "An efficient and effective methodology to control turn-on sequence of power switches for power gating designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1730–1743, 2016.

[9] X.-D.S. Tan, and C-J.R. Shi, "Fast power/ground network optimization based on equivalent circuit modeling," *Proceedings of the 38th annual Design Automation Conference.* 2001.

[10] C.-J. Lee, S.S.Y. Liu, C.-C. Huang, H.-M. Chen, C.-T. Lin and C.-H. Lee, "Hierarchical power network synthesis for multiple power domain designs," *Thirteenth International Symposium on Quality Electronic Design (ISQED).* IEEE, 2012.

[11] M. Popovich, E.G. Friedman, M. Sotman and ,A. Kolodny, "On-chip power distribution grids with multiple supply voltages for high performance integrated circuits," *Proceedings of the 15th ACM Great Lakes symposium on VLSI.* 2005.

[12] J.-M. Lin, C.-Y. Huang and J.-Y. Yang, "Co-synthesis of floorplanning and powerplanning in 3D ICs for multiple supply voltage designs," *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2018.

[13] C. Li, Y. Dong and T. Watanabe, "New power-efficient FPGA design combining with region-constrained placement and multiple power domains.," *2011 IEEE 9th International New Circuits and systems conference.* IEEE, 2011.

[14] C.-P. Lu, I.H.-R. Jiang, C.-C. Peng, Mohd M.-M. Razha and A. Uber, "Power Sub-Mesh Construction in Multiple Power Domain Design with IR Drop and Routability Optimization," *Proceedings of the 2024 International Symposium on Physical Design.* 2024.

Generative Model Based Standard Cell Timing Library Characterization

Hao-Yu Wu[1], Hsin-Tzu Chang[1], Shiuan-Yun Ding[1], Iris Hui-Ru Jiang[1]
Benson Tsao[2], Vinson Wu[2], Wei-Kai Shih[2]
[1]*Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan*
[2]*Synopsys, Inc., Hsinchu, Taiwan*
email: {kobe7682477, sheena9101019, mirkat.ding, huiru.jiang}@gmail.com; {ytsao, vinsonwu, Wei-Kai.Shih}@synopsys.com

***Abstract*—Accurate cell timing characterization is essential, on which static timing analysis relies to verify timing performance and ensure design robustness across various PVT conditions (corners). The corner explosion in modern design amplifies the efficiency and scalability challenge for accurate characterization. However, the conventional characterization approach of SPICE simulation alone becomes prohibitively expensive due to the increasing computational complexity and the amount of characterized data. In this paper, we view the characterization problem from a generative modeling perspective to tackle the efficiency and scalability challenge. With a hybrid of generative adversarial network (GAN) and autoencoder, our generative model learns and generalizes among various timing arcs and corners. Experimental results demonstrate that the proposed framework achieves high accuracy and extensibility while reducing the runtime significantly.**

***Index Terms*—Library Characterization, Corner Explosion, Machine Learning, Generative Model, Corner Selection**

I. Introduction

Standard cell timing library characterization involves analyzing the timing behavior of each standard cell and recording it in the form of look-up tables (LUTs) indexed by input transition and output capacitance, as shown in Table I. Cell timing library provides key timing data, such as cell propagation delays and transition times, etc. The characterized data are utilized by Static Timing Analysis (STA) to evaluate timing paths and ensure design functions properly across different process, voltage, and temperature (PVT) conditions, so-called corners. As fine-grained PVT resolution becomes increasingly important for analyzing design robustness, the need to characterize more corners has grown, adding complexity to the characterization process.

SPICE simulation is typically used to characterize timing libraries, as it provides accurate timing values through extensive computations. However, it faces two main issues. First, its high computational complexity results in long runtime, creating challenges for efficiency and scalability, particularly concerning corner explosion [1]–[3], which poses a significant risk to the industry. Second, SPICE simulation requires input files and configurations from the foundry, as illustrated in Fig. 1. If the foundry needs to maintain data confidentiality, access to the necessary locked data marked in Fig. 1 becomes unavailable. This loss of access causes considerable inconvenience. Therefore, it is crucial to develop a fast characterization approach that does not depend on locked input data.

This work was supported in part by Synopsys, Inc. and the National Science and Technology Council, Taiwan.

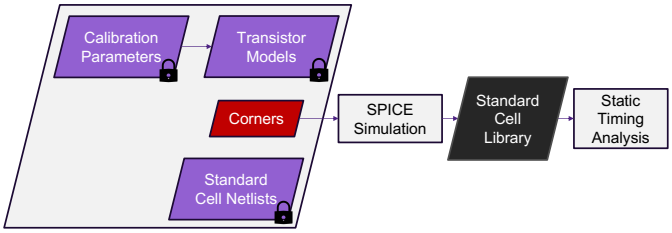


Fig. 1: Conventional library characterization.

Previous research has explored various aspects of library generation. For example, cell layout generation techniques such as the use of reinforcement learning (RL) for design rule checking (DRC) [4] and the automatic cell layout generation (ACLG) framework proposed by [5], primarily focus on optimizing placement and routing processes. Additionally, [6] presents the first drain-to-drain abutment (DDA)-aware placement algorithm. While these contributions are significant, they highlight a broader need for improved methodologies in timing library characterization.

In the realm of timing library characterization, graph learning has emerged as a prominent technique by representing standard cell netlists as graphs. This approach is particularly valuable when graphical information about the cell netlist is given. For example, [7] leverages a Graph Neural Network (GNN) to capture cell structure information and interpolate fine-grained driving strength into the library. In contrast, [8] employs heterogeneous graph learning to account for parasitic RC. However, these graph learning methods require access to cell structure information, which may be confidential in certain situations as marked in Fig. 1.

Machine learning (ML) techniques have also been utilized specifically for timing library characterization, focusing on critical cell features to enhance accuracy. For instance, [9] focuses on variation modeling using a Feedforward Neural Network (FFN). It highlights the trade-off between the number of SPICE-simulated corners required and prediction accuracy, which is an important consideration when evaluating library characterization methods. Additionally, [10] proposes Aadam, which integrates their FFN models into the STA engine for aging library characterization. ML-based methods have proven effective for library characterization. However, most models

TABLE I: A sample cell timing look-up table. Red numbers indicate the pivot value associated with the pivot input transition (ns) and pivot output capacitance (ff).

		Output Capacitance						
		0.15	0.34	1.02	2.31	4.31	7.13	10.83
Input Transition	0.003	0.01199	0.01374	0.01951	0.03052	0.04761	0.07138	0.10250
	0.005	0.01350	0.01527	0.02106	0.03207	0.04916	0.07292	0.10410
	0.014	0.01898	0.02073	0.02651	0.03745	0.05439	0.07805	0.10940
	0.029	0.02834	0.03015	0.03602	0.04698	0.06395	0.08783	0.11920
	0.054	0.03960	0.04161	0.04764	0.05858	0.07551	0.09936	0.13080
	0.088	0.05274	0.05492	0.06095	0.07203	0.08901	0.11270	0.14410
	0.133	0.06691	0.06935	0.07601	0.08701	0.10390	0.12770	0.15890

proposed in previous work can only handle specific cells or timing arcs, which necessitates the development of numerous models to fully characterize the library. On the other hand, FFN models often struggle with over-fitting, particularly when training data is limited.

The challenges mentioned above motivate us to view the timing characterization problem from a new angle to enhance runtime, accuracy, and generality. We can imagine that each timing look-up table is a point lying on a high-dimensional space. By traversing on the space along the direction of corner changing, we can reveal its counterparts under different corners. Therefore, from a generative modeling perspective, we choose a generative adversarial network (GAN) as the backbone. For generating accurate timing tables, we further integrate an autoencoder to remove the randomness from GAN. Unlike FFN models, which rely solely on loss backtracking, generative models seek additional insights from training data for better prediction performance.

To the best of our knowledge, we are the first to design a generative model for cell timing library characterization. Our key contributions are as follows:

- We introduce a generative model that enables fast and accurate timing library generation without requiring confidential data from the foundry.
- Our generative model generalizes the characterization of timing values across various cells and timing arcs, eliminating the need for separate models for each individual cell or timing arc.
- Our framework effectively reduces the volume of SPICE-simulated data and overall runtime and possesses extensibility for tackling corner explosion.

The remainder of this paper is organized as follows: Section II introduces the terminology and problem formulation. Section III details our generative prediction model. Section IV demonstrates experimental results. Finally, section V concludes this work.

II. Preliminaries

A. Terminology

We define four types of corners, two types of cells, and the pivot value in this work. (see Fig. 2.)

Anchor corner in which full information including the PVT condition, input transition indices, output capacitance indices, and timing values in each timing table is provided. In other words, all timing values in anchor corners are characterized by SPICE simulation. The timing tables in anchor corners serve as the training set.

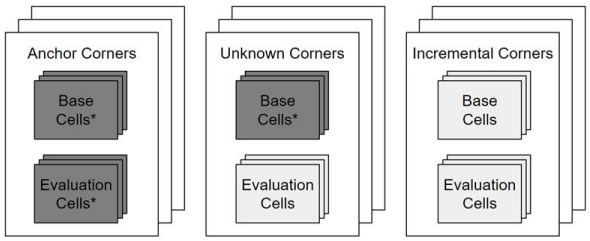


Fig. 2: Base and evaluation cells and the three groups of corners: Cells with a star suffix indicate that their timing values are characterized by SPICE simulation.

Unknown corner in which some of the library cells are characterized by SPICE simulation, and the other cells are characterized by a prediction model. The timing tables in unknown corners serve as the validation set.

Incremental corner in which all library cells are to be characterized by a prediction model. The timing tables in incremental corners are the test set.

Query corner is a corner queried by a user and to be characterized by a prediction model. The corresponding pivot value and timing table indices of a target timing table are encoded as an input to the prediction model. To construct the timing tables for incremental corners, we set each incremental corner as the query corner and use the model to infer and generate its timing tables.

Base cells are library cells which are characterized by SPICE simulation in the unknown corners.

Evaluation cells are library cells which are characterized by a prediction model in the unknown corners.

As shown in Fig. 2, base cells and evaluation cells are characterized by SPICE simulation in anchor corners, but characterized by a prediction model in incremental corners.

Pivot value represents the reference value of a timing table. In our framework, the pivot value is defined as the median value of the table. Taking Table I as an example, the pivot value is 0.04698, highlighted in red.

B. Problem Formulation

As mentioned earlier, accurate and fast cell timing characterization is essential for ensuring design robustness under corner explosion. We attempt to generate incremental cell timing libraries based on SPICE-simulated anchor libraries and corner features. Thus, the problem addressed in this work can be formulated as follows:

The Timing Table Characterization Problem: Given a set of anchor corners with all timing tables of each cell, and a query corner with its PVT configuration, and the table indices and the pivot value of a target timing table, our goal is to characterize the remaining values of the target timing table.

Our goal is to enhance prediction accuracy and reduce the SPICE simulation time. The prediction model is desired to generalize among cells and timing arcs. The timing tables considered in this work include but are not limited to cell rise/fall (propagation) delays and rise/fall transition times.

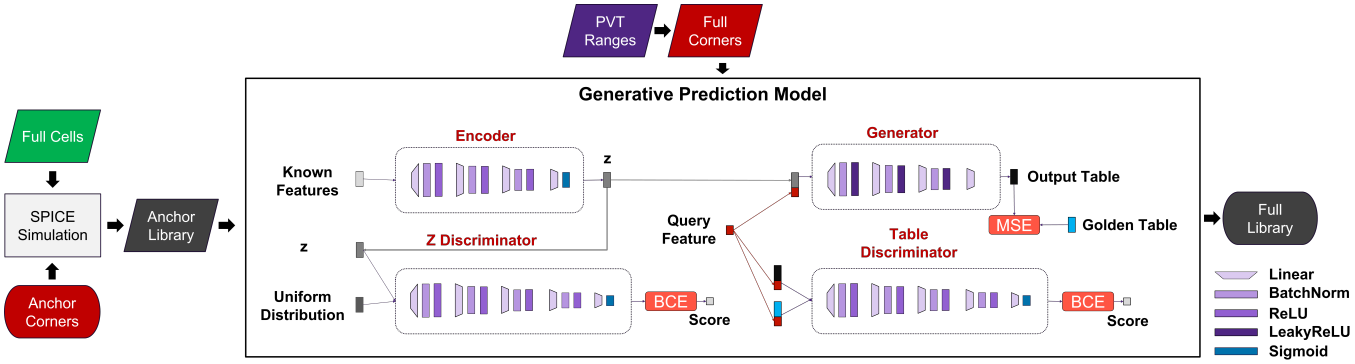


Fig. 3: The overall characterization flow.

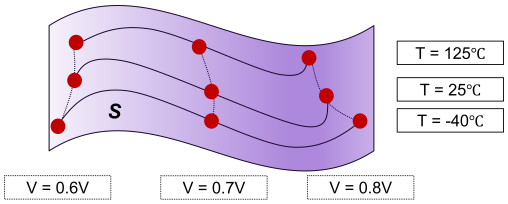


Fig. 4: Cell timing libraries lie on a high-dimensional space S.

III. Generative Prediction Model

A. Overview

Fig. 3 demonstrates the overall flow and framework of our cell timing library characterization. Our generative prediction model produces the timing tables for a cell under a query corner, given anchor corners characterized by SPICE simulation. An incremental library, i.e., a cell timing library under an incremental corner, can be generated by querying the full set of cells for such corner.

We can imagine that each timing table is a point lying on a high-dimensional space S as shown in Fig. 4. Exploring the space in the direction of corner changing allows us to uncover its counterparts under various corner settings. We construct a conditional adversarial autoencoder (CAAE) for cell timing library characterization, which uses the pivot value of a cell timing table in the query corner as a conditional input and concatenates it with extracted cell characteristics to generate the whole target timing table.[1]

The complete structure of CAAE is shown in Fig. 3. We input the query feature (the pivot point of a cell under the query corner) and known features (extracted from the timing tables at anchor corners), and CAAE will output the whole target timing table. There are four components in CAAE, an encoder, a generator, and two discriminators. The generator and the table discriminator constitute a commonly used GAN structure. The input features of the generator are usually random distributions, which bring a degree of randomness to the output of the generator. Therefore, we use an encoder to map given cell features into a smooth distribution, then the generator takes the distribution as the input. The randomness no longer exists. The combination of the encoder and generator can also be regarded as an autoencoder. It can be seen that CAAE is a hybrid of GAN and autoencoder. The two discriminators are the z discriminator and table discriminator. The z discriminator is used to guide the encoder to output a uniform distribution. The uniform distribution can help the training process converge faster. The table discriminator is used to grade the output of the generator and to provide feedback for the generator training.

[1]This formulation is analogous to facial age progression and regression, which generates incremental human face images given personal characteristic and age. The CAAE structure is also applied to facial age progression/regression in [11] with a promising image quality. The similarity between the two problems makes CAAE appear suitable for application in timing library characterization.

B. Features

The features fed into the generative prediction model are divided into known features and query features. These features are represented as X_{known} and X_{query}, respectively:

X_{known} = [process, voltage, temperature, output capacitance list, pivot input transition, pivot output capacitance, pivot value, timing values]

X_{query} = [process, voltage, temperature, output capacitance list, pivot input transition, pivot output capacitance, pivot value]

The descriptions for each feature are as follows:

- **Process (encoded as 0, 1, 2)**: Represents the process type, where 0 corresponds to ff, 1 to tt, and 2 to ss.
- **Voltage**: The operating voltage level.
- **Temperature**: The operating temperature.
- **Output Capacitance List**: Represents different output capacitance values, indexed from the timing table.
- **Pivot Input Transition**: The input transition value corresponding to the pivot point.
- **Pivot Output Capacitance**: The output capacitance value corresponding to the pivot point.
- **Pivot Value**: The median value in a timing table, located at the $\lfloor M/2 \rfloor$-th row and $\lfloor N/2 \rfloor$-th column. This value is characterized through SPICE simulation and used to normalize all values in the timing table. Including this value in the features is crucial for ML model training.
- **Timing Values**: All timing values from the timing table. Each table of size $M \times N$ is flattened into a one-dimensional vector, which is then concatenated with other features.

The known feature vector X_{known} includes the complete feature set for the anchor corners, encapsulating all necessary

information from the timing tables. The query feature vector X_{query} represents the PVT configuration for the query corner, along with features related to the pivot value.

C. Input / Output Specifications

Here we specify the input and the output of the encoder, generator, z discriminator, and table discriminator. Let c denote the cell to be characterized by our generative prediction model, z the output of the encoder, and q the query corner.

1) Encoder: We first prepare all known features for c in all anchor corners, i.e., if the number of anchor corners is k, we have k known features from X^1_{known} to X^k_{known}. All known features share the same input transition list because input transition lists are all the same among our benchmark. We concatenate the input transition list and k known features as the input of the encoder. The features can be easily adjusted for variant input transition lists. The output of the encoder is a vector of length 32, and it is expected to follow a uniform distribution.

2) Generator: The input of the generator is the concatenation of z and X_{query} of c and q. The output of the generator is a vector of length $M \times N$, which is resized to an $M \times N$ table.

3) Z Discriminator: The input of the z discriminator is solely z. The output is a single score. The score increases as the input becomes closer to a uniform distribution.

4) Table Discriminator: The input of the table discriminator is the concatenation of a vector of length $M \times N$ and X_{query}. The output is a single score. The score increases as the input is closer to a SPICE-simulated timing table.

D. Model Structures

The internal structures of four components are shown in Fig. 3. They share similar structures. There is a stack of linear layers, batch normalization layers, and activation functions. The difference lies in the selection of the activation function and the Sigmoid function at the end of the encoder, z discriminator, and table discriminator.

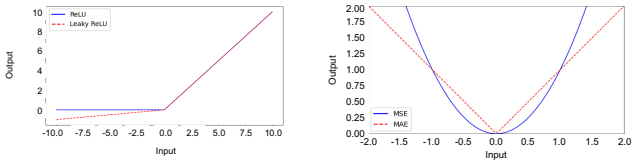
A comparison between ReLU function and LeakyReLU function is shown in Fig. 5a. LeakyReLU function retains information when the input is negative, while ReLU function ignores it. We apply LeakyReLU function on the generator so that the generator will not ignore any information from the already-extracted features. The Sigmoid function is used to confine the output. We expect the output values of the encoder and the two discriminators to range from 0 to 1.

E. Loss Functions

Let n be the length of the predicted and the golden vector, y_i be the $i-th$ element in the golden vector, and $\hat{y_i}$ be the $i-th$ element in the predicted vector. We use two distinct loss functions in our work, mean square error (MSE) and binary cross entropy (BCE). MSE is used in both the encoder and generator, while BCE is used in the two discriminators.

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

$$BCE(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n}(\hat{y_i} * \ln y_i + (1 - \hat{y_i}) * \ln(1 - y_i))$$



(a) ReLU and LeakyReLU (b) MSE and MAE

Fig. 5: Activation and loss functions.

MSE is also known as L2 loss. MSE is the most common loss function for evaluating the distance between the values. MSE tends to provide a more stable result compared to mean absolute error (MAE). MSE is smoother than MAE as shown in Fig. 5b. This is the reason that we apply MSE loss on the comparison of predicted table values and golden table values.

BCE is commonly used in binary classification problems. Discriminators categorize the output into predicted values and golden values. The grading process of the discriminator is also regarded as a binary classification problem. Hence, BCE is adopted for them.

F. Model Training

Due to the competitive nature of the generator and the discriminator, it is challenging to reach Nash equilibrium. The balance between the generator and the discriminator should be carefully maintained. As commonly observed in GANs, we separate the training process into three parts: encoder and generator, z discriminator, and table discriminator.

The encoder and generator are guided not only by the two discriminators but also by the MSE loss function. We measure the distance between the discriminator outputs and the value of 1 through BCE. This makes sense because we want the encoder and generator to deceive the two discriminators, implying that the output should be close to 1. The usage of MSE is similar to that in the typical case. We directly utilize MSE to compare the output of the generator and the ground truth.

For the training process of the z discriminator, we randomly generate a uniform distribution and label it as 1. In contrast, the output of the encoder is labeled as 0. The discriminator should learn to distinguish between them.

For the training process of the table discriminator, the concatenation of the predicted values and X_{query} is labeled as 0, while that of the golden values and X_{query} is labeled as 1. Likewise, the table discriminator should learn to differentiate between the two.

The most difficult part of the training process is to ensure an appropriate balance between the growth of the encoder, generator and discriminators. We employ two techniques to solve this issue. The first one is to backtrack the discriminators once after the encoder and generator have been backtracked m times. This is a common technique used in the training process of GAN. We ensure that the encoder and generator are not overly influenced by the discriminator in the early stage. The second technique is that during the training process of the encoder and generator, we add weights on aforementioned losses. The two loss weights are then adjusted based on the ratio between three types of loss. If the discriminator loss

TABLE II: Corners in two datasets.

Benchmark	Anchor Corners	Unknown Corners	Incremental Corners
CAD Contest	ff0p88vm40c, ff1p1v125c tt0p9v25c ss0p72vm40c, ss0p9v125c	ff0p99v125c, ff1p1vm40c tt0p8v25c ss0p81v125c, ss0p9vm40c	ff0p88v125c, ff0p99vm40c tt1p0v25c ss0p72v125c, ss0p81vm40c
SAED14nm	ff0p7vm40c, ff0p88v125c tt0p6vm40c, tt0p8v125c ss0p6vm40c, ss0p72v125c	ff0p7v125c, ff0p88vm40c tt0p6v125c, tt0p8vm40c ss0p6v125c, ss0p72vm40c	ff0p7v25c, ff0p88v25c tt0p6v25c, tt0p8v25c ss0p6v25c, ss0p72v25c

is too high, we decrease its weight. It is essential to ensure that the loss from the loss function and the loss from the discriminators are of similar magnitude.

IV. Experimental Results

A. Experimental Settings

Our framework was implemented in the Python 3.9.18 language. The experiments were conducted on a Linux workstation with a 2.1 GHz Intel ®Xeon ®Gold 6230R CPU equipped with 408 GB RAM, and three NVIDIA GeForce RTX 3090 GPUs equipped with 24 GB RAM on each.

Our experiments were conducted on two datasets: The CAD Contest dataset contains 400 cells and 15 corners [12], while SAED14nm is a 14nm FinFET process library, with 835 cells and 18 corners [13]. Table II provides the corresponding corner lists. The corner name with three segments indicates its nMOS and pMOS processes, voltage, and temperature in order. For the process segment, f, t, and s represent fast, typical, and slow, respectively. For instance, $ss0p9vm40c$ corresponds to slow nMOS and pMOS processes, 0.9 V, and -40°C. Moreover, the corners are classified into anchor, unknown, and incremental corners. Given the relative simplicity of interpolation in comparison to extrapolation, the extreme values of voltage and temperature are selected for the anchor corners, encompassing both the highest and lowest observed values.

Cells are divided into two sets: base cells and evaluation cells, with a ratio of 3:7. Each cell has multiple variations with different driving strengths. To avoid excessive duplicates in the train or test set, in our splitting algorithm, cells are grouped based on driving strength into three categories: minimum, median, and maximum. We prioritize selecting base cells from the minimum group first, followed by the maximum group, and then the median group.

B. Results and Comparisons

The overall experimental results of the first experiment are illustrated in Tables III and IV. Linear regression and the FFN proposed in [10] are employed for model comparison. The experimental results are presented in three distinct forms for a comprehensive evaluation of the model performance. The timing values in the standard cell library are directly compared. The three models demonstrate superior performance on transition arcs compared to delay arcs because the relations between indices and values are more straightforward for transition arcs.

Two metrics, MAE and mean absolute percentage error (MAPE), are employed for evaluation. MAPE is the evaluation metric used in 2020 CAD Contest. For SAED14nm, the MAE pass rate is also taken into account [14]. The MAE pass rate threshold is set as 2 ps, which is a commonly accepted industry standard in 14nm process. The pass rates are not shown in 2020 CAD Contest results because of the insufficient background information in the benchmark.

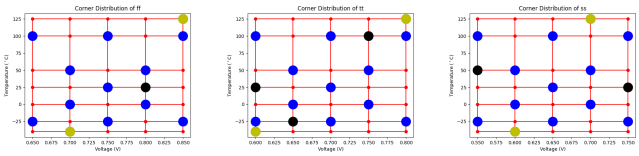


Fig. 6: Anchor corner selection by HDBSCAN. Green points refer to 6 basic anchor corners, black points refer to extra anchor corners, and blue points refer to pre-defined incremental corners.

MAE is the primary metric used to evaluate model performance, as it directly indicates the absolute error, providing a clear assessment of the model.

C. Model Extensibility

The SAED14nm benchmarks used in the first experiment consist of 18 corners, demonstrating our ability to accurately generate timing tables as compared to the linear regression method and FFN model. To address the corner explosion issue, we conduct the second experiment to evaluate the extensibility and scalability of our model.

Assume there are K corners of interest in total, and we would like to select k anchor corners out of them, where $K >> k$. First, we devise an anchor corner selection algorithm to pick k anchor corners for training. The algorithm is based on the hierarchical density-based spatial clustering of applications with noise (HDBSCAN), a clustering algorithm. All of the K corners are separated into multiple clusters by HDBSCAN. Then, for each cluster, a fixed proportion of representative corners are selected to represent the cluster. Finally, we use the selected representative corners for the prediction model training. Moreover, we propose a cell selection algorithm to reduce the number of cells to which the corner selection algorithm applies. With the corner selection algorithm, we can scale the number of anchor corners as the total number of corners increases, maintaining a consistent ratio to guarantee performance.

In this experiment, we consider $K = 105$ corners of SAED14nm libraries and utilize different numbers of anchor corners for training. The process covers ff, tt, and ss; voltage levels are 0.6V, 0.65V, 0.7V, 0.75V, and 0.8V; temperatures are -40°C, -25°C, 0°C, 25°C, 50°C, 100°C, 125°C. The golden results characterized by SPICE simulation are generated by Siliconsmart [15] executed in the most basic mode without any acceleration. To ensure a fair comparison, we select 33 corners as incremental corners. The k anchor corners are selected from the remaining 72 corners, and those not selected as anchor corners are used for validation. Fig. 6 illustrates the selected anchor corners and incremental corners. The experimental results are listed in Table V. We demonstrate the results under 6, 12 and 18 anchor corners. (6 means 6 basic anchor corners, while 12 (18) means adding extra 6 (12) anchor corners by the anchor corner selection algorithm.) The model performance improves as the number of corners increases and saturates as 18 anchor corners.

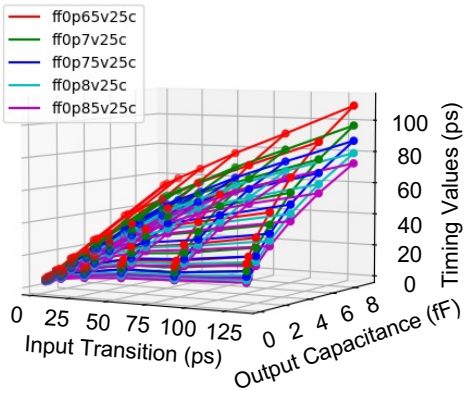


Fig. 7: Cell rise tables of OAI21 at different voltage levels.

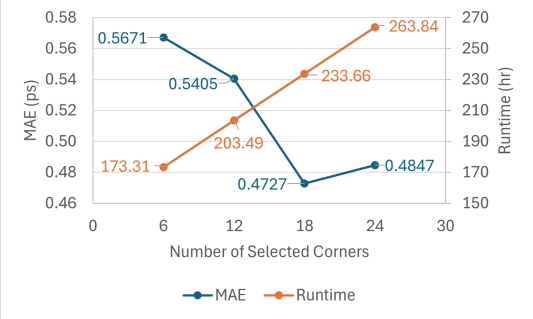


Fig. 8: MAE and runtime with different number of selected anchor corners.

D. Timing Surface Smoothness

As discussed earlier in Section III-A, the generative model effectively captures the high-dimensional space of timing tables across various PVT conditions. To demonstrate smooth traversal and sampling on this high-dimensional surface, we fix the process and temperature while varying the voltage to observe changes in the timing tables. In Fig. 7, we generate cell rise timing tables for the cell OAI21 at different voltage levels, while fixing the process and temperature as ff and $25°C$, respectively. The results show continuous and smooth transitions in the timing tables, confirming that our generative model accurately captures the high-dimensional surface, as illustrated in Fig. 4.

E. Runtime and Accuracy Tradeoff

SPICE simulation done by Siliconsmart to characterize the 105 corner libraries takes 302.83 hours. Siliconsmart operates with 32 CPU cores in this process. Our flow's runtime encompasses the time required for characterizing SPICE-simulated data by Siliconsmart, as well as the time taken for cell and corner selection, model training, and model prediction. The model prediction takes only 133.16 seconds to generate all incremental corner libraries. With our flow, SPICE-simulated data can be reduced to one-third of the original volume. The results demonstrate that our characterization flow exhibits 42.8% and 32.8% runtime reduction for 6 and 12 anchor corners compared with Siliconsmart. Combining the model extensibility evaluation with runtime comparison, Fig. 8 depicts the tradeoff between model accuracy and the number of selected anchor corners.

TABLE III: Model accuracy for CAD Contest dataset.

CAD Contest		MAE (ps)			MAPE		
Timing Arc	Process	Linear Regression	FFN Model	Our CAAE	Linear Regression	FFN Model	Our CAAE
Cell Rise	ff	43.8844	8.1800	1.7389	3.72%	6.12%	0.98%
	tt	55.7203	38.3070	2.2125	4.52%	14.53%	1.18%
	ss	65.2358	232.8860	19.8419	5.25%	16.73%	1.66%
Cell Fall	ff	49.9792	6.0266	1.9085	4.38%	4.06%	1.30%
	tt	65.6919	27.6364	2.4238	5.53%	10.87%	1.57%
	ss	75.4952	155.7370	22.5414	6.14%	12.53%	2.00%
Rise Transition	ff	69.6856	5.3936	2.0656	5.42%	5.25%	1.58%
	tt	35.3363	13.7483	2.5037	2.78%	8.28%	1.90%
	ss	72.9652	322.8612	1.3998	7.10%	26.09%	0.81%
Fall Transition	ff	85.9545	3.8278	2.1772	6.85%	4.48%	2.07%
	tt	50.9714	7.9771	3.2441	3.70%	7.21%	2.50%
	ss	98.5066	237.7742	8.3982	9.25%	25.72%	1.40%
Average		64.1188	88.3629	5.8713	5.39%	11.82%	1.58%
Mixed	ff	69.5385	4.8770	2.0969	5.59%	4.35%	1.53%
	tt	50.1529	13.8812	2.9484	4.04%	8.37%	1.79%
	ss	69.9842	886.0233	10.7462	6.33%	38.96%	1.10%
Average		63.2252	301.5398	5.2638	5.32%	17.23%	1.47%

TABLE IV: Model accuracy for SAED14nm dataset.

SAED14nm		MAE (ps)			MAE Pass rate			MAPE		
Timing Arc	Process	Linear Regression	FFN Model	Our CAAE	Linear Regression	FFN Model	Our CAAE	Linear Regression	FFN Model	Our CAAE
Cell Rise	ff	1.3510	1.0813	0.5675	78.26%	86.78%	94.28%	3.19%	2.72%	1.11%
	tt	1.2867	1.2919	0.5675	80.61%	79.58%	95.38%	2.47%	2.34%	0.87%
	ss	1.2662	1.5482	0.8285	80.07%	73.34%	88.73%	1.91%	2.39%	1.06%
Cell Fall	ff	1.3604	1.6923	0.5164	78.69%	67.82%	96.46%	4.23%	4.04%	1.20%
	tt	1.7517	2.0323	0.5105	69.86%	58.46%	97.15%	3.22%	3.89%	0.90%
	ss	1.4735	2.0080	0.5715	75.36%	62.24%	95.76%	2.13%	3.09%	0.81%
Rise Transition	ff	0.4961	0.8048	0.1653	95.95%	91.65%	99.55%	3.55%	9.26%	0.90%
	tt	0.4569	0.8277	0.1881	95.36%	90.71%	99.41%	2.17%	6.50%	0.80%
	ss	0.6111	1.0076	0.2161	94.77%	86.62%	99.27%	2.69%	7.70%	0.86%
Fall Transition	ff	0.6960	1.1460	0.1198	92.03%	81.69%	99.88%	5.20%	6.14%	0.78%
	tt	0.7404	1.0668	0.1525	89.87%	86.11%	99.83%	4.13%	7.21%	0.74%
	ss	0.7547	1.0530	0.2030	90.98%	85.34%	99.76%	3.60%	7.20%	0.78%
Average		1.0204	1.2967	0.3839	85.08%	79.20%	97.12%	3.21%	5.21%	0.90%
Mixed	ff	1.4305	1.3405	0.4214	77.32%	79.63%	96.45%	6.66%	5.64%	1.15%
	tt	1.5802	1.7580	0.5645	74.64%	68.78%	92.97%	6.32%	5.03%	1.12%
	ss	1.6233	1.6113	0.6904	73.91%	71.15%	90.19%	5.25%	4.23%	1.21%
Average		1.5447	1.5699	0.5588	75.29%	73.19%	93.20%	6.08%	4.97%	1.16%

V. Conclusions

In this work, we propose a cell timing library characterization approach from a generative model perspective. The proposed approach enables designers to generate incremental libraries without the need for confidential data provided by the foundry. The framework not only preserves the adversarial framework of the GAN but also eliminates randomness. A combination of an encoder and a generator is employed to extract salient features from the input, which are then fed into the generator. The model achieves high accuracy with only one-third of the SPICE-simulated data. The model exhibits an exceptional ability to generalize across a wide range of combinational timing arcs. Future work includes extending the approach to constraint timing arcs and aging library characterization.

TABLE V: Model extensibility.

No. of Anchor Corners		MAE (ps)				MAE Pass rate				MAPE			
Timing Arc	Process	6	6+6	6+12	6+18	6	6+6	6+12	6+18	6	6+6	6+12	6+18
Cell Rise	ff	0.4090	0.3338	0.3147	0.3472	98.00%	98.81%	98.91%	98.92%	1.37%	1.12%	1.08%	1.17%
	tt	0.5044	0.4203	0.3985	0.4272	96.94%	97.92%	97.97%	98.07%	1.21%	1.03%	1.00%	1.03%
	ss	0.6892	0.5868	0.5250	0.5573	93.62%	94.92%	96.55%	98.91%	1.19%	1.01%	0.94%	1.00%
Cell Fall	ff	0.4046	0.4453	0.3285	0.3235	97.82%	97.66%	98.57%	98.64%	3.40%	1.72%	1.32%	1.29%
	tt	0.4900	0.5462	0.3977	0.4024	96.81%	96.04%	98.12%	97.88%	1.32%	1.38%	1.05%	1.07%
	ss	0.7766	0.7830	0.5337	0.5788	92.09%	91.97%	96.00%	95.56%	1.38%	1.35%	0.94%	1.04%
Rise Transition	ff	0.4861	0.4361	0.4246	0.4263	97.10%	97.72%	97.75%	97.76%	2.32%	2.08%	2.06%	2.09%
	tt	0.5682	0.5267	0.5272	0.5224	95.68%	96.53%	96.46%	96.56%	2.19%	2.02%	2.03%	2.03%
	ss	0.7620	0.7034	0.6998	0.6933	91.93%	93.10%	93.10%	93.23%	2.29%	2.12%	2.13%	2.09%
Fall Transition	ff	0.4562	0.4711	0.3953	0.3978	96.96%	97.25%	97.89%	97.83%	2.49%	2.63%	2.10%	2.17%
	tt	0.5086	0.5139	0.4701	0.4698	96.47%	96.62%	97.03%	97.10%	2.35%	2.40%	2.06%	2.08%
	ss	0.7410	0.7201	0.6577	0.6705	92.36%	92.81%	93.75%	93.43%	2.46%	2.47%	2.15%	2.17%
Average		0.5671	0.5405	0.4727	0.4847	95.48%	95.95%	96.84%	96.99%	2.00%	1.78%	1.57%	1.60%
Mixed	ff	0.4141	0.3728	0.3546	0.3535	97.64%	98.05%	98.35%	98.48%	1.87%	1.78%	1.63%	1.59%
	tt	0.4901	0.4403	0.4286	0.4249	96.66%	97.38%	97.64%	97.74%	1.65%	1.53%	1.46%	1.44%
	ss	0.7038	0.6020	0.5663	0.5810	93.03%	94.80%	95.38%	95.18%	1.73%	1.55%	1.43%	1.46%
Average		0.5360	0.4717	0.4498	0.4531	95.77%	96.74%	97.13%	97.13%	1.75%	1.62%	1.51%	1.50%

REFERENCES

[1] Taming corner explosion in complex chips. [Online]. Retrieved February 23, 2023, from https://semiengineering.com/taming-corner-explosion-in-complex-chips/.

[2] Andrew B. Kahng, Uday Mallappa, Lawrence Saul, and Shangyuan Tong. "unobserved corner" prediction: Reducing timing analysis effort for faster design convergence in advanced-node design. In *Proceedings of Design, Automation Test in Europe Conference & Exhibition (DATE)*, pages 168–173, 2019.

[3] Iris Hui-Ru Jiang. Lightning talk: All routes to timing closure. In *Proceedings of Design Automation Conference (DAC)*, pages 1–2, 2023.

[4] Haoxing Ren and Matthew Fojtik. Invited-NVCell: Standard cell layout in advanced technology nodes with reinforcement learning. In *Proceedings of ACM/IEEE Design Automation Conference (DAC)*, pages 1291–1294, 2021.

[5] Aaron C.-W. Liang, Charles H.-P. Wen, and Hsuan-Ming Huang. A general and automatic cell layout generation framework with implicit learning on design rules. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, 30(9):1341–1354, 2022.

[6] Yih-Lang Li, Shih-Ting Lin, Shinichi Nishizawa, Hong-Yan Su, Ming-Jie Fong, Oscar Chen, and Hidetoshi Onodera. NCTUcell: A DDA- and delay-aware cell library generator for FinFET structure with implicitly adjustable grid map. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 41(12):5568–5581, 2022.

[7] Tianliang Ma, Zhihui Deng, Xuguang Sun, and Leilai Shao. Fast cell library characterization for design technology co-optimization based on graph neural networks. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 472–477, 2024.

[8] Xu Cheng, Yuyang Ye, Guoqing He, Qianqian Song, and Peng Cao. Heterogeneous graph attention network based statistical timing library characterization with parasitic RC reduction. In *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 171–176, 2024.

[9] Eunice Naswali, Namhoon Kim, and Pravin Chandran. Fast and accurate library generation leveraging deep learning for OCV modelling. In *Proceedings of International Symposium on Quality Electronic Design (ISQED)*, pages 349–354, 2021.

[10] Seyed Milad Ebrahimipour, Behnam Ghavami, Hamid Mousavi, Mohsen Raji, Zhenman Fang, and Lesley Shannon. Aadam: A fast, accurate, and versatile aging-aware cell library delay model using feed-forward neural network. In *Proceedings of International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[11] Zhifei Zhang, Yang Song, and Hairong Qi. Age progression/regression by conditional adversarial autoencoder. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4352–4360, 2017.

[12] 2020 CAD Contest in fast timing model estimation for new pvt. [Online]. Retrieved July 18, 2023, from https://drive.google.com/file/d/1XDLVPTlumdFpFPv8TnkKE5wlWrd_EYy-/view.

[13] Synopsys, Inc. University software program: SAED_EDK14_FINFET. Retrieved July 18, 2023, from https://www.synopsys.com/academic-research/university.html.

[14] Megan Marsh and Wei-Lii Tan. Improving library characterization quality and runtime with machine learning. [Online]. Retrieved July 18, 2023, from https://resources.sw.siemens.com/en-US/white-paper-improving-library-characterization-with-machine-learning.

[15] Synopsys, Inc. Siliconsmart, version 2020.12. Retrieved December, 2020, from https://www.synopsys.com/academic-research/university.html.

GNN-MLS: Signal Routing in Mixed-Node 3D ICs through GNN-Assisted Metal Layer Sharing

Jiawei Hu[1], Pruek Vanna-iampikul[3], Zhen Zhuang[2], Tsung-Yi Ho[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA
[2]The Chinese University of Hong Kong; [3]Department of Electrical Engineering, Burapha University, Chonburi, Thailand;
{jiaweihu, limsk}@gatech.edu; zhuangzhen1995@gmail.com; tyho@cse.cuhk.edu.hk; pruek.va@eng.buu.ac.th;

***Abstract*—Native 3D Integrated Circuit (3D IC) design offers enhanced performance and density but faces challenges in signal routing due to limited true 3D EDA tool support. Pseudo-3D flows bridge this gap but lack cross-tier optimization, critical for both mixed-node and homogeneous designs. Metal Layer Sharing (MLS) addresses this by enabling cross-tier routing co-optimization but risks timing degradation if not applied strategically. Additionally, MLS creates open connections in hybrid-bonded 3D ICs, making chips untestable. We propose GNN-MLS, a Graph Neural Network-based framework for precise MLS net selection, combined with a tailored DFT solution for robust testability. Experiments show GNN-MLS reduces timing violations by 79% and improves WNS and TNS by 81% and 94%, and moves designs closer to true 3D ICs.**

I. Introduction

Three-dimensional Integrated Circuits (3D ICs) are a promising solution to extend Moore's Law by providing increased device density and performance through vertical integration. However, current Electronic Design Automation (EDA) tools, primarily designed for 2D ICs, face significant limitations in supporting true 3D integration. While dedicated 3D flows have been proposed for placement [1]–[3], true 3D routing remains underexplored, and these methods often fall short of achieving manufacturable quality. As a result, pseudo-3D flows, such as sequential-2D [4], Memory-on-Logic (MoL) [5], and pin-3D [6], have emerged as practical alternatives, leveraging 2D EDA tools to achieve commercially viable 3D designs. However, these flows fail to exploit fine-grained cross-tier optimization, leaving untapped opportunities for performance improvements [7]–[10].

Metal Layer Sharing (MLS) offers a promising pathway to bridge the gap between pseudo-3D and true 3D capabilities by enabling cross-tier routing co-optimization through 2D EDA tools. MLS facilitates the use of routing layers across tiers, as depicted in Figure 1, unlocking additional paths and enhancing timing performance. However, state-of-the-art (SOTA) approaches to MLS rely on heuristic strategies with limited net-level control, leading to indiscriminate MLS application that can degrade timing performance. For example, in our experiments with the MAERI 128PE design [11], SOTA MLS applied to specific nets worsened slack (e.g., net n146095 had slack degraded to -48ps), whereas disabling MLS improved slack to -45ps as shown in Table I. These findings underscore the need for more targeted MLS strategies. Figure 2 illustrates that while SOTA improves timing with MLS, our proposed method achieves far superior results.

Additionally, MLS introduces testability challenges, particularly in hybrid-bonded 3D ICs. Open connections created during fabrication render critical signals unobservable or uncontrollable, leading to non-testable designs. As illustrated in Figure 3, these open connections significantly affect testability in hybrid-bonded designs. Addressing these issues is crucial for realizing MLS's potential in cross-tier co-optimization and advancing pseudo-3D designs toward true 3D integration.

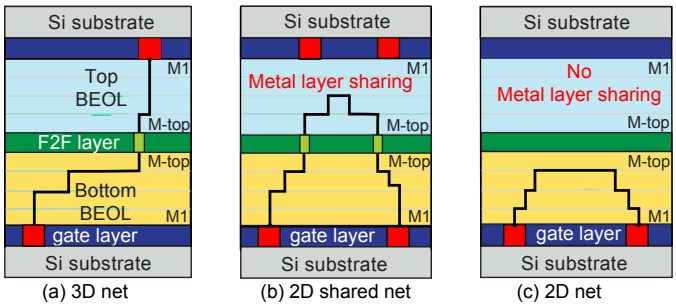


Fig. 1: Illustration of 3-d net, 2d-shared net, and 2d-net.

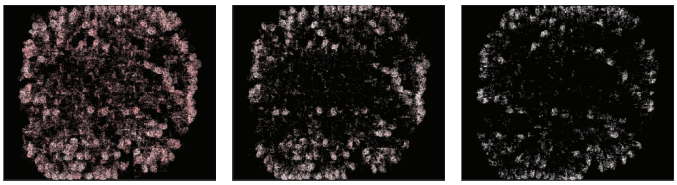
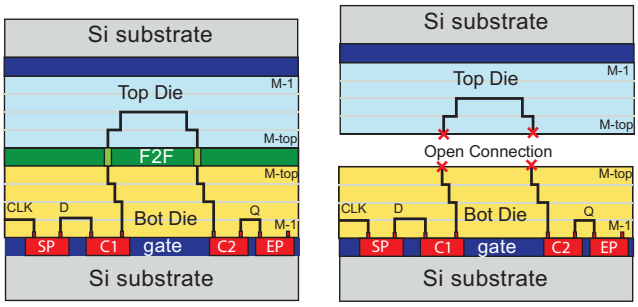


Fig. 2: Timing Violation Points: Registers with violations are highlighted. No MLS has the most violations, SOTA reduces violations by 68%, and GNN-MLS achieves an 80% reduction.

To address these challenges, we propose **GNN-MLS**, a Graph Neural Network-assisted framework for fine-grained, net-level MLS decision-making focused on optimizing timing in 3D ICs. The core challenge lies in evaluating the timing impact of MLS on each net, which traditionally requires exhaustive Static Timing Analysis (STA) across all configurations—a computationally prohibitive process. GNN-MLS eliminates this need by leveraging a novel approach that converts the hyper-edge nature of nets into a node-centric problem, enabling efficient graph-based modeling of timing paths.

Using a transformer-based encoder, GNN-MLS captures the intricate interdependencies among nets along critical timing paths, ensuring precise MLS decisions that prioritize timing improvements. Additionally, we introduce a tailored Design-for-Test (DFT) solution to address testability challenges in hybrid-bonded 3D ICs, ensuring signal observability and controllability across tiers with minimal overhead. By strategically focusing on timing and testability, GNN-MLS advances pseudo-3D flows toward the performance and reliability of true 3D ICs, offering a scalable and practical solution for next-generation integration.

In this work, we propose a comprehensive framework for cross-tier routing co-optimization in 3D ICs, combining GNN-MLS for MLS decision-making, DFT for testability, and PDN design for heterogeneous stacking. Our main contributions are as follows:



(a) Timing Path using MLS (b) Pre-bond Testing

Fig. 3: Testability Issue in Hybrid-Bonded Design with MLS: Each MLS induces an open connection during individual die fabrication, which makes the chip untestable. The timing path consists of start-point (SP), combinational logic (C1-2), and endpoint (EP).

1) We propose GNN-MLS, integrating hyper-graph conversion, a Transformer for net interdependencies, contrastive pretraining, and supervised fine-tuning for MLS decision-making.
2) We design two DFT strategies for MLS in hybrid-bonded 3D ICs, ensuring high test coverage with minimal overhead.
3) We develop a multi-power domain framework and a PDN for mixed-node 3D ICs to support heterogeneous integration experiments.
4) Our framework achieves up to 75% improvement in WNS, 94% in TNS, and a 75% reduction in timing path violations compared to sequential 2D designs.

II. Motivation

A. Why MLS and What are its Limitations?

Metal Layer Sharing (MLS) is a critical innovation for routing efficiency in 3D ICs, addressing key challenges such as congestion and timing optimization. By enabling nets to utilize BEOL layers across tiers, MLS provides additional routing resources and unlocks new pathways for critical connections, which are otherwise constrained in 2D ICs. For example, in the MAERI architecture with 16PE, MLS improves critical path slack from -76 ps without MLS to -18 ps with selective MLS, demonstrating its potential to enhance timing and overall performance.

However, current EDA tools do not model or systematically implement MLS. The only systematic approach to applying MLS is through iterative STA, which evaluates the timing impact of MLS on a net-by-net basis. This process involves repeated disconnection, rerouting, and slack recalculation, making it computationally infeasible for large designs. Furthermore, MLS introduces severe testability challenges in hybrid-bonded 3D ICs, as inter-tier connections disrupt signal propagation and render dies non-testable. These limitations restrict the practical adoption of MLS despite its clear performance benefits.

To address these issues, we propose a comprehensive framework that combines machine learning-based MLS decision-making to eliminate exhaustive STA evaluations with tailored DFT solutions to ensure robust testability. This approach significantly improves the efficiency and practicality of MLS in modern 3D ICs.

B. Graph Neural Networks for Metal Layer Sharing

Metal Layer Sharing (MLS) decisions are essential for optimizing timing in 3D ICs, but systematically applying MLS presents significant challenges. Slack, a key timing metric, depends on the cumulative delays of all nets in a timing path, where changes to one net propagate through others, creating intricate interdependencies. The only systematic approach for evaluating MLS decisions is through Static Timing Analysis (STA), which requires exhaustively exploring the entire MLS configuration space. This process involves iterative disconnection, rerouting, and recalculating slack for each net, making it computationally prohibitive for large designs and delaying cross-tier routing optimization.

TABLE I: Metal layer sharing (MLS) impact on slack with single net of MAERI 128PE using the heterogeneous 3D integration ($16nm$ logic, $28nm$ memory).

Net name	before MLS		after MLS	
	slack (ps)	used metals	slack (ps)	used metals
n480132	-62	M1-6(bot)	-45	M1-6(bot)+M5-6(top)
n146095	-45	M1-4(bot)	-48	M1-6(bot)+M6(top)

To address the challenges of modeling timing paths, we leverage Graph Neural Networks (GNNs) to represent timing paths as directed acyclic graphs (DAGs), where nodes correspond to devices and edges to nets. GNNs provide a natural framework for propagating information across the graph, capturing local dependencies and structural relationships within the path. However, timing paths exhibit complex interdependencies that extend beyond immediate neighbors, requiring the ability to model both local interactions and long-range dependencies effectively. To tackle this, we integrate the Transformer [12] architecture into the GNN framework. Transformers, with their self-attention mechanism, dynamically weigh interactions across nodes, enabling them to model long-range relationships along the path. This ability to capture both sequential and global interactions complements the GNN's structural modeling, making the combined framework particularly powerful for understanding the cascading impact of MLS decisions and optimizing timing paths.

A significant challenge in implementing MLS is the scarcity of labeled data, as generating MLS labels through STA is computationally expensive and impractical at scale. To overcome this, we employ self-supervised learning with Deep Graph InfoMax (DGI) [13] to pretrain the Transformer. DGI maximizes mutual information between global path representations and local node embeddings, enabling the extraction of meaningful features from unlabeled timing path data. These pre-trained embeddings are then fine-tuned using a limited labeled dataset, which includes 500 timing paths each from the heterogeneous A7 single-core and MAERI 128PE designs, as well as 500 timing paths from the corresponding homogeneous configurations. This approach significantly reduces reliance on STA while ensuring efficient and accurate MLS predictions.

By combining the sequential modeling capabilities of Transformers, the graph-structured representation of GNNs, and the data efficiency of DGI, our framework addresses the challenges of interdependencies, computational overhead, and data scarcity. This enables early, efficient, and accurate MLS decision-making, advancing cross-tier routing optimization in 3D IC designs.

III. GNN-MLS Algorithms

Cross-tier co-optimization in 3D ICs with Metal Layer Sharing (MLS) poses significant computational challenges. Evaluating the timing impact of MLS relies on post-routing Static Timing Analysis (STA), which requires iterating through multiple MLS configurations. This process is infeasible for large designs due to its high computational cost. To address this, we propose GNN-MLS, a machine learning-based framework that predicts MLS decisions early in the design flow. By moving MLS decisions to the routing stage, GNN-

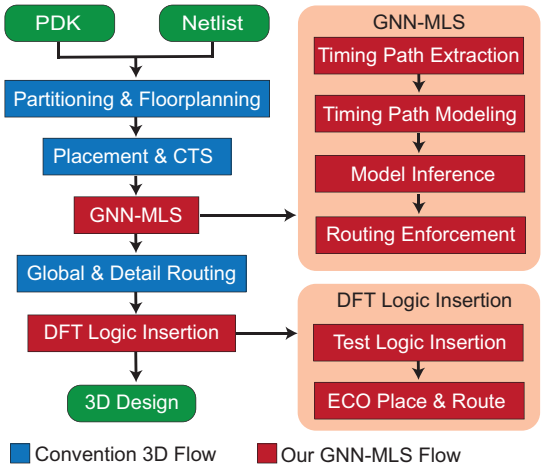


Fig. 4: GNN-MLS Design Flow

MLS reduces dependency on STA and enables efficient cross-tier routing optimization.

A. Design Flow

Our design flow integrates GNN-MLS to tackle timing optimization for monolithic and hybrid-bonded 3D ICs, while also ensuring testability in hybrid-bonded designs. Without MLS-specific tools, current methods rely entirely on post-routing STA to determine MLS decisions, restricting EDA tools from leveraging MLS during earlier stages. This late-stage dependency results in limited optimization opportunities and increased computational overhead. By enabling MLS predictions during routing, GNN-MLS integrates seamlessly into the design flow, facilitating timing optimization and ensuring testability at an earlier stage.

Figure 4 illustrates the key stages of the flow:

1) MLS Decision-Making with GNN-MLS: The GNN-MLS module generates binary MLS decisions for individual nets, indicating whether MLS should be applied. These decisions identify whether MLS should be applied based on a learned node representations. By providing these decisions early, GNN-MLS enables efficient routing optimization, allowing EDA tools to consider MLS during cross-tier routing.
2) Targeted Routing and DFT Integration: Using the MLS decisions from GNN-MLS, targeted routing enforces selective metal layer sharing, ensuring that only specific nets share metal layers across tiers to maximize timing performance. Following routing, custom Design-for-Test (DFT) logic is integrated to ensure testability. This addresses challenges unique to MLS-enabled designs, ensuring that cross-tier connections remain verifiable.

This design flow reduces the computational cost of iterative STA while achieving practical cross-tier co-optimization and addressing testability challenges.

B. Timing Path Modeling

Evaluating the timing impact of MLS on each net before routing is essential to optimize timing performance and eliminate iterative steps in the design flow. This fine-grained decision problem can be formulated as:

***Problem** 1 (Net-Level MLS Decision):* Given a netlist $\mathcal{N}$ and a timing path set $\mathcal{P}$, each net in a timing path should decide whether it uses MLS to maximize the slack of its timing path.

TABLE II: Hand-crafted Features used in our GNN-MLS.

Features	Description	Unit
cell location (x,y)	location of the cells	μm
cell delay	delay of the cells	ps
pin capacitance	pin capacitance of the output pin	pF
wirelength	early-global routing wirelength of the net	μm
wire capacitance	estimated wire capacitance of the net	pF
wire resistance	estimated wire resistance of the net	Ω

The optimal slack for each path p_i is given by:

$$\text{slack}_{\text{opt}}(p_i) = \text{slack}_{\text{2D}}(p_i) + f(\delta(n_1), \delta(n_2), \ldots, \delta(n_{|p_i|})), \quad (1)$$

where $slack_{2D}(p_i)$ represents the slack of p_i in the sequential-2D design (no-MLS). $\delta(n_i)$ is a binary variable to identify if the net $n_i \in \mathcal{N}$ uses MLS. $f(\cdot)$ represents the slack variation caused by the MLS decision compared with the sequential-2D design. The optimization objective then becomes:

$$\max_{\delta(n_i)\in\{0,1\}} \sum_{p_i \in \mathcal{P}} \text{slack}_{\text{opt}}(p_i) \quad (2)$$

However, solving this optimization directly is infeasible due to the exponential number of MLS configurations and the need for STA evaluations to compute $f(\cdot)$. Instead, GNN-MLS approximates this solution by learning to predict the optimal MLS decisions $\delta(n_i)$ based on timing path dependencies. By framing the problem in this manner, GNN-MLS allows efficient decision-making while indirectly optimizing the total slack objective.

We propose a timing path modeling methodology tailored to the unique characteristics of timing paths, as shown in Figure 5. Circuits are first modeled as directed acyclic graphs (DAGs) based on signal transmission. However, multi-pin nets, which connect three or more pins and interconnect multiple devices, introduce significant challenges for MLS decision-making due to their complexity.

To simplify this, each multi-pin net is treated as a hyperedge with a single source node, corresponding to the output pin in the netlist. This transformation allows net-related MLS decisions to be reformulated as node-specific decisions, effectively combining edge features with node features. Graph Transformer architectures then process these features, transforming the initial node attributes into informative embeddings for accurate MLS predictions. The fused features are shown in Table II.

C. Graph Transformer-Based MLS Decisions

The Transformer architecture plays a central role in GNN-MLS, as it effectively captures the interdependencies among MLS decisions across nets within a timing path. The slack improvement for a timing path depends on how MLS is applied to all nets along the path. This interdependence, combined with the sequential nature of timing paths, necessitates a model that can capture both local and global relationships. The entire graph Transformer-based framework is shown in Figure 5.

Traditional GNNs, which aggregate information from immediate neighbors, are insufficient for this task because they rely on fixed neighborhood structures. The Transformer's self-attention mechanism dynamically weighs interactions between all nodes in a path, regardless of distance, making it ideal for capturing long-range dependencies. Additionally, the sequential nature of timing paths is preserved through positional encodings, which ensure that the order of nodes is explicitly encoded in the model. To balance expressiveness and efficiency, the proposed Transformer architecture has three layers. Each layer consists of a three-head self-attention

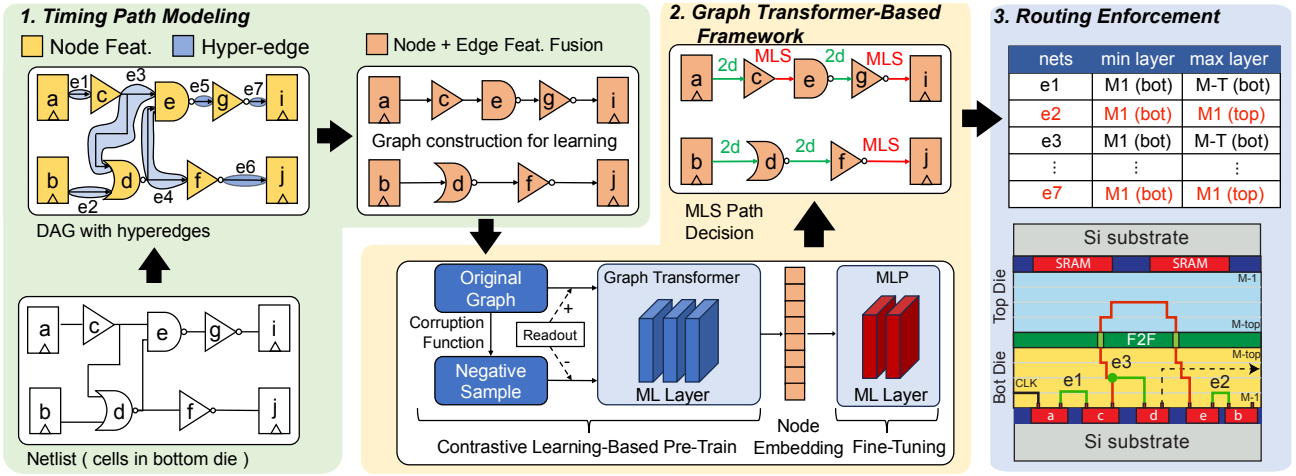


Fig. 5: The GNN-MLS framework converts netlist into hypergraph. It embeds hyperedge features into output-pin-associated nodes and extracts timing paths. The Graph Transformer, pretrained with DGI, processes these paths. MLP fine-tunes the embeddings to make MLS decisions.

mechanism. This architecture is tailored to the complexity of MLS decision-making, addressing the combinatorial dependencies among nets while maintaining computational efficiency.

Obtaining labeled data for MLS decisions, such as STA-based ground truth, is computationally expensive. To address this, we use Deep Graph Infomax (DGI) for self-supervised pretraining, allowing the model to learn effective node embeddings from unlabeled timing path graphs. DGI maximizes the mutual information between global path representations $g(Y)$ and local node embeddings v, ensuring that embeddings capture both structural and timing-critical dependencies. Negative sampling, achieved by perturbing node features, enhances robustness by forcing the model to distinguish true node-path relationships from noise. We adopt sigmoid function σ to map inner product to probability and aid training stability. The DGI loss for a graph Y is defined as:

$$I(Y) = \sum_{v \in Y} \log(\sigma(\langle v, g(Y) \rangle)) + \sum_{v^* \in C(Y)} \log(\sigma(\langle v^*, g(Y) \rangle)) \tag{3}$$

Following DGI pretraining, the Transformer-generated embeddings are fine-tuned using a two-layer Multi-Layer Perceptron with collected labeled data. The MLP maps each node embedding to a binary decision $\delta(n_i)$, indicating whether MLS should be applied. The entire training scheme is shown in Algorithm 1.

Algorithm 1: Training Process for MLS Decision-Making

Input: Set of timing path graphs $\{Y_i\}_{i=1}^{M}$; node features X; binary MLS labels for fine-tuning.
Output: Trained Transformer and MLP.

DGI Pretraining:
for *each timing path graph* Y_i **do**
  Compute node embeddings v using graph Transformer.
  Generate global summary vector $g(Y_i)$.
  Create perturbed graph $C(Y_i)$ for negative samples v^*.
  Compute DGI loss to maximize mutual information.
MLP Fine-Tuning:
for *each labeled timing path graph* Y_i **do**
  Pass DGI-pretrained node embeddings by a 2-layer MLP.
  Train MLP on labels to predict MLS decisions $\delta(n_j)$.

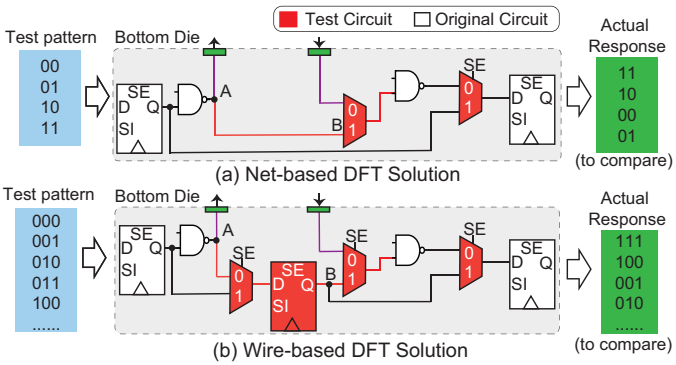


Fig. 6: Two DFT solution for MLS: (a) Net-based DFT where fault can be detected if either outgoing and incoming signal from F2F pads are incorrect, (b) Wire-based DFT where the mechanism can detect if outgoing or incoming F2F pads are fault.

D. DFT Strategy for MLS-Enabled Designs

While Metal Layer Sharing (MLS) significantly improves 3D routing by enabling cross-die connections, it introduces critical testability challenges. Each MLS connection creates an open circuit, disrupting signal propagation and preventing observability and controllability across dies, as shown in Figure 3. This renders upstream signals non-observable and downstream signals non-controllable, making MLS-enabled designs non-testable in hybrid-bonded 3D ICs. Conventional Design-for-Test (DFT) strategies fail to address these challenges, necessitating tailored solutions for MLS testability.

We propose two targeted DFT strategies designed for MLS connections, inserted post-routing to align with their precise locations. The first method, illustrated in Figure 6(a), adds a MUX at the MLS connection, toggling between functional and test modes. In test mode, the scan chain redirects signal flow across the MLS path, enabling observability and controllability. The second method, depicted in Figure 6(b), introduces a scan flip-flop (FF) at the MLS connection, improving test coverage by registering the upstream

DFT method	Total fault	Detected fault	WNS
Net-based DFT	444,296	438,152	-21ps
Wire-based DFT	444,346	438,276	-23ps

TABLE III: Two DFT approaches comparison for MLS nets: (a) net-based MLS DFT, (b) wire-based MLS DFT.

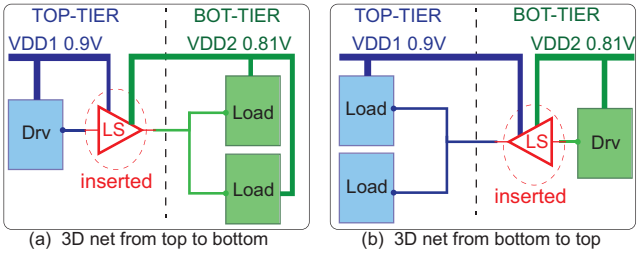


Fig. 7: Power Design for Hetero-Integration.

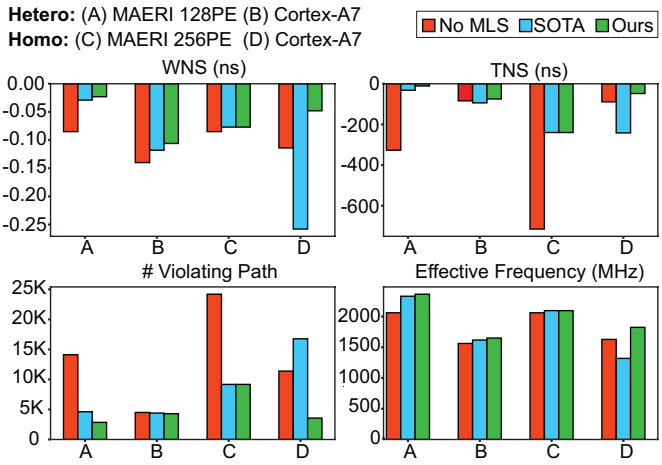


Fig. 8: Timing Metric Comparison on different design benchmark.

signal and driving the downstream signal during testing.

We evaluated these methods on the MAERI 16PE 4BW design with 16 MLS nets. As shown in Table III, while the MUX-based method minimizes additional logic, the FF-based method achieves superior fault coverage, detecting 94 more faults at the cost of 50 additional faults due to added logic. Post-routing ECO adjustments ensure that the timing impact of these solutions remains minimal. These results validate both approaches, with the FF-based method offering enhanced testability for MLS-enabled designs.

E. Power Delivery for Mixed-Node 3D ICs

In mixed-node 3D IC heterogeneous integration setup, we introduce a unique 3D power domain configuration, shown in Figure 7. The top-level operates at 0.9V, with the 28nm memory sub-domain also at 0.9V and the 16nm logic sub-domain at 0.81V. Level shifters are inserted for each 3D signal connection to manage voltage differences across tiers, facilitating reliable communication between the 16nm and 28nm technologies.

The PDN is implemented with specific width and pitch to ensure that the IR-drop of all design are within 10% of lowest VDD (0.81V) as defined in Table IV. The remaining routing resource are utilized for the 2D or MLS nets.

IV. Experimental Results

A. Experimental Setup

We evaluate the GNN-MLS framework on homogeneous and heterogeneous 3D integrations with Face-to-Face (F2F) bonding using three benchmarks: 128PE 32BW MAERI-ARCH, dual-core Cortex A7, and 256PE 64BW MAERI-ARCH. Heterogeneous designs use TSMC $28nm$ for the logic die and $16nm$ for the memory die, while homogeneous designs use $28nm$ for both dies. 3D Place-and-Route (PnR) is performed with Cadence DDI 23.12.000 (Innovus) and the Macro-3D flow [5]. F2F via parameters are configured as size $0.5\mu m$, pitch $1.0\mu m$, resistance 0.5Ω, and capacitance 0.2fF.

We compare GNN-MLS with the state-of-the-art (SOTA) MLS technique [9] and a sequential 2D (no MLS) stacking method, analyzing timing, routing efficiency, and testability. Runtime, based on timing path depth, includes transformer-based training and inference. Results highlight GNN-MLS's advantages in both integration contexts.

B. Results for Heterogeneous 3D ICs

In heterogeneous 3D ICs integrating 16nm logic with 28nm memory, GNN-MLS achieves substantial timing improvements (Figure 8, Table IV). For the MAERI 128PE design, which has a balanced logic-memory ratio, GNN-MLS reduces TNS to -11 ns from -32 ns (SOTA) and -327 ns (No MLS), and decreases violating paths to 2.8K from 4.6K (SOTA) and 14K (No MLS). WNS improves to -23 ps, outperforming both SOTA (-29 ps) and No MLS (-85 ps).

Similarly, in the A7 Dual-Core configuration, GNN-MLS reduces TNS to -75 ns versus -94 ns (SOTA) and -84 ns (No MLS). Violating paths decline to 4.2K from 4.4K (SOTA) and 4.5K (No MLS), with WNS improving to -106 ps compared to -118 ps (SOTA) and -140 ps (No MLS).

Notably, GNN-MLS applies MLS more selectively in heterogeneous designs. For MAERI 128PE, MLS usage drops to 2.4K nets from 9.5K (SOTA); for A7 Dual-Core, usage decreases to 2.6K nets from 3.5K (SOTA). This selective approach demonstrates GNN-MLS's adaptability, applying MLS strategically to maximize timing benefits and minimize overhead.

These results highlight GNN-MLS's ability to optimize timing in heterogeneous designs through targeted and efficient MLS application, meeting the specific demands of hybrid-bonded 3D ICs.

C. Results for Homogeneous 3D ICs

In homogeneous 3D ICs stacking 28nm memory on 28nm logic, GNN-MLS significantly improves timing (Figure 8, Table V). For MAERI 256PE, selected for balanced logic-memory ratio, GNN-MLS reduces TNS to -240 ns from -715 ns (SOTA) and -513 ns (No MLS). Violating paths drop to 9,173 from 24,195 (SOTA) and 16,037 (No MLS), and WNS improves to -0.077 ns, surpassing SOTA (-0.085 ns) and No MLS (-0.083 ns).

In the A7 Dual-Core design, GNN-MLS lowers TNS to -48 ns compared to -242 ns (SOTA) and -89 ns (No MLS). Violating paths decrease to 3,569 from 16,770 (SOTA) and 11,391 (No MLS), with WNS significantly improving to -0.048 ns versus -0.258 ns (SOTA) and -0.114 ns (No MLS).

GNN-MLS also expands MLS usage strategically in homogeneous designs. For MAERI 256PE, MLS coverage increases to 1,600 nets from 870 (SOTA); in A7 Dual-Core, it rises sharply to 73K nets from 8.4K (SOTA). This adaptive approach highlights GNN-MLS's ability to balance extensive MLS coverage with optimal timing.

TABLE IV: PPA metrics comparison between GNN-MLS, State-of-the-Art [9], and SOTA without MLS in Heterogeneous Integration. The frequency is in MHz, power is in mW, run time is in minutes. M-T represents the top-most layer of the memory die where W, P, and U denote the width, pitch, and utilization of the PDN on this metal layer. The highlighted entry denotes the best value among all design options.

	16nm Logic + 28nm Memory					
	MAERI 128PE			A7 Dual-Core		
	No MLS	SOTA	Ours	No MLS	SOTA	Ours
BEOL	6+6			8+8		
Target Freq.	2,500			2,000		
FP (mm^2)	0.38			1.11		
WL (m)	5.23	5.18	5.16	7.60	8.30	8.10
WNS (ps)	-85	-29	-23	-140	-118	-106
TNS (ns)	-327	-32	-11	-84	-94	-75
#Vio. Paths	14,000	4,600	2,800	4,500	4,400	4,200
#MLS Nets	0	9,500	2,370	0	3,542	2,621
Run-Time	-	-	20	-	-	15
Pwr	1,472	1,404	1,389	1,008	1,061	1,052
IR-drop (%)	10.00	9.50	9.40	1.90	2.00	1.98
M-T:W/P/U	2.00μm / 7μm / 14%			2.70μm / 9μm / 30%		
L.S Pwr	40	45	46	31	32	33
Eff. Freq.	2,061	2,330	2,363	1,562	1,618	1,650

TABLE V: PPA metrics comparison between GNN-MLS, State-of-the-Art [9], and SOTA without MLS in Homogeneous Integration. The frequency is in MHz, power is in mW, run time is in minutes. The highlighted entry denotes the best value among all design options.

	28nm Logic + 28nm Memory					
	MAERI 256PE			A7 Dual-Core		
	No MLS	SOTA	Ours	No MLS	SOTA	Ours
Target Freq.	2500			2000		
FP(mm^2)	1.42			1.11		
WL.(m)	14.5	14.6	15.5	14.5	12.1	11.2
WNS (ps)	-83	-85	-77	-114	-258	-48
TNS (ns)	-513	-715	-240	-89	-242	-48
#Vio. Paths	16K	24K	9K	11K	16K	3.5K
#MLS Nets	0	870	1.6K	0	8.4K	73K
Run-Time	-	-	35	-	-	15
Pwr.(mW)	4680	4747	4804	1425	1412	1442
Eff. Freq.	2,070	2,061	2,096	1,628	1,319	1,824

These results confirm GNN-MLS's effectiveness in enhancing timing through targeted MLS application in homogeneous designs.

D. Design-for-Test Results

Hybrid-bonded 3D ICs face testability issues from open MLS connections, affecting signal observability. We introduce a Design-for-Test (DFT) solution using scan flip-flops at critical MLS points with power gating for minimal overhead. DFT is applied to No MLS and GNN-MLS designs (excluding SOTA due to impractical probe pad needs).

As shown in Table VI, the combined GNN-MLS and DFT framework delivers robust testability and timing improvements. MAERI 128PE achieves 98.38% coverage, 75% fewer violating paths and WNS, 94% lower TNS, and a 15% frequency increase. A7 Dual-Core maintains 97.49% coverage, reduces violating paths by 17%, TNS by 10%, WNS by 5%, and frequency increases by 4.3%.

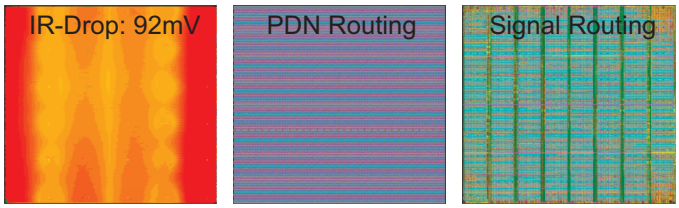
These results confirm the framework effectively addresses MLS testability and enhances timing in hybrid-bonded 3D ICs.

E. Power Delivery Network Design Results

We integrate the PDN into the heterogeneous 3D design, summarized in Table IV, carefully selecting PDN width and spacing to maintain IR-drop within 10% of V_{DD}. Specifically, MAERI 128PE reaches a 10% IR-drop due to its higher power consumption, whereas the A7 Dual-Core achieves a lower IR-drop of 2%, as shown in Figure 9(a). Figures 9(b)–(c) illustrate the sharing of top metal layer resources between PDN and Metal Layer Sharing (MLS) nets, demonstrating effective balancing of power delivery and signal routing. Level shifter (LS) power overhead remains comparable across designs, slightly increasing in SOTA and GNN-MLS due to additional MLS nets, resulting in a total power overhead of about 2–3%.

TABLE VI: Comparison of MLS in Testable Designs between GNN-MLS and State-of-the-Art (28nm / 16nm Technology Node). The frequency is in MHz, power is in mW, run time is in minutes. The highlighted entry denotes the best value among all design options.

	MAERI 128PE		A7 Dual-Core	
	No MLS	GNN-MLS	No MLS	GNN-MLS
Target Freq.	2500	2500	2000	2000
FP. (mm^2)	0.38	0.38	1.11	1.11
WL. (m)	5.95	5.93 (0.3%)	9.40	9.30 (1.1%)
Test Cover.(%)	98.25	98.38 (0.1%)	97.32	97.49 (0.2%)
WNS (ps)	-86	-21 (75%)	-159	-132 (17%)
TNS (ns)	-358.00	-20.00 (94%)	-112.00	-76.00 (32%)
#Vio. Paths	15,321	3,766 (75%)	6,055	5,267 (13%)
#MLS Nets	0	2,425	0	2,536
Run-Time	-	20	-	15
Pwr	1,539	1,523 (1.0%)	1,157	1,152 (0.4%)
Eff. Freq.	2,062	2,375 (15.2%)	1,517	1,582 (4.3%)



(a) (b) (c)

Fig. 9: (a)Hetero-MAERI 128PE IR-drop of 92mV. (b)PDN routing Layout. (c)Signal routing Layout.

V. Conclusion

This work presents GNN-MLS, a data-driven framework that enables efficient and effective cross-tier co-optimization through Metal Layer Sharing (MLS), significantly improving timing performance in 3D IC designs. Complementing this, our tailored Design-for-Test (DFT) solution ensures the testability of MLS connections, addressing a critical challenge in hybrid-bonded 3D ICs. Additionally, our power design supports heterogeneous integration with mixed-node configurations, facilitating robust experimentation in diverse 3D IC setups. Together, these contributions advance pseudo-3D flows toward true 3D IC performance by bridging the gap between 2D-based EDA tools and the demands of native 3D designs. This framework lays the groundwork for manufacturable 3D methodologies, marking a significant step toward fully optimized, high-density 3D ICs for commercial adoption.

VI. Acknowledgement

This work was supported by the Semiconductor Research Corporation under the JUMP 2.0 Center Program (CHIMES), Samsung Advanced Institute of Technology (SAIT) under the AI for Semiconductors Program, and the Ministry of Trade, Industry & Energy of South Korea (1415187652, RS-2023-00234159).

REFERENCES

[1] Peiyu Liao, Yuxuan Zhao, Dawei Guo, Yibo Lin, and Bei Yu. Analytical die-to-die 3-d placement with bistratal wirelength model and gpu acceleration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(6):1624–1637, 2024.

[2] Xueyan Zhao, Shijian Chen, Yihang Qiu, Jiangkao Li, Zhipeng Huang, Biwei Xie, Xingquan Li, and Yungang Bao. ipl-3d: A novel bilevel programming model for die-to-die placement. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[3] Jai-Ming Lin, Yu-Chien Lin, Hsuan Kung, and Wei-Yuan Lin. Hyplace-3d: A hybrid placement approach for 3d ics using space transformation technique. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–8, 2023.

[4] Nesara Eranna Bethur, Anthony Agnesina, Moritz Brunion, Alberto Garcia-Ortiz, Francky Catthoor, Dragomir Milojevic, Manu Komalan, Matheus Cavalcante, Samuel Riedel, Luca Benini, and Sung Kyu Lim. Hier-3d: A methodology for physical hierarchy exploration of 3-d ics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(7):1957–1970, 2024.

[5] Anthony Agnesina, Moritz Brunion, Jinwoo Kim, Alberto Garcia-Ortiz, Dragomir Milojevic, Francky Catthoor, Manu Perumkunnil, and Sung Kyu Lim. Power, performance, area and cost analysis of memory-on-logic face-to-face bonded 3d processor designs. In *2021 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 1–6, 2021.

[6] Sai Surya Kiran Pentapati, Kyungwook Chang, Vassilios Gerousis, Rwik Sengupta, and Sung Kyu Lim. Pin-3d: A physical synthesis and post-layout optimization flow for heterogeneous monolithic 3d ics. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2020.

[7] T. Kim et al. Multi-Stack Wafer Bonding Demonstration utilizing Cu to Cu Hybrid Bonding and TSV enabling Diverse 3D Integration. In *2021 IEEE 71st Electronic Components and Technology Conference (ECTC)*, pages 415–419, 2021.

[8] Christopher Netzband, Kevin Ryan, Yuji Mimura, Son Ilseok, Hirokazu Aizawa, Nathan Ip, Xuemei Chen, Hideyuki Fukushima, and Shinichi Tan. 0.5 m pitch next generation hybrid bonding with high alignment accuracy for 3d integration. In *2023 IEEE 73rd Electronic Components and Technology Conference (ECTC)*, pages 1100–1104, 2023.

[9] Sai Pentapati and Sung Kyu Lim. Metal layer sharing: A routing optimization technique for monolithic 3d ics. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 30(9):1355–1367, 2022.

[10] H. Park et al. Pseudo-3D Physical Design Flow for Monolithic 3D ICs: Comparisons and Enhancements. *ACM Trans. Des. Autom. Electron. Syst.*, 26(5), jun 2021.

[11] Hyoukjun Kwon, Ananda Samajdar, and Tushar Krishna. MAERI: Enabling flexible dataflow mapping over dnn accelerators via reconfigurable interconnects. *ACM SIGPLAN Notices*, 53(2):461–475, 2018.

[12] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[13] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

RE3: Finding Refinement Relations with Relational Mapping Abstraction

You Li*, Guannan Zhao*, Yunqi He, and Hai Zhou
Northwestern University, Evanston, USA
{you.li, gnzhao, yunqi.he}@u.northwestern.edu, haizhou@northwestern.edu

***Abstract*—A refinement relation captures the state equivalence between two sequential circuits. It finds applications in various tasks of VLSI design automation, including regression verification, behavioral model synthesis, assertion synthesis, and design space exploration. However, manually constructing a refinement relation requires an engineer to have both domain knowledge and expertise in formal methods, which is especially challenging for complex designs after significant transformations. This paper presents a rigorous and efficient sequential equivalence checking algorithm for non-cycle-accurate designs. The algorithm can automatically find a concise and human-comprehensible refinement relation between two designs, helping engineers understand the essence of design transformations. We demonstrate the usefulness and efficiency of the proposed algorithm with experiments and case studies. In particular, we showcase how refinement relations can facilitate error detection and correction for LLM-generated RTL designs.**

I. Introduction

Design engineers and optimization tools are performing even more aggressive sequential transformations to meet the growing demands of high performance and low power consumption. Examples of sequential transformations include retiming, pipelining, pre-computation, resource reallocation, clock gating, and memory partitioning. Recently, adaptive pipelining and latency-insensitive design methodologies have been proposed to further improve the throughput of high-performance integrated circuits. These techniques can cause irregular data dependencies and dynamic latencies [1] and are prone to error.

Sequential equivalence checking is the key enabler of these transformations. It assures that the two designs before and after a series of transformations are functionally equivalent across clock cycles. Traditional sequential equivalence checking algorithms require that the two designs are cycle-accurate and have a static one-to-one latch mapping. However, these assumptions are no longer valid after radical sequential transformations. On the other hand, non-cycle-accurate sequential equivalence checking algorithms [2], [3] can tolerate timing differences between the two designs. Nevertheless, they encounter scalability issues on large industrial designs because their search space is a product of the state space of the two designs and the space of all possible alignment patterns.

A refinement relation captures the state equivalence between two functionally equivalent designs. It has wide applications in various stages of electronic design automation. For instance, regression verification leverages refinement relations to determine whether a sequence of successively modified designs are functionally equivalent [4], [5]. Behavioral model synthesis takes a refinement relation to generate a system-level reference model using program synthesis techniques [6], [7]. Refinement relation is also crucial for formal-guided design space exploration and root cause analysis [8]. Notably, the emerging Instruction-level Abstraction (ILA) framework relies on refinement relations for various tasks, including property checking and assertion synthesis [9], [10]. Manually constructing a refinement relation requires both domain knowledge and expertise in formal methods. It is notoriously difficult for humans to write a correct refinement relation for complex designs after significant transformations [6], [9].

This paper presents RE3, an efficient symbolic model checking algorithm for non-cycle-accurate sequential equivalence checking. RE3 employs an orchestration of techniques to automatically extract a concise and human-comprehensible refinement relation from two equivalent hardware designs. Specifically, it *i)* reduces the problem complexity by dividing the equivalence checking problem into two tasks: synthesizing a scheduling function and finding a corresponding inductive invariant; *ii)* steers the searching process towards a simple refinement relation by controlling the granularity of the scheduling function; *iii)* captures the dynamic latch mapping between two hardware designs by constructing a dedicated relational mapping abstraction domain; *iv)* filters out bad states with a lightweight simulation-based self-alignment mechanism; and *v)* ensures the incrementality of the model checking process and maintains the compatibility with state-of-the-art model checking algorithms. With all those techniques, RE3 addresses the challenges in non-cycle-accurate sequential equivalence checking and generates high-quality refinement relations to facilitate various downstream tasks.

This paper makes the following key contributions:

- It proposes a rigorous and efficient non-cycle-accurate sequential equivalence checking algorithm;
- It develops a novel abstraction scheme dedicated to hardware equivalence checking;
- It presents the first approach to automatically synthesize a concise and human-comprehensible refinement relation for hardware designs;
- It demonstrates the roles and potentials of refinement relations in conventional and AI-based VLSI design and verification flows.

II. Problem Definition and Analysis

A. Preliminaries

We consider standard first-order logic. A *term* is a variable or a *function* symbol. The terms with non-Boolean values are also referred to as *words*. A *predicate* represents a relation over a set of terms and evaluates to a Boolean value. An *atom* is either a predicate symbol or a Boolean variable. A *formula* is built from atoms with propositional logic. A *literal* is either an atom or a negated atom. A *clause* is a disjunction of literals, whereas a *cube* is a conjunction of literals.

A transition system M is defined as a tuple $\langle X, I, T\rangle$, where X denotes the set of *state variables*, X' denotes the corresponding set of next-state variables, $I(X)$ is a formula representing the *initial condition*, and $T(X, X')$ is a formula representing the *transition relation*. It is a common practice to model input variables as additional state variables [11]–[13] such that $X = X_{state} \cup X_{in}$. The next-state input variables X'_{in} are either unconstrained or controlled by an external specification. A state $s \in S$ is a full assignment to all state variables.

*Equal contribution.

†This work is partially supported by the National Science Foundation under grants 2113704 and 2148177.

We say s is a ϕ-state if s satisfies a formula ϕ modulo the underlying theory. A formula ψ *implies* another formula ϕ, denoted by $\psi \Rightarrow \phi$, if all state satisfying ψ also satisfies ϕ. A finite or infinite *path* of M is a sequence of states such that the first state is an I-state and all consecutive steps satisfy $T(X, X')$. A path is a ϕ-path if all states along the path are ϕ-states.

M satisfies a safety property P if all paths of M are P-paths. A prominent method to prove safety is through finding an *inductive invariant*, Inv, such that

$$\text{(a) } I \Rightarrow Inv, \text{ (b) } Inv \wedge T \Rightarrow Inv', \text{ (c) } Inv \Rightarrow P. \tag{1}$$

We require M to satisfy a progress property: its output value should eventually change unless it reaches a designated terminal state. This property can either be enforced by the designer or checked by a standard tool [14].

B. Non-cycle-accurate Sequential Equivalence

A sequence π *stutters* at step k if it keeps the same value for indices k and $k+1$. For instance, the sequence $\langle a, a, b, c, \cdots \rangle$ stutters at step 1. We denote the *stutter-free sequence* of π as $\natural\pi$, which is obtained by removing all stuttering steps from π. Two sequences π_1 and π_2 are *equivalent modulo stuttering*, denoted as $\pi_1 \simeq \pi_2$, if $\natural\pi_1 = \natural\pi_2$ [11]. For instance, $\natural\langle a, a, b, c\rangle = \langle a, b, c\rangle$, and $\langle a, a, b, c\rangle \simeq \langle a, b, c, c\rangle$.

A hardware design can be refined across different abstraction levels or optimized with sequential transformation techniques. A pair of designs before and after a *sequential transformation* have the same functionality if they always exhibit the same observable behavior modulo timing differences:

Definition 1. *M_A and M_B are observational equivalent if and only if for every valid input sequence, the corresponding output sequences are equivalent modulo stuttering.*

C. Refinement Relation

The VLSI design flow can be viewed as an iterative refinement process. Each refinement step modifies the design while it preserves the original functionality. A *refinement relation* captures the correspondence between the states of the two designs before and after a sequential transformation:

Definition 2. *$R \subseteq S_A \times S_B$ is a refinement relation for M_A and M_B if the following conditions hold:*

- $I_\times(s_A, s_B) \Rightarrow (s_A \; R \; s_B)$, *(2a)*
- $(s_A \; R \; s_B) \Rightarrow (s'_A \; R \; s'_B \;\vee\; s'_A \; R \; s_B \;\vee\; s_A \; R \; s'_B)$, *(2b)*
- $(s_A \; R \; s_B) \Rightarrow P_\times(s_A, s_B)$. *(2c)*

$I_\times$ represents the *initial correspondence* of the two designs. As discussed in §II-A, the input sequence can be encoded into the initial states of both designs. In this situation, the initial correspondence requires that the X_{in} parts are identical and the X_{state} parts are the corresponding reset states. $I_\times$ can also be specified by the user. $P_\times$ denotes the *observational equivalence property*, defined as $P_\times(s_A, s_B) \triangleq (s_A.out = s_B.out)$, where out represents the output value of the associated state. If some intermediate states are considered non-observable, a special value τ can be assigned to their output values.

The refinement relation in Definition 2 is a bisimulation equivalence relation. It states that *a)* all pairs satisfying the initial correspondence are related by R; *b)* for every pair related by R, there exists a stuttering pattern such that the next pair also satisfies R; *c)* for all pairs related by R, the two designs produce the same output. The existence of a refinement relation implies functional equivalence of the two designs:

Listing 1 $gcd_A(x, y)$

```
while (x − y) ≠ 0 do
    x ← (x−y) > 0 ? (x−y) : x
    y ← (y−x) > 0 ? (y−x) : y
output x
```

Listing 2 $gcd_B(x, y)$

```
while (x − y) ≠ 0 do
    x ← (x−y) > 0 ? (x−y) : y
    y ← (x−y) > 0 ? y : x
output x
```

Lemma 3. *M_A and M_B are observational equivalent if there exists a refinement relation R for M_A and M_B.*

Proof. The existence of a refinement relation ensures that for any input sequence (2a), there is a way to insert stuttering steps (2b), such that the two designs produce identical output sequences (2c) denoted by π. Inserting infinite stuttering steps to either M_A or M_B before they terminate, while maintaining R, would contradict with their progress property. This ensures $\pi_A \simeq \pi$ and $\pi_B \simeq \pi$, and hence $\pi_A \simeq \pi_B$. □

D. Problem Statement

This paper re-investigates the non-cycle-accurate sequential equivalence checking problem. Our objective is to devise an efficient model checking algorithm that can solve both equivalent and non-equivalent instances. Moreover, the algorithm should compute a compact and human-readable refinement relation for equivalent instances.

E. Running Example

The Euclid's algorithm computes the *greatest common divisor (gcd)* of two natural numbers. It iteratively subtracts the smaller number from the greater one until the two numbers are equal. gcd_A (Listing 1) and gcd_B (Listing 2) are two RTL implementations of the algorithm. In gcd_A, two subtractors are initiated simultaneously to compute both $x - y$ and $y - x$. In gcd_B, the designer allocates only one subtractor, so the design needs to switch x and y when x is less than y.

The two designs have the same functionality but are different in timing. Suppose the initial values of x and y are 5 and 1, respectively. Both designs take 4 clock cycles to reach termination. On the other hand, suppose the initial values are 5 and 3. gcd_A takes 3 cycles $\langle(5,3),(2,3),(2,1),(1,1)\rangle$, while gcd_B takes 5 cycles $\langle(5,3),(2,3),(3,2),(1,2),(2,1),(1,1)\rangle$. Notice that the two designs have a *variable period* respect to each other depending on the inputs. Furthermore, there exists no static state mapping or internal node mapping between the two designs.

A refinement relation can capture the correspondence between the states of the two designs, and thus prove their equivalence. For example, $R \triangleq (x_A = x_B \wedge y_A = y_B) \vee (x_A = y_B \wedge x_B = y_A)$ is a simple refinement relation satisfying all the conditions (2a)-(2c).

III. The Refinement Relation Searching Algorithm

A. Overview

This section presents RE3, a rigorous and efficient algorithm to check observational equivalence. Given two hardware designs M_A and M_B, it generates a compact and human-comprehensible refinement relation if they are equivalent, or an input sequence as the counterexample if they are non-equivalent. The general workflow of RE3 is shown in Fig. 1.

RE3 first constructs a *composed transition system* $M_\times$ from M_A and M_B (§III-B). It then leverages a new abstraction paradigm, *relational mapping abstraction*, to implicitly build an abstract model for $M_\times$ (§III-E). On the same abstract domain, the algorithm synthesizes a *scheduling function* that controls the alignment of $M_\times$ (§III-B). Afterward, it launches an IC3-style model checking procedure to find

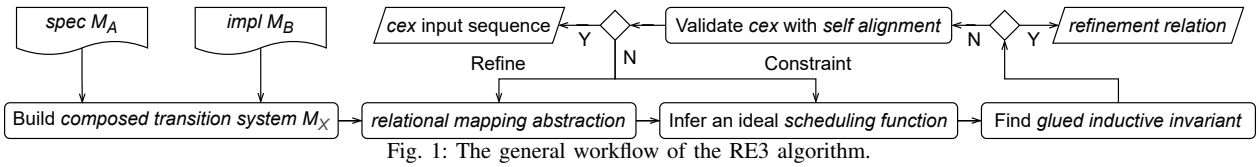


Fig. 1: The general workflow of the RE3 algorithm.

a *glued inductive invariant* $Inv_\times$ (§III-F). If that procedure succeeds, the inductive invariant contains a *relational transition diagram*, which represents the desired refinement relation (§III-D). If that procedure fails, RE3 launches a simulation on the concrete domain using the *self-alignment mechanism* (§III-C) to validate the abstract counterexample. If that validation passes, the algorithm returns an input sequence corresponding to the counterexample. Otherwise, it refines either the scheduling function or the abstract domain to eliminate the spurious counterexample and repeat the verification process.

B. Composed Transition System

Self composition [15], [16] is recently developed to verify hyper-safety properties on software programs. We employ self composition to divide the sequential equivalence checking problem into two separate problems: *i)* inferring a *scheduling function* f_s for $M_\times$, and *ii)* finding a glued inductive invariant $Inv_\times$ with respect to f_s. For each *glued state* $s_\times \in S_\times$, where $S_\times \triangleq S_A \times S_B$, f_s controls which of M_A and M_B will move forward in the next clock cycle by selecting a *stuttering pattern*:

Definition 4. *A scheduling function* $f_s : S_\times \to \{M_A, M_B, M_A \cup M_B\}$ *maps each glued state to a non-empty set of designs that are to execute in the next clock cycle.*

It is usually the case that a high-level specification M_A always runs no slower than its implementation M_B, as the former typically has fewer resource constraints. In this situation, we let f_s output a Boolean value: `true` represents a *synchronizing* transition where both M_A and M_B move forward, and `false` represents a *stuttering* transition where only M_B moves forward. Synthesizing f_s thus becomes a binary classification problem in the glued state space.

A composed transition system $M_\times$ is a parallel composition of M_A and M_B. It executes deterministically when augmented with f_s:

Definition 5. *Given two designs* M_A, M_B *and a scheduling function* f_s, *the composed transition system* $M_\times(f_s)$ *is a tuple* $\langle X_A \cup X_B, I_\times, T_\times \rangle$, *where* $I_\times$ *is the initial correspondence (§II-C), and*

$T_\times(s_A, s_B, s'_A, s'_B) \triangleq$
$\wedge$ `if` $M_A \in f_s(s_A, s_B)$ `then` $T_A(s_A, s'_A)$ `else` $s'_A = s_A$
$\wedge$ `if` $M_B \in f_s(s_A, s_B)$ `then` $T_B(s_B, s'_B)$ `else` $s'_B = s_B$.

In Definition 5, the output of f_s at a certain glued state controls whether to stall a branch of $M_\times$. With $M_\times(f_s)$, the problem of finding a refinement relation can be reduced to an ordinary safety checking problem:

Lemma 6. *There exists a refinement relation* R *for* M_A *and* M_B, *if there exists a scheduling function* f_s *and a glued inductive invariant* $Inv_\times$ *for* $M_\times(f_s)$, *such that the following conditions hold:*
- $I_\times \Rightarrow Inv_\times$, *(6a)*
- $Inv_\times \wedge T_\times \Rightarrow Inv'_\times$, *(6b)*
- $Inv_\times \Rightarrow P_\times$. *(6c)*

Proof. Let R equal $Inv_\times$, *i.e.*, for each $s_\times$ within $Inv_\times$, the corresponding s_A and s_B are related by R. Conditions (2a) and (2c) can be immediately derived from (6a) and (6c). Meanwhile, f_s explicitly selects a stuttering pattern for each $s_\times$ to form $T_\times$. Therefore, (2b) can also be derived from (6b). □

Lemma 6 suggests a method to convert the non-cycle-accurate equivalence checking problem to a safety checking problem. In §III-F, we present a comprehensive model checking algorithm to find both f_s and $Inv_\times$ for hardware designs.

C. Self-alignment Mechanism

A glued inductive invariant should not contain any *bad states* that will inevitably reach a $\neg P_\times$-state. With the *self-alignment mechanism*, we can quickly determine whether a concrete glued state $s_\times$ is a bad state. In comparison, property directed self composition (PDSC) [15], [16] needs to account for all stuttering patterns. Specifically, it constructs a tree whose root is $s_\times$ and whose edges are the valid stuttering patterns. Then it traverses the tree in a reverse topological order. A state is labeled as a bad state if it is a $\neg P_\times$-state or if all of its successors are bad states.

The self-alignment mechanism considers a single stuttering pattern for each glued state. Specifically, if the output value of one design changes and that of the other design remains the same, the mechanism stalls the first design while it executes the second design in the next clock cycle. Otherwise, it executes both designs simultaneously in the next clock cycle. Kairos [2] uses a similar strategy to verify hardware modules, but it assumes that handshake signals are always available.

Theorem 7 describes the relationships among observational equivalence, scheduling function, glued inductive invariant, and the self-alignment mechanism. We omit the proof due to space limit.

Theorem 7. *The following statements are always equivalent:*
- M_A *and* M_B *are observational equivalent;* *(7a)*
- *A refinement relation* R *exists for* M_A *and* M_B; *(7b)*
- *There exists a scheduling function* f_s *and a corresponding glued inductive invariant* $Inv_\times$ *for* $M_\times$; *(7c)*
- *There exists a scheduling function* f_s *such that all paths of* $M_\times(f_s)$ *are observational equivalent paths;* *(7d)*
- *All paths produced by the self-alignment mechanism on* $M_\times$ *are observational equivalent paths.* *(7e)*

Lemma 8. *A glued state* $s_\times$ *is a bad state if and only if the two output sequences produced by the self-alignment mechanism starting from* $s_\times$ *are different.*

Lemma 8 can be proved by treating $s_\times$ as $I_\times$ and using the same reasoning as the *(7a)* $\Rightarrow$ *(7e)* proof in Theorem 7. It suggests that we can check whether $s_\times$ is bad by running a simulation with the self-alignment mechanism from $s_\times$.

Based on Theorem 7, a viable approach to verify observational equivalence is to launch model checking on the composed transition system augmented by the self-alignment mechanism. However, the main challenge lies in the "quality" of the scheduling function. Some scheduling functions admit easier-to-find inductive invariants, which can lead to more concise refinement relations. RE3 simultaneously synthesizes the scheduling function and finds the corresponding inductive invariant to accelerate convergence and maximize the chance of finding a simple inductive invariant.

D. Relational Transition Diagram

Abstraction techniques can reduce the complexity of the model checking problem while preserving the facts relevant to the problem. Consider an abstraction function $\alpha : S_\times \to \widehat{S_\times}$ that maps concrete glued states into abstract ones. There is an abstract transition between two abstract states $\widehat{s}_1$ and $\widehat{s}_2$ if there is a concrete transition between any pair of concrete states s_1 and s_2, such that $\alpha(s_1) = \widehat{s}_1$ and $\alpha(s_2) = \widehat{s}_2$. Suppose we have found a glued inductive invariant $\widehat{Inv_\times}$ on an abstract domain. Then we can construct a *relational transition diagram* as follows:

Definition 9. *A relational transition diagram is a directed graph whose vertices are all the abstract glued states within $\widehat{Inv_\times}$ and whose edges are the abstract transitions between these states.*

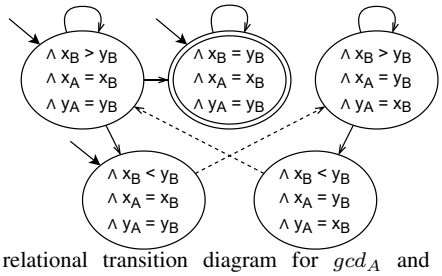


Fig. 2: A relational transition diagram for gcd_A and gcd_B when $f_s := x_B \geq y_B$? `true` : `false`. Solid (resp. dashed) arrows represent *synchronizing* (resp. *stuttering*) transitions. An input arrow indicates an initial state and a double circle indicates a terminal state.

As an illustrative example, Fig. 2 shows a relational transition diagram for gcd_A and gcd_B. Recall that gcd_A is always no slower than gcd_B. We choose a scheduling function that executes both designs when $x_B \geq y_B$ and executes only gcd_B otherwise. This scheduling function leads to a simple diagram that consists of only five abstract states.

Because a relational transition diagram is constructed from a glued inductive invariant and a valid scheduling function, it induces a transition relation R (Lemma 6). Two states $s_A \in S_A$ and $s_B \in S_B$ are related by R if they are glued together, *i.e.,* $\alpha((s_A, s_B))$ is in the diagram. In the *gcd* example, $s_A = (5, 3)$ and $s_B = (5, 3)$ are glued together by the top-left state in Fig. 2.

In the rest of this section, we will discuss how to construct an abstract domain that allows a compact relational transition diagram, and how to efficiently search for a glued inductive invariant.

E. Relational Mapping Abstraction

Predicate abstraction [17] and syntax-guided abstraction [13] are two prevalent abstraction schemes for hardware verification. Predicate abstraction introduces a set of predicates over the state variables. A predicate-abstracted state represents all those concrete states with the same valuation to the predicates. Syntax-guided abstraction utilizes the terms in the syntax of the design to encode the state space. A syntax-abstracted state is a *partition assignment* that captures all equality and inequality among the terms. For the gcd_A example, the concrete state $(2, 4)$ can be mapped to the abstract state $(\neg(x-y) > 0) \ \wedge \ ((y-x) > 0) \ \wedge \ \{x, y-x \mid y \mid x-y\}$, where vertical bars divide terms and formulas into equivalent classes.

During our experiments on hardware equivalence checking, we make several key observations on abstraction schemes. First, M_A and M_B operate on two disjoint sets of state variables. Therefore, we introduce two separate sets of predicates over the corresponding sets of state variables. This can simplify the generation of new predicates and thus reduce the search space of a model checking algorithm. Second, we notice that latches* are the first-class citizens in an equivalence checking problem. A conditional latch mapping almost always exists for a pair of equivalent designs [9]. Hence, an abstraction scheme can benefit from explicitly capturing the one-to-one correspondence between a subset of latches from both designs. Third, we notice that the equivalence of latches is more important than the non-equivalence of latches. Thus, we only add constraints for the pairs of equivalent of latches. Compared with syntax-guided abstraction, this approach can usually significantly reduce the number of abstract states.

In light of these observations, we propose a novel abstraction scheme, *relational mapping abstraction* (RMA), that is dedicated to hardware equivalence checking. An RMA abstracted state is represented by an *r-cube* consisting of three fields: *r-cube* $\triangleq pred_A \wedge pred_B \wedge mapping$. Each field is either empty or a conjunction of atomic predicates. The state variables of M_A can appear only in $pred_A$ and the left-hand side of $mapping$, whereas the state variables of M_B can appear only in $pred_B$ and the right-hand side of $mapping$. Intuitively, the $mapping$ field explicitly expresses the latch mapping, while $pred_A$ and $pred_B$ specify the conditions for that mapping.

A glued inductive invariant $\widehat{Inv_\times}$ on an RMA abstract domain is represented as a disjunction of r-cubes. Such an inductive invariant naturally gives rise to a relational transition diagram (§III-D). For instance, every node in Fig. 2 is represented by an r-cube. The disjunction of all those r-cubes comprises an inductive invariant of the the composed transition system $M_\times(f_s)$.

F. Finding Refinement Relation with RE3

Algorithm 1 The RE3 algorithm.

- **Input:** high-level design M_A, low-level design M_B, initial correspondence $I_\times$.
- **Output:** a relational transition diagram $D_\times$ that represents the refinement relation if M_A and M_B are observational equivalent; a concrete witness of non-equivalence $s^*_\times \in I_\times$ otherwise.

```
1: Construct M× from M_A and M_B                          ▷ §III-B
2: Initialize α with initial predicates                   ▷ abstract function, §III-E
3: while true do
4:     F_s ← ⊤                                            ▷ reset constraints on scheduling function, §III-B
5:     while true do
6:         (reachable, Inv×, cex) ← isReachable(M×(F_s), α)
7:         if reachable = false then                      ▷ found inductive invariant
8:             D× ← buildDiagram(Inv×, F_s, α)            ▷ §III-D
9:             return D×
10:        else                                           ▷ found abstract counterexample trace
11:            (spurious, s*×) ← isSpurious(cex)
12:            if spurious = false then return s*×        ▷ witness
13:            else
14:                F_s ← addConstraints(F_s, cex, α)
15:                if F_s is invalid then break
16:    α ← refine(α)
```

The RE3 Algorithm. A high-level description of RE3 is given in Algorithm 1. Leveraging the implicit abstraction technique [17], RE3 exploits relational mapping abstraction throughout its execution, and we omit *hat* symbols in the pseudocode to improve readability. The algorithm starts by constructing a composed transition system $M_\times$ from the two designs (Line 1). If the user has extracted a set of control statements from the designs, RE3 can initialize the abstract domain

*In this context, "latches" can refer to any memory elements at either bit-level or word-level.

with those statements (Line 2). The algorithm initiates a nested loop to find a compact refinement relation. In every iteration of the outer loop, RE3 incrementally refines a non-deterministic set of scheduling functions F_s, which is initially unconstrained. In every iteration of the inner loop, RE3 calls an IC3-style [18], [19] model checking procedure on $M_\times$ augmented by F_s (Line 6). If $\neg P_\times$ is unreachable from $I_\times$ on the current abstract domain, a glued inductive invariant $Inv_\times$ is returned by the model checking procedure. In this case, RE3 builds a relational transition diagram (Line 8) that represents the refinement relation and provides it to the user. Otherwise, an abstract counterexample trace is returned by the model checking procedure. RE3 validates the counterexample with the self-alignment mechanism (Line 11). If the validation passes, a concrete glued state $s^*_\times \in I_\times$ is extracted as a witness to prove that M_A and M_B are not observational equivalent (Line 12). If the counterexample turns out to be spurious, RE3 strengthens the constraints on F_s to eliminate the counterexample and continues to the next iteration (Line 14). However, if F_s is no longer *valid* after adding the new constraints (Line 15), RE3 exits to the outer loop and refines the abstract domain (Line 16). It resets the constraints on F_s before it enters the inner loop again (Line 4).

In the following, we highlight several key components and features of the RE3 algorithm.

• RE3 reverses the sequence of frames so that the relational mapping abstraction scheme can be compatible with the IC3 framework. Recall that the RMA scheme represents a glued inductive invariant as a disjunction of r-cubes. On the other hand, a frame in the IC3 framework is a conjunction of clauses. RE3 takes the negation of the former during model checking to ensure compatibility. Accordingly, RE3 reverses the order of the frames, such that $F_0 = \neg P_\times$ and $F_i \Rightarrow \neg I_\times$.

• RE3 navigates the searching process towards finding a concise refinement relation. It always starts from an unconstrained set of scheduling functions and gradually refines the set with spurious counterexamples. By controlling the granularity, the algorithm is more likely to find a general and compact inductive invariant. Besides, RE3 prioritizes the refinement of scheduling functions over the refinement of the abstract domain. The latter is executed only when necessary, *i.e.,* all the remaining scheduling functions are invalid. We adopt this strategy because the latter dominates the complexity of model checking and the refinement relation, while the former can be restored periodically.

• RE3 utilizes a lightweight simulation-based approach to find bad states and thus accelerate convergence. Whenever a counterexample trace is returned by the *isReachable* procedure, or a proof obligation is added to the priority queue, RE3 randomly samples N concrete glued states within the targeted abstract state. RE3 runs a simulation on the concrete domain with the self-alignment mechanism (§III-C) from each of those states. All resulting bad states are permanently recorded and utilized by all subsequent invocations of the *isReachable* procedure. Because this approach is speculative, missing a bad state during sampling does not affect the soundness of RE3.

• RE3 ensures that the model checking procedures in every iteration of the outer loop are fully incremental. Each time an abstract counterexample trace is returned by the *isReachable* procedure, RE3 analyses the trace and adds new scheduling constraints to F_s. During this process, the transition relation of $M_\times(F_s)$, denoted as $T_\times$, is monotonically strengthened. This ensures that the whole sequence of frames can be reused in the next execution. Furthermore, RE3 labels a glued abstract state as a *temporary bad state* if every one of its successors is a temporary bad state or contains a concrete bad state. Once an $I_\times$-state is contained in a temporary bad state, F_s becomes *invalid*, *i.e.*, it is impossible to find an $Inv_\times$ for any of the remaining scheduling functions within F_s. In that case, RE3 resets F_s, removes all temporary bad states, and calls the *refine* procedure.

• RE3 employs an efficient strategy to eliminate the counterexample tree and thus refine the abstract domain. Briefly speaking, it walks along a single path from the root until it finds a suitable abstract state to split. For $pred_A$ and $pred_B$, it leverages *interpolation* to generate new predicates to split the targeted state. For the $mapping$ field, it collects a set of potentially equivalent latch pairs by simulating the two designs, utilizes *unsat core* extraction to find a minimal set of those pairs, and adds them to $mapping$. Finally, RE3 chooses the simplest one among the three fields to settle down the refinement.

Correctness and Termination. We rely on the correctness of the original IC3 algorithm to establish the correctness of the RE3 algorithm. When RE3 returns a refinement relation, it must have found a valid scheduling function and an associated inductive invariant. According to Theorem 7, M_A and M_B must be observational equivalent. On the other hand, the two designs must be non-equivalent when RE3 returns a concrete witness of non-equivalence that satisfies the initial correspondence. This is guaranteed by the correctness of the self-alignment mechanism.

No complexity bounds exist for model checking algorithms if the composed transition system is in the infinite state space. Nevertheless, if all variables of the two designs are constant-sized bit vectors, RE3 will eventually terminate. This is because the abstraction refinement process is strictly monotonic, while the scheduling function pruning process and the IC3 converging process are also strictly monotonic on a fixed abstract domain.

IV. Evaluation

A. Experimental Results

We implemented RE3 in Python on top of the Boolector SMT solver [20]. We selected 7 high-level hardware specifications from the HLSynth benchmark suite [21]. The same set of specifications is used in the SE3 paper [3]. We leveraged a high-level synthesis tool, Xilinx Vivado HLS, to generate 6 RTL designs with different timing for each specification. Thus, we obtained 105 pairs of equivalent designs for all those specifications. Moreover, we manually introduced errors into the designs to create 105 pairs of non-equivalent designs. All experiments are conducted on a Linux machine with a 3.2GHz processor. Each instance is executed on a single thread with a 4GiB of memory limit and a 7,200-second timeout.

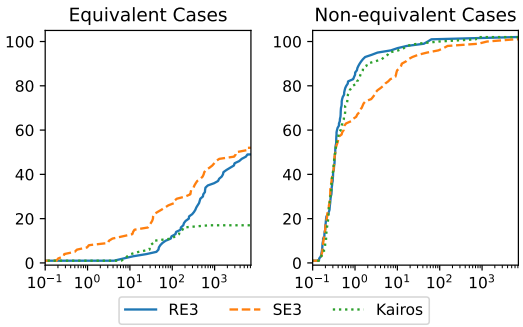


Fig. 3: Comparing the number of solved instances over time *(s)*.

Figure 3 compares RE3 against two existing non-cycle-accurate sequential equivalence checking algorithms, SE3 and Kairos [2], for their efficiency. On the equivalent benchmarks, RE3 and SE3 are

significantly faster than Kairos. RE3 is slightly slower than SE3 on these benchmarks. We believe this is because SE3 adopts a monolithic strategy that solves the scheduling function, the inductive invariant, and the reachability information at the same time. In comparison, RE3 discards some of the previous results when it starts a new iteration of the outer loop. However, such an overhead is justified by the fact that RE3 can find a refinement relation in addition to the verification result. On the non-equivalent benchmarks, both RE3 and Kairos are faster than SE3. It is because RE3 employs the simulation-based self-alignment mechanism, and Kairos always concentrates on a fixed alignment pattern throughout the model checking process.

We measured the size of an inductive invariant by counting the number of cubes (§II-A) within its logic formula. We collected all pairs of equivalent designs that can be solved by both RE3 and SE3. In Figure 4, each cross mark represents such a pair of designs. As plotted in the figure, the inductive invariants produced by RE3 are almost always more concise than those produced by SE3. The average size of the former is 10.3 cubes, whereas the average size of the latter is 33.1 cubes. In this regard, RE3 achieves a 69% reduction in size on average.

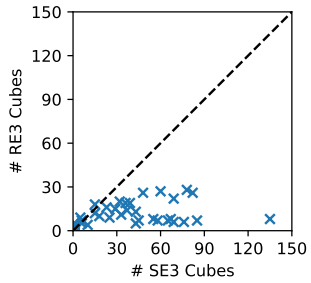


Fig. 4: Comparing the size of inductive invariants: RE3 vs. SE3.

B. Case Study

The following study showcases how a concise and explainable refinement relation can benefit IC design and verification. Recently, an abundant literature focuses on leveraging large language models (LLMs) for RTL code generation [22]–[25]. They rely on user feedback and prompt engineering to iteratively refine a design and improve its quality. Nevertheless, LLMs have an Achilles heel: their tendency to *hallucinate*. They can generate designs that seem plausible but contain subtle functional errors [24]. It is notoriously difficult to detect and fix such errors.

Existing work relies on simulation to determine the correctness of the generated design [22]–[25]. However, an error trace produced by a simulator does not reveal the root cause of the error. According to our experiments with *GPT-4o* [26], an LLM can be trapped in repeated interactions with a simulator because it is incapable of interpreting and reasoning about the error traces.

We propose a comprehensive method to detect and fix errors in LLM-generated designs. Starting from a reference design D_0, it incrementally applies a sequence of transformations with LLM to optimize the performance and incorporate new features. Suppose it has verified the correctness of D_i, the i-th version of the design, and obtained R_i, the refinement relation between D_0 and D_i. Given a new version D_{i+1}, our method traverses the relational transition diagram associated with R_i to check whether D_{i+1} violates any of the transitions in the diagram. Upon detecting a violation, the transition, its start state, and its end state are encapsulated in a prompt for the LLM to fix that error. When D_{i+1} is proved correct, the method computes a new refinement relation R_{i+1} and proceeds to the next iteration.

$x \leftarrow x \ll 1$
$y \leftarrow y \gg 1$
$z \leftarrow y[0]$ **?** $z + x$ **:** z

(a) $multi_A$

$x \leftarrow y[1]$ **?** $x \ll 1$ **:** $x \ll 2$
$y \leftarrow y[1]$ **?** $y \gg 1$ **:** $y \gg 2$
$z \leftarrow y[0]$ **?** $z + x$ **:** z

(b) $multi_B$

if $y[3:1] = 000$ **then**
  $x \leftarrow x \ll 4;\ y \leftarrow y \gg 4$
else if $y[2:1] = 00$ **then**
  $x \leftarrow x \ll 3;\ y \leftarrow y \gg 3$
else if $y[1] = 0$ **then**
  $x \leftarrow x \ll 2;\ y \leftarrow y \gg 2$
else $x \leftarrow x \ll 1;\ y \leftarrow y \gg 1$
$z \leftarrow y[0]$ **?** $z + x$ **:** z

(c) $multi_C$ (expected)

if $y[3:0] = 0001$ **then**
  $x \leftarrow x \ll 1;\ y \leftarrow y \gg 1$
else if $y[3:0] = 0010$ **then**
  $x \leftarrow x \ll 2;\ y \leftarrow y \gg 2$
else if $y[3:0] = 0100$ **then**
  $x \leftarrow x \ll 3;\ y \leftarrow y \gg 3$
else if $y[3:0] = 1000$ **then**
  $x \leftarrow x \ll 4;\ y \leftarrow y \gg 4$
else $x \leftarrow x \ll 1;\ y \leftarrow y \gg 1$
$z \leftarrow y[0]$ **?** $z + x$ **:** z

(d) $multi_D$ (incorrect)

Fig. 5: Loop bodies of 4 implementations of `multi_16bit` benchmark.

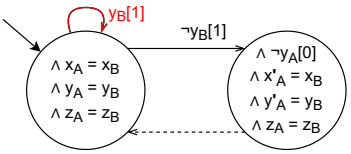


Fig. 6: A relational transition diagram for $multi_A$ and $multi_B$. x'_A and y'_A denote the next states of x_A and y_A, respectively.

We use `multi_16bit` from the *RTLLM* benchmark suite [23] to illustrate the above-mentioned method. Fig. 5(a) displays the simplified loop body of the reference design given by *RTLLM*. In every clock cycle, the multiplicand x is shifted to the left by 1 bit. If the last bit of the multiplier y is 1, the product z accumulates the shifted value of x. In practice, one can use a high-level specification, a certified initial implementation, or a transactional-level model (TLM) as the reference design. Suppose the designer decides to improve the performance of the design by looking into the next 2 bits instead of 1. Specifically, both x and y will be shifted twice if $y[1]$ is 0. Fig. 5(b) displays the pseudocode of the loop body generated by *GPT-4o*. Notice that this transformation is non-cycle-accurate since the timing of $multi_B$ is dependent on input values. We run RE3 on $multi_A$ and $multi_B$, yielding a concise refinement relation R_B. The corresponding relational transition diagram is shown in Fig. 6.

Suppose the designer wants to improve the performance further by looking into the next 4 bits. The two operands will be shifted multiple times if there is a consecutive sequence of zeros in y. Fig. 5(c) displays a correct implementation. Nevertheless, the implementation generated by *GPT-4o* (Fig. 5(d)) is incorrect and cannot pass the testbench provided by *RTLLM*. We attempted to prompt the LLM with the error traces, but it could not identify the root cause from this information. Hence, we traverse the relational transition diagram of R_B to check whether $multi_D$ violates any of those transitions. As shown in Fig. 6, the transition marked in *red* is violated. In a nutshell, $multi_D$ overlooked that the shifting behavior should not depend on the value of $y[0]$. We prompted *GPT-4o* with the transition and the associated states, which enabled the LLM to detect and fix the error.

V. Conclusion

In this paper, we present a new method for non-cycle-accurate sequential equivalence checking. This method can automatically synthesize a refinement relation during its execution. We illustrate potential applications of the refinement relation in the IC design and verification flow. We plan to explore additional use cases and integrate the refinement relation into EDA tools in the future.

REFERENCES

[1] S. Dai, G. Liu, R. Zhao, and Z. Zhang, "Enabling adaptive loop pipelining in high-level synthesis," in *ACSSC 2017*, pp. 131–135.

[2] L. Piccolboni, G. Di Guglielmo, and L. P. Carloni, "Kairos: Incremental verification in high-level synthesis through latency-insensitive design," in *FMCAD 2019*.

[3] Y. Li, G. Zhao, Y. He, and H. Zhou, "Se3: Sequential equivalence checking for non-cycle-accurate design transformations," in *DAC 2023*.

[4] B. Godlin and O. Strichman, "Regression verification," in *DAC 2009*.

[5] R. Dureja and K. Y. Rozier, "Fuseic3: An algorithm for checking large design spaces," in *FMCAD 2017*.

[6] J. Shi and S. A. Seshia, "Lifting hardware models from implementations for verification," Master's thesis, EECS Department, University of California, Berkeley, 2022.

[7] P. Subramanyan, B.-Y. Huang, Y. Vizel, A. Gupta, and S. Malik, "Template-based parameterized synthesis of uniform instruction-level abstractions for soc verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 8, pp. 1692–1705, 2017.

[8] A. Bernardini, W. Ecker, and U. Schlichtmann, "Where formal verification can help in functional safety analysis," in *ICCAD 2016*.

[9] W. Fang, G. Hu, and H. Zhang, "r-map: Relating implementation and specification in hardware refinement checking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[10] B.-Y. Huang, H. Zhang, A. Gupta, and S. Malik, "Ilang: A modeling and verification platform for socs using instruction-level abstractions," in *TACAS 2019*.

[11] M. Abadi and L. Lamport, "The existence of refinement mappings," *Theoretical Computer Science*, vol. 82, no. 2, pp. 253–284, 1991.

[12] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "Parameter synthesis with ic3," in *FMCAD 2013*.

[13] A. Goel and K. Sakallah, "Model checking of verilog rtl using ic3 with syntax-guided abstraction," in *NASA Formal Methods 2019*.

[14] J. Daniel, A. Cimatti, A. Griggio, S. Tonetta, and S. Mover, "Infinite-state liveness-to-safety via implicit abstraction and well-founded relations," in *CAV 2016*.

[15] R. Shemer, A. Gurfinkel, S. Shoham, and Y. Vizel, "Property directed self composition," in *CAV 2019*.

[16] A. Shenoy, S. Prabhu, K. Madhukar, R. Shemer, and M. Srivas, "Automated property directed self composition," in *ATVA 2023*.

[17] A. Cimatti, A. Griggio, S. Mover, and S. Tonetta, "Ic3 modulo theories via implicit predicate abstraction," in *TACAS 2014*.

[18] A. R. Bradley, "Sat-based model checking without unrolling," in *VMCAI 2011*.

[19] N. Een, A. Mishchenko, and R. Brayton, "Efficient implementation of property directed reachability," in *FMCAD 2011*.

[20] A. Niemetz, M. Preiner, C. Wolf, and A. Biere, "Btor2, btormc and boolector 3.0," in *ICCAD 2018*. Springer, pp. 587–595.

[21] P. R. Panda and N. D. Dutt, "1995 high level synthesis design repository," in *ISSS 1995*, pp. 170–174.

[22] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *ICCAD 2023*.

[23] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *ASP-DAC 2024*, pp. 722–727.

[24] K. Xu, J. Sun, Y. Hu, X. Fang, W. Shan, X. Wang, and Z. Jiang, "Meic: Re-thinking rtl debug automation using llms," *ICCAD 2024*.

[25] X. Wang, G.-W. Wan, S.-Z. Wong, L. Zhang, T. Liu, Q. Tian, and J. Ye, "Chatcpu: An agile cpu design and verification platform with llm," in *DAC 2024*.

[26] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

High-throughput Point-Cloud Accelerator with Sparsity-aware Hierarchical Neighbor Voxel Search and Skipping

Yun-Chia Yu[†], Suraj PN Reddy[§†], Aryan Devrani[§†], Anirudh Srinivasan[†], Saianudeep Reddy Nayini[†], Sohyeon Kim[†], Sung-Joon Jang[*], Sang-Seol Lee[*], Mingu Kang[†]
[†]University of California, San Diego (UCSD), [*]Korea Electronics Technology Institute (KETI)
{yunchia, surajpnreddy, adevrani, a6srinivasan, snayini, sohyeon, mingu}@ucsd.edu,
{sjjang0626, sslee81}@keti.re.kr

***Abstract*—Point cloud-based 3D sparse convolution networks are widely employed to process voxel features efficiently. However, the irregularity of voxel sparsity poses significant challenges, leading to increased hardware complexity and inefficiencies. We propose an algorithm-hardware co-design for sparse 3D convolution. At the algorithm level, an on-the-fly threshold-based voxel skipping is adopted, enhancing efficiency. At the hardware level, a hierarchical 3-stage Voxel Search and Skipping is developed to systematically narrow down the non-zero search space, enhancing both performance and hardware utilization. We implemented the proposed accelerator in a 65 nm process to demonstrate a 77.7% reduction in delay compared to a baseline design, which does not support the proposed sparsity adaptations. The proposed system also achieved the 1.34× and 2.22× higher energy efficiency and throughput as compared to the state-of-arts.**

***Index Terms*—point-cloud, 3D object detection, accelerator, autonomous driving.**

I. Introduction

Autonomous driving systems use advanced machine perception to extract insights from sensor inputs, enabling recognition and tracking of road users. LiDAR-generated point cloud (PC) data [1] is crucial for 3D object detection, providing spatial information to understand the environment [2] [3]. Perception tasks typically operate at up to 100 frames per second for timely hazard detection. However, PC-based 3D object detection models such as VoxelNet [4] face challenges in real-time and low-power execution, especially in resource-limited setups, due to the extensive processing required for 3D point data.

Recently, the high sparsity of PC data (often exceeding 97%) has been leveraged to skip redundant computations, enhancing processing speed [5]. However, the irregular distribution of sparse data points complicates hardware implementation, particularly in identifying active inputs and corresponding weights. Furthermore, in conventional convolutional operations, non-zero elements increase due to dilation, reducing sparsity benefits. Sub-manifold convolutions [6] address this by producing non-zero outputs only when the receptive field's center point is non-zero, maintaining sparsity. However, this introduces additional hardware complexity in determining which inputs and weights to process or skip in real-time.

Previous works SECOND [7] stored all possible combinations of [input index–weight index–output index] in a hash table to facilitate the skipping of sparse pixels at the cost of brute-force large table and searching overhead. In contrast, SPADE [8] introduced a more advanced computing logic that eliminated the need for large hash table storage, thereby reducing memory overhead and improving area efficiency, albeit at the cost of increased computational complexity. However, SPADE focuses primarily on conventional 3D convolutions without discussing optimizations for sub-manifold convolutions.

We aim to address these challenges through a hardware and software co-optimization approach that employs dynamic and hierarchical voxel search and skipping mechanisms. This approach skips less critical voxel processing by evaluating their importance in real-time, all while maintaining minimal hardware overhead. Although this paper primarily focuses on 3D voxel-based models, the proposed hardware design is also compatible to pillar-based sparse approaches, demonstrating its versatility. The detailed contributions are as follows.

We propose a hierarchical voxel search mechanism that operates in three stages: (1) it initially identifies non-zero center voxels, and (2) subsequently searches for non-zero neighboring voxels within the receptive field surrounding the center voxel. These steps significantly reduce computation by focusing solely on non-zero center and neighboring voxels. However, brute-force searching can result in processing inefficiency, creating cycle bubbles when the searched coordinates are turned out to be zero. To address this, (3) we narrow down the search space by independently checking the x-, y-, and z-coordinates, further optimizing the process.

Building on the above hardware mechanism, we address the decrease in sparsity over layers due to the dilation by leveraging magnitude-based voxel skipping algorithm [9]. Although this approach does not reduce the overall count of active (non-zero) voxels, it identifies and selects key voxels essential for processing. To leverage this mechanism, we enhance the hardware with the capability to dynamically compute key

[§]Equal contribution

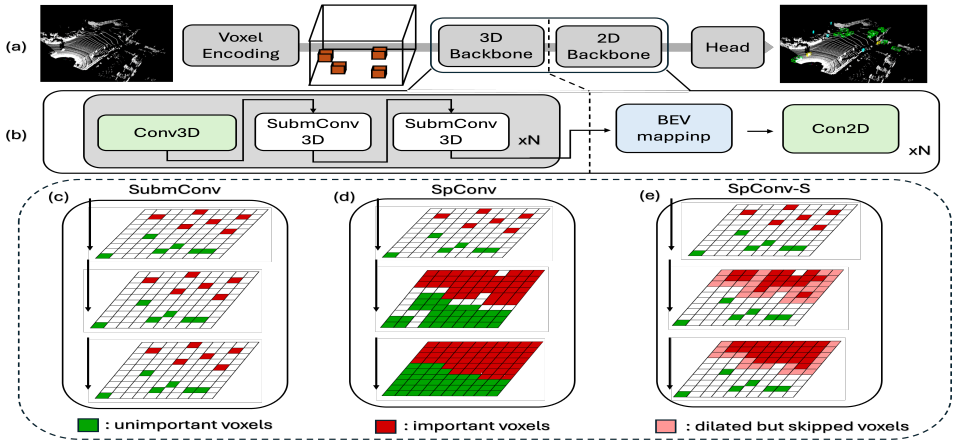


Fig. 1: Overview of voxel-based model: (a) model architecture, (b) Building blocks of 3D backbone. Comparison of output voxel set of various 3D sparse convolutions: (c) SubmConv, (d) SpConv, and (e) SpConv-S.

voxels on the fly during GEMM operations. These selected key voxels are then processed in subsequent layers, allowing the system to skip a significant portion of voxels and achieve a skipping ratio of approximately 65.7%.

To evaluate the efficacy of the proposed hardware and algorithm co-optimizations, we designed the accelerator using a 65 nm CMOS process and evaluated it with the Voxel R-CNN [10] architecture on the KITTI [11] dataset. Post-layout simulations indicate that the proposed architecture achieves 77.7% delay reductions.

II. Background

This section provides an overview of voxel-based models and their key computational kernels. Additionally, we discuss a recently introduced method aimed at minimizing the computational costs of these models.

A. Voxel-based Model

Voxel-based 3D object detection involves representing a point cloud (PC) in 3D space by discretizing it into a regular voxel grid and aggregating the point cloud information within each voxel to form voxel features, enabling efficient processing with 3D convolutional neural networks (3D-CNNs). Each voxel is defined as $V = \{v_{p_k}\} = \{(p_k, f_{p_k})\}$, where $p_k = (x_k, y_k, z_k)$ represents the 3D coordinates of the k-th voxel, and f_{p_k} denotes its aggregated feature vector. This voxelization facilitates the extraction of spatial features for the 3D object detection. It is noteworthy that voxel-based models exhibit significant sparsity in raw data (>98%), resulting in numerous inactive (all-zero) voxel positions due to the inherently sparse nature of 3D space. This paper aims to effectively exploit this opportunity to enhance computational efficiency.

B. Regular & Sub-manifold Sparse Convolution

Regular sparse convolution (SpConv) and sub-manifold convolution (SubmConv)are key kernels widely adopted in voxel-based models for 3D object detection. SpConv aggregates features within the receptive field of each input voxel, updating both existing voxels and activating new ones. Formally, the output feature of voxel is computed as:

$$f_p^{out} = \sum_{k \in \mathcal{K}} w_k \cdot f_{p+k}^{in}, \tag{1}$$

where f_p^{out} and f_{p+k}^{in} are the feature of output voxel v_p^{out} and neighboring input voxel v_{p+k}, respectively. w_k denotes the kernel weights, and $\mathcal{K}$ represents the set of kernel offsets. The output voxel set for SpConv is defined as:

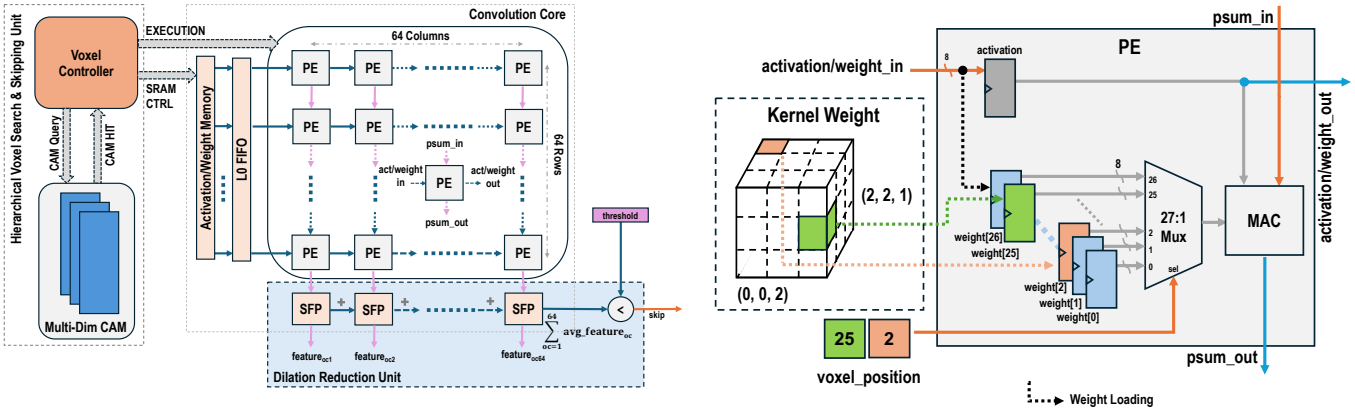
$$\mathcal{P}_{\text{out}} = \bigcup_{p \in \mathcal{P}_{\text{in}}} (p + \mathcal{K}), \tag{2}$$

where $\mathcal{P}_{\text{in}}$ ($\mathcal{P}_{\text{out}}$) is the set of active input (output) voxels' coordinates. Here, for simplicity, the stride of "1" is assumed. This expands the receptive field by activating new voxels as shown in Fig. 1d, improving feature propagation but increasing computational cost as layer goes by. SubmConv addresses the computing cost increase issue in SpConv by restricting the output voxel set to the original input voxel set:

$$\mathcal{P}_{\text{out}} = \mathcal{P}_{\text{in}}, \tag{3}$$

Here, the SubmConv generates a non-zero output only when the center of the receptive field is non-zero (Fig. 1c). By doing so, no additional voxels are activated, and sparsity is preserved before and after this kernel. While SubmConv is employed to reduce computation cost, it typically results in degraded accuracy. To address this trade-off, many voxel-based models [10] often alternate between SubmConv and standard kernels, as illustrated in Fig. 1, to balance accuracy and computational efficiency.

Additionally, the nature of SubmConv, which produces output only when the center of the receptive field is non-zero, offers an opportunity to skip more computations. However, this also increases hardware complexity, as it necessitates checking the center position before performing the MAC operation for any position between the kernel and input voxel. This paper will further leverage this opportunity by proposing the low-complexity sparse-aware processor.



(a) (b)

Fig. 2: (a) System architecture & data path illustrating convolution core and the Hierarchical Voxel Search & Skipping Unit (HVSS), (b) PE design illustrating weight stationary flow with all kernel weights stored to support MAC for active voxels.

C. Magnitude-based Dynamic Voxel Skipping

Magnitude-based dynamic voxel skipping [9] enhances the efficiency of sparse convolution by introducing a mechanism to skip less important voxels, significantly reducing the computational cost. The importance score of each voxel is computed by the sigmoid of the mean of its features. Based on a pre-defined percentile threshold T_p, the input voxel set is divided into two subsets: the important voxel set $\mathcal{P}_{\text{im}}$ and the unimportant voxel set $\mathcal{P}_{\text{nim}}$. Formally:

$$\mathcal{P}_{\text{im}} = \{p \in \mathcal{P}_{\text{in}} \mid S_p(f_p) \geq T_p\}, \quad \mathcal{P}_{\text{nim}} = \mathcal{P}_{\text{in}} \setminus \mathcal{P}_{\text{im}}, \quad (4)$$

where $S_p(f_p)$ is the importance score of voxel v_p in terms of percentile, e.g., high percentile with higher score. For SpConv with voxel skipping (SpConv-S), the SpConv is performed for the $\mathcal{P}_{\text{im}}$ whereas SubmConv is performed on $\mathcal{P}_{\text{nim}}$ (Fig. 2e). The output voxel set includes all unimportant voxels and the neighborhoods of important voxels, defined as:

$$\mathcal{P}_{\text{out}} = \mathcal{P}_{\text{nim}} \cup \bigcup_{p \in \mathcal{P}_{\text{im}}} (p + \mathcal{K}), \quad (5)$$

This indicates that the only $\mathcal{P}_{\text{im}}$ creates dilation while $\mathcal{P}_{\text{nim}}$ maintains its original receptive field. This method optimally balances accuracy and computational costs based on their importance. This paper further explores how to implement the above algorithmic innovation with low-cost hardware support.

III. Proposed System

This section introduces hardware and software co-optimization approach that employs dynamic and hierarchical voxel search and skipping mechanisms to boost the speed and energy efficiency.

A. Architecture Overview

The proposed architecture includes 1) *convolution core*, which performs GEMM operations and 2) *Hierarchical Voxel Search & Skipping (HVSS)* Unit, which provides the sparse-adaptive computing capability to boost the performance. The convolution core features a 2D systolic array optimized for 3D convolution as shown in Fig. 2a. While it is flexible, we choose 64×64 array size in this work. Each Processing Element (PE) performs Multiply-Accumulate (MAC) operations for 8-bit activation and weight data. Activation and weight data are loaded into on-chip SRAM, with an L0 FIFO managing retrieval during both the weight-loading into the PE array and execution stages. The architecture adopts a widely-adopted weight-stationary dataflow, where weights are stored in the register of PE as illustrated in Fig. 2b. In the proposed design, each PE has 27 registers to hold all the $3\times3\times3$ weight kernel elements for a specific input and output channel. Within the 64×64 PE array, activation data flows horizontally from the west to east, while partial sums (psums) move vertically down through each column. While the input channel index is mapped on the row index, each column represents an output channel, allowing concurrent processing across multiple channels and ensuring high computational throughput.

At the bottom of each column, a single row of Special Function Processors (SFPs) accumulates psums generated by the corresponding PE column. By delivering psums from non-zero voxel features to the SFP consecutively, intermediate psum storage in memory is eliminated. This enables the SFP to perform on-the-fly accumulation of psums and directly store the computed output feature in the output SRAM. Unlike the brute-force method, which requires 27 times of writing into psum memory for the computation of single output feature, our approach significantly reduces memory access by computing the output feature on the fly.

While this work focuses on the voxel-based approaches, the proposed hardware accelerator is also fully compatible with pillar-based models, offering flexibility for diverse point-cloud processing tasks. For pillar-based models, which rely on 2D convolution, each PE is configured to store nine weights, matching the kernel size of 3×3. The remaining registers can be utilized to store the kernels for multiple channels to process in the time interleaved fashion.

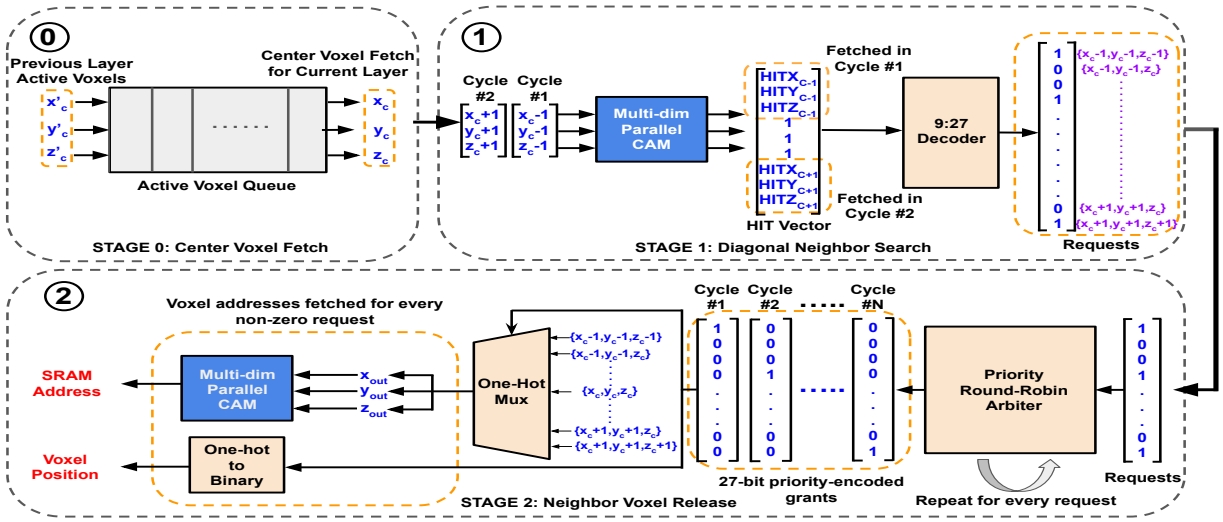


Fig. 3: 3-stage control flow for the Hierarchical Voxel Search & Skipping (HVSS) unit (all control signals are generated by the voxel controller, which is not shown for simplicity).

B. Hierarchical Voxel Search & Skipping (HVSS) Unit

The proposed architecture leverages the inherent sparsity in point cloud models, which often exhibits up to $> 98\%$. This is achieved through the HVSS unit, which integrates a voxel control unit, *Multi-Dim Parallel CAM*, and other auxiliary logic. The overall control flow of the unit can be broken down into the following key stages:

• **STAGE-0: Center Voxel Fetch:** For SubmConv, the output is generated only if the center voxel exists. Rather than brute-forcely storing and computing zeroes for inactive voxel positions, we maintain an *Active Voxel Queue*, which is filled up during the computation of previous layer. This buffer queue contains the coordinates and physical memory addresses of active voxels. By doing so, only the active voxels are stored in the memory, saving the storage space dramatically. For computation, we bring the non-zero coordinates (x_c, y_c, z_c) directly from the active voxel queue to selectively compute for only non-zero center voxels, saving compute cycles significantly.

• **STAGE-1: Diagonal Neighbor Search:** Around the center voxel at (x_c, y_c, z_c), the neighboring 26 voxels in 3D space need to be used for the computation to generate the output. However, due to the inherent high sparsity, many of these neighboring voxels remain inactive (zero). For example, on average, only about 50% of the neighboring voxels are active in the KITTI [11] dataset. This indicates a great opportunity to further reduce the energy and cycle count by leveraging this sparsity.

To exploit this, it is necessary to search for all the neighboring active voxels using CAMs, which requires a total of 26 cycles. However, due to high sparsity, around 50% of these searches will be a miss, indicating an inactive neighbor voxel, thereby wasting valuable search cycles. Such misses will create a worst-case of 26 bubbles in computing cycles for the convolution core, degrading the overall utilization.

To minimize these search bubble cycles, we propose a *Multi-Dim Parallel CAM* for the x-, y-, z- parallel search. The neighbor search begins with obtaining two diagonally opposite corner coordinates - $(x_c - 1, y_c - 1, z_c - 1)$ & $(x_c + 1, y_c + 1, z_c + 1)$ by subtracting and adding "1" from the center coordinate obtained from STAGE-0. Then, $(x_c - 1)$ and $(x_c + 1)$ are searched from the CAM-X for two consecutive cycles (see Fig. 3 Stage-① and Fig. 4). This process confirms whether there is any active voxel whose x-coordinate is either $(x_c - 1)$ and $(x_c + 1)$. Note that we do not perform the search for x_c as the search result is guaranteed to match due to the presence of the center voxel. Similar operations are performed in parallel for CAM-Y and CAM-Z as well, enabling all three CAM operations to execute simultaneously. This diagonal neighbor search approach drastically reduces the number of search bubble cycles from a worst-case of 26 (where every neighbor is inactive) to merely two.

Through above operations, the 9-bit HIT vector (hit_x/y/z[2:0]) is generated by CAMs, which gets passed to the 9:27 decoder (see Fig.3 Stage-①), generating a 27-bit *Requests* vector. Here, each bit corresponds to one of the 27 coordinates in 3D space, with a "1" indicating a potentially active voxel in the corresponding position.

• **STAGE-2: Neighbor Voxel Release:**

Suppose the 27-bit *Requests* vector contains N "1"s. The round-robin arbiter will then generate N one-hot encoded 27-bit vectors in N consecutive cycles (see Fig. 3 Stage-②). The one-hot mux converts these 27-bit vectors (*Grants*) into x-, y-, z- coordinates which are used to fetch from the multi-dim parallel CAM. If the CAM indicates a complete "match", the voxel is now guaranteed to be active (non-zero). Once the active voxel neighbors are identified, each voxel's feature contents pointed by the output SRAM address, as well as the voxel position is passed to the convolution core for computation of the current active voxel with corresponding weight out of 27 registers (see Fig. 2b). By pipelining Stages-

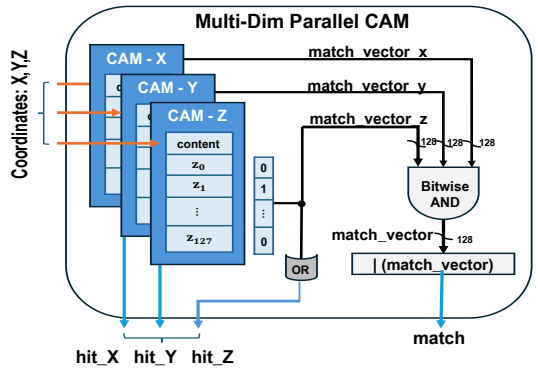


Fig. 4: Multi Dimension Parallel CAM - A wrapper integrating the three dedicated CAMs to store non-zero voxel coordinates.

① and ②, the search takes a total N cycles, which is significantly lower than 26 as will be shown in Section IV.

C. On-the-fly Dilation Reduction

While the original voxel skipping algorithm [9] introduced in Section II-C was designed to demarcate important and redundant voxels based on their percentile score among all voxels, it has a significant limitation: it does not allow for on-the-fly separation. Specifically, the decision about a voxel's importance cannot be made until all elements are processed, as sorting is required to rank them in magnitude order. To address this limitation, we modified the algorithm to be more hardware-friendly by using a comparison with an absolute-magnitude threshold (T) instead, as follows:

$$\mathcal{P}_{\text{im}} = \{p \in \mathcal{P}_{\text{in}} \mid S(f_p) \geq T\}, \quad \mathcal{P}_{\text{nim}} = \mathcal{P}_{\text{in}} \setminus \mathcal{P}_{\text{im}}, \qquad (6)$$

where $S(f_p)$ is the absolute importance score computed as the sigmoid of the mean of its feature. The value of T is determined from the training data by maximizing it within the range where accuracy remains consistent.

To capitalize on the above opportunity, the proposed architecture incorporates a threshold-based voxel skipping mechanism that dynamically evaluates the importance of computed output features. This is achieved by leveraging the Special Function Processors (SFPs) within the systolic array, which compute the absolute mean of the output feature across all output channels.

As illustrated in Fig. 2a, the mean computation is achieved through the horizontal movement of averaged output feature across the SFPs. The final mean value is fetched at the output of the last SFP. Note that the sigmoid function is monotonic. Therefore, instead of implementing hardware to compute the sigmoid directly, we compare the computed average with the pre-stored sigmoid$^{-1}(T)$ using an on-chip comparator.

By performing the mean calculation during the convolution process itself, the architecture avoids the additional cost of fetching psums from memory and computing the mean separately, thereby saving cycles and enhancing overall efficiency. If the computed mean falls below the threshold, the corresponding output feature is skipped for the subsequent layer. This dynamic filtering approach mitigates the effects of dilation, maintaining sparsity and improving the efficiency of subsequent processing stages.

IV. Evaluations

A. Evaluation Setup

To evaluate the efficacy of the proposed techniques, including hierarchical neighbor search and on-the-fly dilation reduction, we select a representative voxel-based model, Voxel R-CNN [10], as the baseline. Furthermore, we apply the threshold-based voxel skipping algorithm to this model, denoted as Voxel R-CNN-S, to facilitate comparisons with the vanilla model. For this study, we utilize the KITTI [11] dataset as the benchmark. To access the performance of proposed hierarchical neighbor voxel searching (HierS), we choose brute-force neighbor voxel searching (BFS) as baseline, where each center voxel will perform searching to all of the voxels within its receptive window.

Our design, including the proposed Hierarchical Voxel Search and Skipping Unit (HVSS) and the systolic array-based convolution core, was implemented and evaluated using a comprehensive RTL-to-GDSII flow. The RTL design was synthesized and placed-and-routed (PnR) using a 65 nm CMOS process with a general-purpose (GP) standard cell library. The implementation achieved timing closure at a 1 GHz clock frequency. To model hardware behavior, a cycle-accurate simulator was developed to calculate cycle counts, read/write operations on the on-chip SRAM, and search operations in the CAM. To evaluate power consumption, we performed post-layout power analysis using a value change dump (VCD) file, which captures realistic toggling behavior based on simulated data flow patterns. For SRAM memory, we use a Memory Compiler with ARM High density 65 nm GP single-port SRAM version, while the 65 nm CAM search energy and speed are derived from [12]. Key performance metrics, including power, performance, & area (PPA), are analyzed and compared with existing state-of-the-art accelerators, highlighting the efficiency of our proposed architecture.

B. Evaluation Results

• **Threshold-based voxel skipping ratio analysis:** The layer-wise analysis of number of important voxels within the 3D backbone of Voxel R-CNN & Voxel RCNN-S is shown in Fig. 5a. Voxel R-CNN-S effectively skips 65.7% of the voxels on average while achieving a slight accuracy improvement of 0.48%, significantly improving computational efficiency.

• **Voxel neighbor search counts reduction analysis:** The Layer-wise analysis of average neighbor search counts reduction is shown in Fig. 5b. Compared to BFS, HierS achives 5.16 & 6.34 search counts reduction in Voxel R-CNN & Voxel R-CNN-S, respectively, with minimal 0.71% area overhead and 0.014% power overhead.

• **Energy, area, and throughput analysis** The normalized energy efficiency and throughput analysis are illustrated in Fig. 5c. Compared to BFS, HierS achieves a 1.20× and 1.24× improvement in energy efficiency and throughput for Voxel R-CNN and Voxel R-CNN-S, respectively. Thanks to the

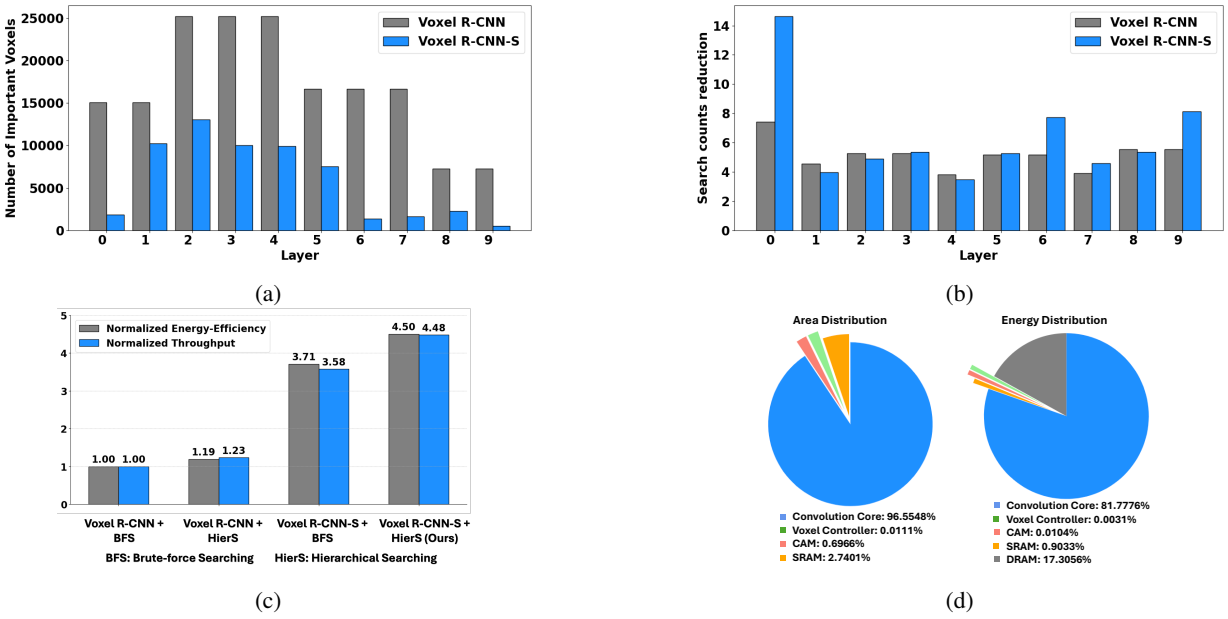


Fig. 5: (a) Layer-wise analysis of number of important voxels within the 3D backbone of Voxel R-CNN & Voxel R-CNN-S, (b) Layer-wise analysis of number of neighbor search counts reduction, (c) Comparison of normalized energy-efficiency & throughput, (d) Area & energy distribution of accelerator components.

TABLE I: Hardware Evaluation: Comparison with State-of-the-Art point-cloud accelerators.

	PointAcc	SPADE	Naive	Proposed (HW)	Proposed (HW+SW)
Cores	64x64=4096				
SRAM (KB)	776	654	156		
Area (mm^2)	15.70 (34.41*)	11.60 (38.28*)	45.28		
Frequency	1 GHz				
DRAM Bandwidth	HBM-2 256 GB/s				
Technology	40nm	32nm	65nm		
Peak Performance	8 TOPS				
Precision	N/A	A8-W8-Acc32			
GOPS/mm^2	**521.80 (132.08*)**	**706.2 (112.89*)**	**180.92**	**223.06**	**810.51**
GOPS/W	**N/A**	**1365.3 (299.29*)**	**336.1**	**400.3**	**1514**

* After scaling to a 65 nm technology by the rule in [13].

proposed on-the-fly dilation reduction technique, our hardware can effectively translation algorithmic computation efficiency into tangible actual hardware improvements. By leveraging the synergy between HierS with the on-the-fly dilation reduction, an overall 4.48× improvement is achieved compared with baseline setting. Fig. 5d breaks down the area and power consumption. It can be noticed that Voxel Controller and Multi-Dim CAM only covers less than 0.02% energy and 1% area overhead, respectively.

• **Hardware overhead:** Note that even with three separate CAMs in HVSS, there is no additional storage overhead, as each CAM stores exclusive content. Moreover, all the auxiliary logic outside of the CAMs (depicted in yellow in Fig. 3) consists of highly simplified circuitry, completing operations within a single cycle at 1 GHz, including a single CAM search.

• **Comparison with State-of-the-art:** Table I shows the comparison of proposed work with SOTA hardware accelerators. When matched in technology scaling factor [13], our work demonstrates 1.34× higher energy efficiency over SPADE, and and 2.22× higher throughput over PointAcc.

V. Conclusion

The sparsity in voxels presents a great opportunity for aggressive energy and delay efficiencies, yet the irregularity of voxel coordinates poses new challenges for the efficient processing of 3D Point Cloud perceptrons. Our hardware-algorithm co-optimized accelerator addresses these challenges by leveraging the inherent sparsity of voxels and proactively enhancing it through on-the-fly dilation reduction, enabling the skipping of additional computations. The proposed HVSS mechanism is seamlessly integrated to further reduce redundant computations, achieving high computational efficiency with minimal accuracy degradation and negligible energy and area overheads. This breakthrough facilitates real-time 3D object detection at the edge, empowering applications in autonomous driving, robotics, AR/VR, and beyond.

Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2022-0-00394, Development of AI semiconductor for level 4 self-driving based on functional safety) and by the PRISM center in JUMP 2.0, an SRC program sponsored by DARPA.

REFERENCES

[1] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2021.

[2] B. Li, "3d fully convolutional network for vehicle detection in point cloud," 2017. [Online]. Available: https://arxiv.org/abs/1611.08069

[3] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," 2017. [Online]. Available: https://arxiv.org/abs/1611.07759

[4] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," 2017. [Online]. Available: https://arxiv.org/abs/1711.06396

[5] H. Tang, Z. Liu, X. Li, Y. Lin, and S. Han, "Torchsparse: Efficient point cloud inference engine," 2022. [Online]. Available: https://arxiv.org/abs/2204.10319

[6] B. Graham and L. van der Maaten, "Submanifold sparse convolutional networks," 2017. [Online]. Available: https://arxiv.org/abs/1706.01307

[7] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/10/3337

[8] M. Lee, S. Park, H. Kim, M. Yoon, J. Lee, J. W. Choi, N. S. Kim, M. Kang, and J. Choi, "Spade: Sparse pillar-based 3d object detection accelerator for autonomous driving," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 454–467.

[9] J. Liu, Y. Chen, X. Ye, Z. Tian, X. Tan, and X. Qi, "Spatial pruned sparse convolution for efficient 3d object detection," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, ser. NIPS '22. Red Hook, NY, USA: Curran Associates Inc., 2024.

[10] J. Deng, S. Shi, P. Li, W. Zhou, Y. Zhang, and H. Li, "Voxel r-cnn: Towards high performance voxel-based 3d object detection," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, pp. 1201–1209, May 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16207

[11] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[12] K. Lee, W. Choi, and J. Park, "A 65-nm 0.6-fj/bit/search ternary content addressable memory using an adaptive match-line discharge," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 8, pp. 2574–2584, 2021.

[13] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm," *Integration*, vol. 58, pp. 74–81, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926017300755

ParGNN: A Scalable Graph Neural Network Training Framework on multi-GPUs

Junyu Gu[1,2,*], Shunde Li[1,2,*], Rongqiang Cao[1,2,*], Jue Wang[1,2,†], Zijian Wang[1,2], Zhiqiang Liang[1,2], Fang Liu[1,2], Shigang Li[3], Chunbao Zhou[1,2], Yangang Wang[1,2,†], Xuebin Chi[1,2]

[1]*Computer Network Information Center, Chinese Academy of Sciences, Beijing, China*

[2]*University of Chinese Academy of Sciences, Beijing, China*

[3]*School of Computer Science, Beijing University of Posts and Telecommunications, Beijing, China*

{*jygu,lishunde*}*@cnic.cn,* {*caorq,wangjue*}*@sccas.cn,* {*wangzj,zqliang*}*@cnic.cn, liufang@sccas.cn, shigangli.cs@gmail.com,* {*zhoucb,wangyg,chi*}*@sccas.cn*

***Abstract*—Full-batch Graph Neural Network (GNN) training is indispensable for interdisciplinary applications. Although full-batch training has advantages in convergence accuracy and speed, it still faces challenges such as severe load imbalance and high communication traffic overhead. In order to address these challenges, we propose ParGNN, an efficient full-batch training system for GNNs, which adopts a profiler-guided adaptive load balancing method along with graph over-partition to alleviate load imbalance. Based on the over-partition results, we present a subgraph pipeline algorithm to overlap communication and computation while maintaining the accuracy of GNN training. Extensive experiments demonstrate that ParGNN can not only obtain the highest accuracy but also reach the preset accuracy in the shortest time. In the end-to-end experiments performed on the four datasets, ParGNN outperforms the two state-of-the-art full-batch GNN systems, PipeGCN and DGL, achieving the highest speedup of 2.7× and 21.8× times respectively.**

***Index Terms*—Graph neural network, Full-batch distributed training, Load balancing, Computation and communication overlapping.**

I. Introduction

Graph Neural Networks (GNNs) [1], which are based on graph theory and the graph data structure, have witnessed rapid growth in recent years. GNNs have a wide range of applications in recommender systems [2], chip design [3], energy resource planning [4], etc. They take graphs as input and utilize the structured information of graphs for learning. The core computations in GNN training involve Sparse-Matrix Dense-Matrix Multiplication (SpMM) and Sampled Dense-Dense Matrix Multiplication (SDDMM) operations [5]. The sparse matrix represents the graph topology, while the dense matrices represent the vertex features and model weights. Many real-world datasets can be represented as large-scale graphs, which are difficult to handle on a single computing unit. As the size and complexity of graph data increase, there is growing interest in accelerating GNN training using multiple GPUs [6].

GNN training methods are divided into two categories: mini-batch training and full-batch training. Mini-batch training is a compromise for large-scale graph learning, where subgraphs are sampled and used as mini-batches in each training epoch. Full-batch training, which leverages the entire graph data, generally leads to more accurate and robust models [7].

*These authors contributed equally to work. †Corresponding authors.
ParGNN source code: https://github.com/echogujy/ParGNN

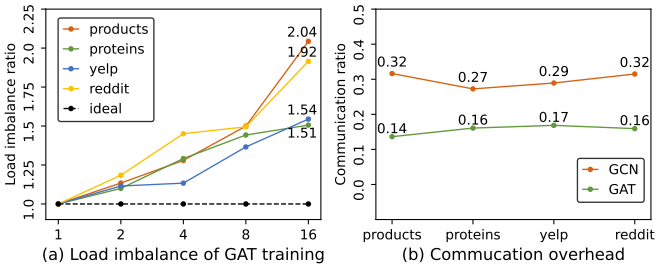


Fig. 1: Load imbalance and communication overhead of distrbuted GNN training.

However, full-batch training faces challenges such as severe load imbalance and high communication overhead when performed on multiple GPUs [2].

To alleviate the load imbalance of distributed GNN training, existing training systems commonly utilize graph partition tools like METIS [8] to divide the original graph into multiple subgraphs, which are scheduled to run on different GPUs. Those partition tools aim to minimize the number of cut edges while maintaining a balance in the number of vertices or edges between subgraphs. However, the performance of SPMM and SDDMM on GPUs is not only dependent on the number of nonzero elements and the dimension size but also on many other features, such as locality and skew (i.e. the mean, standard deviation, Gini index) [9]. Moreover, specific hardware acceleration implementations based on Tensor Core and others are emerging [10]. Despite these efforts, graph partition methods often fail to achieve true load balancing. Existing studies have not sufficiently focused on quantifying the degree of load balancing achieved. Specifically, they have overlooked an examination of the load imbalance ratio, which is defined as the longest running time divided by the average running time. Fig. 1(a) shows that the load imbalance ratio in METIS-based partition distributed GNN training increases with the number of subgraphs and can exceed 2 in real-world datasets.

The high communication traffic has been recognized as a formidable challenge in the distributed training of GNNs, and several studies have contributed to its reduction. Fig. 1(b) demonstrates that the communication time can occupy 14% to 32% of the training time. DistGNN [11] utilizes a delayed communication strategy to overlap communication with computation on multi-CPU platforms. PipeGCN [12] also hides the communication latency by using the delay strategy and

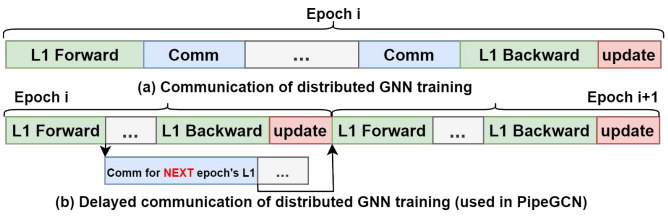


Fig. 2: Communication pattern of distributed GNN training.

breaking the dependency of computation and communication within the graphs on multi-GPU platforms. Although these methods alleviate the communication overhead, they reduce the convergence speed and accuracy of GNN training.

In this paper, we introduce ParGNN, a full-batch training framework for GNNs on multi-GPU platforms, aiming to tackle the challenges of processing large-scale graph data. The key of ParGNN is to adopt an over-partition approach and utilize profiler information in pre-execution to guide workload partition. The over-partition approach not only balances workload but also provides opportunities for true communication and computation overlap while maintaining the accuracy of GNN training. Our main contributions are summarized as follows:

- We propose a Profiler-Guided Adaptive Load Balancing (PGALB) algorithm in conjunction with a two-level graph partition to alleviate load imbalance.
- Based on the over-partition approach, we present a Subgraph Pipeline (SP) algorithm to overlap communication and computation in each GNN layer in order to accelerate GNN training.
- Extensive experiments of the ParGNN system demonstrate that ParGNN can not only obtain the highest accuracy but also reach the preset accuracy in the shortest time. ParGNN surpasses the two state-of-the-art full-batch GNN systems, PipeGCN and DGL, achieving the highest speedup of 2.7× and 21.8× times respectively.

II. Background

Graph Neural Network. Graph neural network is stacked by multiple layers of graph neural network layers. In each layer of graph neural network, the features on each vertex are updated by aggregating the features of theirs neighboring vertices along topological weights. Given the output feature matrix H^{l-1} on layer $l-1$, the forward propagation process of layer l is Equation (1), where W^l is the weight matrix, σ is the activation function, and A is a sparse topological weight matrix [13].

$$H^l = \sigma\left(AH^{l-1}W^l\right) \tag{1}$$

The most expensive operations are SpMM in $A \times H^{l-1}W^l$. For GNN models that update the topological weights at each layer (such as GAT [14]), they carry out the updates via SDDMM-like operations in Equation (2).

$$A^{'} = A \odot ((H^{l-1}W^l_{src}) \times (H^{l-1}W^l_{dst})^T) \tag{2}$$

where W^l_{src} and W^l_{dst} represent the network weights applied to the features of the source and destination vertices on the edges, respectively.

Distributed Full-batch GNN Training. To facilitate the training of full-batch GNNs on large-scale graphs, graph partition techniques are commonly employed. Specifically, the original graph is partitioned into multiple subgraphs, and each subgraph is allocated in GPU memory. This approach effectively alleviates memory constraints. By simultaneously training each subgraph, the overall training is significantly accelerated.

Communication Pattern. Since each GPU's subgraph contains only a portion of the entire graph's data, it is necessary to communicate and update the vertex feature information of the subgraph between two layers of GNN training. As shown in Fig. 2(a), during distributed GNN training, when finishing the calculation of one layer of GNN and starting the calculation of the next layer, the currently updated information is promptly communicated to other GPUs. In Fig. 2(b), the communication is delayed by one epoch, that is, the L1 update of epoch i is communicated and used for L1 of epoch $i+1$.

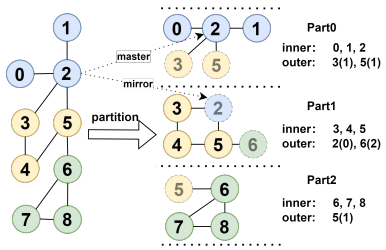


Fig. 3: Inner and outer vertices.

Fig. 3 shows an example where an original graph is divided into three subgraphs and vertices in each subgraph are divided into inner and outer vertices. The inner vertices, such as vertices ③, ④, and ⑤, are assigned to Part1 by the partitioner. The outer vertices, such as vertices ② and ⑥ in Part1, connect to the inner vertices but are assigned to other subgraphs. The outer vertices are copies of the inner vertices of other subgraphs. These copied inner vertices are also called the master vertices of the outer vertices in other subgraphs, while the outer vertices are named the mirrors of the master vertices. The inner vertices in each subgraph are updated through local calculations. However, the outer vertices (mirrors) are updated by receiving copies of the corresponding inner vertices (masters) from other subgraphs. For example, in subgraph Part1 in Fig. 3, the updates for inner vertices ③, ④, and ⑤ are obtained through aggregation calculations involving all vertices in the subgraph (vertices ②, ③, ④, and ⑤, and ⑥). Meanwhile, outer vertex ② receives its update from Part0, and outer vertex ⑥ receives its update from Part2.

III. System Design

ParGNN is a framework specifically designed for training large-scale graph neural networks on multiple GPUs. It leverages the powerful performance of GPU clusters to handle the growing amount of graph neural network data and is committed to addressing future scalability challenges. We name it Parallel Graph Neural Network (ParGNN). ParGNN uses full-batch processing to ensure the integrity of training information. At the same time, the entire graph is partitioned

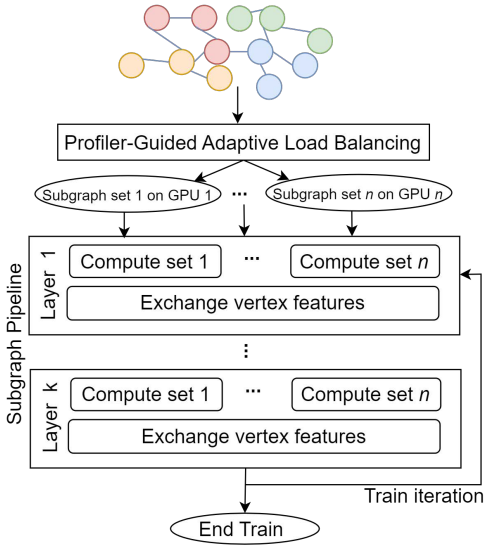


Fig. 4: Overview of ParGNN system.

and mapped into GPU memory. Different from other GNN training frameworks, ParGNN pays attention to the actual load of graph data on different GPUs and reconsiders the performance evaluation of sparse matrix operators required for GNN training on GPUs, aiming to enhance load balancing effects and improve training efficiency.

ParGNN is mainly composed of two parts, PGALB and SP. The purpose of PGALB is to allocate load-balanced data partitions for GPUs, while SP provides computation and communication overlap when training partitioned graph data after allocation.

Fig. 4 shows the workflow of ParGNN performing distributed full-batch GNN training on n GPUs. The original graph data is divided into n subgraph sets by PGALB's two-stage graph partitioning strategy. Each subgraph set contains several subgraphs and is mapped to one GPU. In the training stage, each GPU is responsible for training one subgraph set. In each GNN layer, ParGNN enables SP. This implements the pipeline overlapping of vertices features update computation and communication between subgraph sets of different GPUs. After completing the forward and backward computation of k layers of GNN in one epoch, ParGNN will check whether to continue iterative calculation or end training according to preset training end conditions, such as reaching the specified number of epochs or loss value.

IV. Profiler-Guided Adaptive Load Balancing

The sparsity of graph data causes a significant load imbalance during the execution process of distributed GNN. To alleviate this severe load imbalance, it is essential to partition the original graph data effectively, ensuring that computation tasks are distributed as evenly as possible among different computing units. However, as mentioned in Section 1, the method of evaluating computational load based on the number of vertices or edges is crude. The computational performance of sparse matrices largely depends on the sparsity of vertices and the density of memory access. Therefore, we propose a

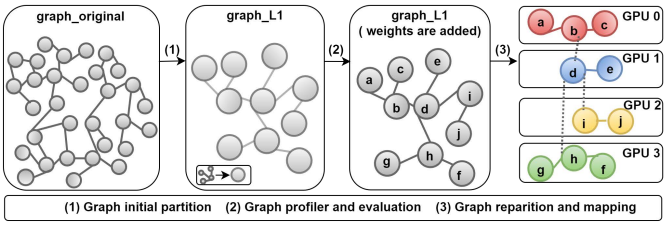


Fig. 5: Workflow of PGALB.

novel profiler-guided two-level partition approach guided to achieve adaptive load balancing in distributed GNN training.

The workflow of two-level adaptive load balancing when used for the data partition is described in Fig. 5 with the following three steps:

(1) Graph initial partition. The raw graph data is processed to initialize the data graph (graph_original) for subsequent partition and evaluation. Then, the graph_original is coarsened into a small-sized graph (graph_L1) to facilitate the evaluation of computational workload and load balancing. In graph_L1, each vertex represents a subgraph composed of multiple vertices from graph_original. The partition is based on the principle of having an approximately equal number of vertices and minimizing the number of cut edges to reduce inter-subgraph communication. This enables us to effectively control the scale of the graph, facilitating subsequent adjustments to load balancing.

(2) Graph profiler and evaluation. During the graph evaluation process, we evaluate all subgraphs of graph_L1 in a pre-executed manner to assess the computational load of the entire graph. We construct a GNN model that contains various dense and sparse operations that need to be performed during real training. By sequentially executing the model's forward calculation process for each subgraph and accurately recording the actual time for each execution, we can measure the computational load of each subgraph. Then, we assign weights to each vertex and edge in graph_L1 based on the calculated time and the number of cut edges.

(3) Graph reparition and mapping. We partition the subgraphs to each GPU based on the subgraph weights on graph_L1. The number of partitions is equal to the number of GPUs. The goal of partitioning graph_L1 is to achieve as much load balancing of computational tasks as possible across the partitions. In addition, the weights of edges between vertices are also considered during the mapping process. By minimizing the number of cutting edges, data communication between partitions can be effectively reduced. Each GPU is assigned multiple subgraphs (vertices in graph_L1). Thus, we have completed the data preparation for each GPU and can proceed with the model training process. During the training phase, for each GPU, we sequentially train these subgraphs.

By employing the above workflow, we can effectively reduce the load imbalance during distributed GNN training. We can achieve the optimal utilization of computational resources to enhance system efficiency and performance. The frameworks using one time graph partition have certain drawbacks. In case of a slight change in the original graph or a change in the number of underlying hardware processing units, they

must carry out a new partition of the original graph, and this process is very time consuming. Our strategy is to carry out large graph partitioning with time-consuming only once in the initial partition. The repartition and mapping step acts on the small graph_L1, and its actual consumption can be neglected. Meanwhile, for the computational load evaluation of each subgraph (vertex in graph_L1), it can be reassigned according to the actual situation during training and adapted to find a more balanced subgraph-GPU mapping relationship. In addition, it can be quickly adjusted according to the changes in the computing resources needed. Specifically, our algorithm is more suitable for continuous learning of GNNs.

V. Subgraph Pipeline Algorithm

ParGNN adjusts the workload between GPUs by allocating multiple subgraphs on one GPU, providing the opportunity to overlap computation and communication. However, two challenges need to be solved: (1) the number of subgraphs between different GPUs is inconsistent; (2) there is temporal dependence between subgraphs.

We assume GPU i has $\mathbf{k}_i$ subgraphs. And $\mathbf{G}_{ij}$ represents the j-th subgraph on GPU i, which adopts a compressed storage format. Each subgraph triggers a communication operation after its local computation is completed. The number of subgraphs on each GPU is different, so the number of communications on different GPUs is also different. The temporal dependence between subgraphs is manifested as follows: after a subgraph $\mathbf{G}_{ij}$ completes its local computation and updates the inner vertices, it needs to send the features of master vertices to the mirrors of other subgraphs. At the same time, the outer vertices need to wait for the reception from remote master vertices which have completed the update and sent out. However, the subgraphs on each GPU are computed sequentially. The master vertices of subgraph $\mathbf{G}_{ij}$ need to be sent to the subgraph $\mathbf{G}_{i'j'}(i' \neq i, j' \neq j)$ on other GPUs, but the $\mathbf{G}_{i'j'}$ may not be in the process of execution. It might have been executed($j' < j$) or still be waiting to be executed($j' > j$).

SP algorithm designs a structure that separates the communication sending and update retrieving operations of subgraph features to uniformly solve the above two challenges. For subgraph $\mathbf{G}_{ij}$, its corresponding set of outer vertices is $\mathbf{V}^o_{ij}$. The set of outer vertices that the GPU i wants to retrieve is $\mathbf{V}^o_i = \mathbf{V}^o_{i0} \cup ... \cup \mathbf{V}^o_{i\mathbf{k}_i-1}$. SP constructs a feature receiving buffer $\mathbf{B}_i$ based on the outer vertex set of all subgraphs $\mathbf{V}^o_i$ on the current GPU i for receiving all vertex features from other GPUs. Although all communications use the same receiving buffer $\mathbf{B}_i$, each communication fills a different part of the buffer. The all2allv primitive is used to implement irregular peer-to-peer communication between GPUs. After all communications are completed, each subgraph $\mathbf{G}_{ij}$ retrieves its outer vertices features from buffer $\mathbf{B}_i$ for update. $\mathbf{k}_{max}$ represents the maximum number of subgraphs across all GPUs, indicating the number of communications required to fill all the feature receiving buffers on all GPUs. It should be noted that due to some GPUs having less than $\mathbf{k}_{max}$ subgraphs, there

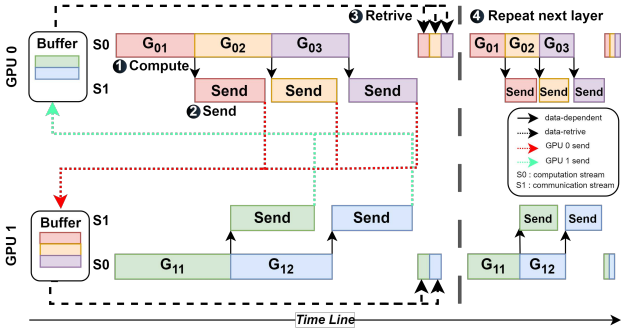


Fig. 6: Example for SP workflow. For subgraph G_{01} on GPU0, it first conducts local computations❶. After the computations are completed, the system immediately issues the asynchronous sending of the features of its master vertex❷. The sending process overlaps with the computation process of subsequent subgraphs. After the receiving buffer is full, G_{01} updates the outer vertices by retrieving the remote feature from the buffer❸. Then, G_{01} finishes all vertex updates and gets ready for the next computation❹.

are additional $\mathbf{k}_{max} - \mathbf{k}_i$ communication calls with keeping receiving but sending empty content.

Fig. 6 shows an example of SP execution when training on two GPUs. There are three subgraphs on GPU 0 and two subgraphs on GPU 1. SP utilizes the multi-stream execution feature on the GPU to configure two CUDA streams for each GPU during computation. There are the computation stream and the communication stream for computation and communication tasks respectively. The computation and asynchronous communication calls of the three subgraphs on GPU 0 are sequentially interleaved. Under the control of the CUDA stream, the communication time of $\mathbf{G}_{01}$ and $\mathbf{G}_{02}$ can be overlapped by the computation time of $\mathbf{G}_{02}$ and $\mathbf{G}_{03}$. The data on GPU 0 is communicated to the receiving buffer on GPU 1. Similarly, the data that GPU 0 wants is also communicated to the local buffer. After the buffer is full (that is, all the desired feature data has been received), the three subgraphs sequentially retrieve the updated features of the outer vertices from the buffer and start the computation of the next network layer.

The intermediate variables $\mathbf{F'}_{ij}$ and $\mathbf{F''}_{ij}$ are introduced to represent the computation and update process of the vertex features $\mathbf{F}_{ij}$ for subgraph $\mathbf{G}_{ij}$. The forward propagation execution process of SP is shown in Equation (3) and (4).

$$\mathbf{F'}_{ij} = \mathrm{ComputeGraph}(\mathbf{G}_{ij}, \mathbf{F}_{ij}) \tag{3}$$

$$\mathbf{F''}_{ij} = \mathrm{UpdateGraph}(\mathbf{B}_i, \mathbf{F'}_{ij}) \tag{4}$$

Since Equation (4) is an indexing filling process, i.e., $\frac{\partial(\mathbf{F''}_{ij})}{\partial(\mathbf{F'}_{ij})} = 1$. According to the chain rule for differentiation [15], the backward propagation for calculating the gradient of the loss with respect to $\mathbf{F}_{ij}$ is as shown in Equation (5).

$$\begin{aligned}\frac{\partial(\mathrm{loss})}{\partial(\mathbf{F}_{ij})} &= \frac{\partial(\mathrm{loss})}{\partial(\mathbf{F''_{ij}})} \cdot \frac{\partial(\mathbf{F''}_{ij})}{\partial(\mathbf{F'}_{ij})} \cdot \frac{\partial(\mathbf{F'}_{ij})}{\partial(\mathbf{F}_{ij})} \\ &= \frac{\partial(\mathrm{loss})}{\partial(\mathbf{F'}_{ij})} \cdot 1 \cdot \frac{\partial(\mathbf{F'}_{ij})}{\partial(\mathbf{F}_{ij})} \\ &= \frac{\partial(\mathrm{loss})}{\partial(\mathbf{F'}_{ij})} \cdot \frac{\partial(\mathrm{ComputeGraph}(\mathbf{G}_{ij}, \mathbf{F}_{ij}))}{\partial(\mathbf{F}_{ij})}\end{aligned} \tag{5}$$

The SP algorithm provides a feature computation and communication overlap pipeline for distributed GNN training. Note that although the SP algorithm is derived based on the characteristic of ParGNN storing multiple subgraphs on one GPU, the architecture that separates communication sending and feature retrieval can be easily extended to the common case where there is only a single subgraph on a GPU. In this case, only row splitting of the topological adjacency matrix corresponding to the subgraph is needed, and then the asynchronous communication overlap ability provided by the SP algorithm can be used.

VI. Evaluation

Experiment environment. We conduct experiments on DGX-H100 server. Which is equipped with an Intel Xeon Platinum 8358P 48-core processor, eight NVIDIA H100 80 GB GPUs with NVLINK connect, and 2048 GB host memory. We carry out the experiments using PyTorch 2.3.1 [16] and DGL 2.4.0 [5] with cuda-12.1.

TABLE I: Dataset information.

Dataset	products	proteins	yelp	reddit
Vertices	2,449,029	132,534	716,847	232,965
Edges	61,859,140	39,561,252	13,954,819	114,615,892
Features dim	100	128	300	602
Categories	47	112	100	41

Datasets. We use 4 public and authoritative real-world graph datasets. The details of these datasets are listed in Table I. Ogbn-Products (products) is an Amazon products co-purchasing network [17]. Ogbn-proteins (proteins) is a protein–protein association networks [18]. Yelp is a network dataset that classifies business types based on user reviews and social relationships [19]. Reddit is a community network of online posts based on user comments [20].

Models and Parameters. We use GCN [13] and GAT [14] as the experimental backbone networks for classification tasks in 300 epochs. Each model is stacked with 3 layers and the hidden dimension is 256. For the GAT model, the number of attention heads is 4. We train all datasets and models on 4 GPUs in 300 epochs with the following experimental parameters: Dropout [21] rate is 0.5, the Adam [22] optimizer with a learning rate of 0.001, weight decay is 0.0001, and the random seed is set to 2024.

A. Load Balancing Evaluation

Table II presents the load balancing performance comparisons between METIS and the PGALB method based on METIS' initial partitions. METIS uses the edge-cut partitioning algorithm to partition four datasets, and the number of subgraphs partitioned by PGALB is four times that of the number of GPUs. Compared to the execution time of the METIS partition, the increase in the execution time of load balancing adjustment using PGALB is within 1.5%, indicating that the increase in time for PGALB is negligible. At the same time, the PGALB algorithm reduces the load imbalance ratio from 6.59% to 43.38%. Since the graph processed in the graph profiler and repartition stages is relatively small in scale, PGALB has less additional execution time after the initial partition, enabling PGALB to perform rapid adaptive load balancing adjustments.

TABLE II: PGALB v.s. METIS. The upper part shows the partitioning time, while the lower part indicates the degree of load imbalance.

Dataset	products	proteins	yelp	reddit
METIS time(s)	36.07	17.73	4.81	26.22
PGALB time(s)	36.26	17.88	4.87	26.51
increase ratio(%)	0.52%	0.85%	1.33%	1.12%
Load imbalance evaluation				
METIS (%)	118.27%	126.75%	112.45%	145.98%
PGALB (%)	105.93%	102.31%	105.85%	102.59%
reduction ratio(%)	12.34%	24.45%	6.59%	43.39%

B. Distributed GNN System Evaluation

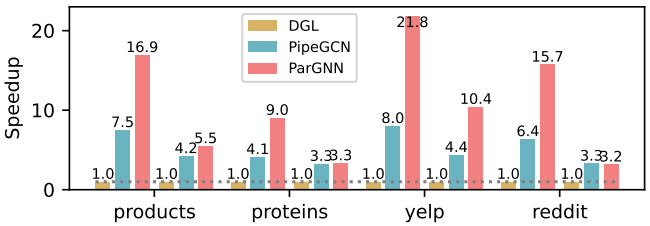


Fig. 7: Execution time speedup of three GNN systems for all datasets. In each dataset, the left three columns represent the speedup on the GCN model, and the right three columns are also for the GAT model.

We select two state-of-the-art distributed GNN training systems for comparison, namely DGL [5] (the most widely used industrial GNN system) and PipeGCN [12] (a system that achieves speedup through the delaying strategy of the communication pipeline).

We present the speedup of the average epoch training time of ParGNN and PipeGCN relative to DGL in Fig. 7. In general, the execution efficiency of ParGNN is better than both PipeGCN and DGL, achieving the highest speedup of 2.7× and 21.8× times respectively, while DGL has the poorest computing efficiency on all datasets. The acceleration effect of ParGNN on DGL is 9.0× to 21.8× on GCN and 3.2× to 10.4× on GAT. For PipeGCN, ParGNN can generally obtain about a 2× acceleration effect on GCN and similar speedup in GAT.

We present the training accuracy on products under three different systems in Fig. 8, which shows the variation of test accuracy with the increase in training epochs. The three systems are converging to a very similar accuracy on the two models. The training accuracy of the last 100 epochs is enlarged in the figure, and the highest accuracy reached

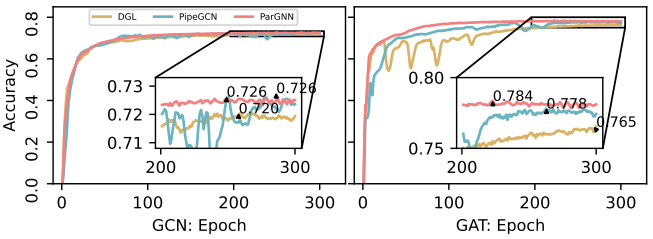


Fig. 8: Accuracy variation with epochs for the products dataset. The left subfigure shows the results of GCN, and the right one presents those of GAT.

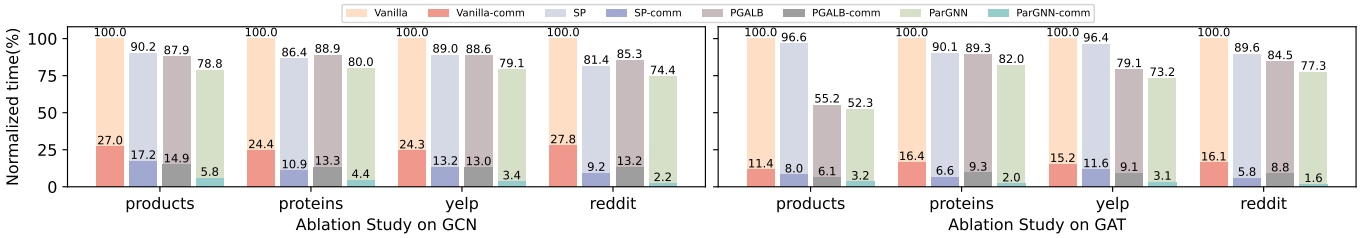


Fig. 9: Ablation study of ParCGNN for all datasets. The left and right subfigures respectively show the normalized time of GCN and GAT based on Vanilla average epoch time for each dataset, including average epoch and communication time.

by different systems is marked respectively. It shows that the difference between them is in a bound of 2.5%. This indicates that ParGNN can have a stable convergence effect like PiPeGCN and DGL.

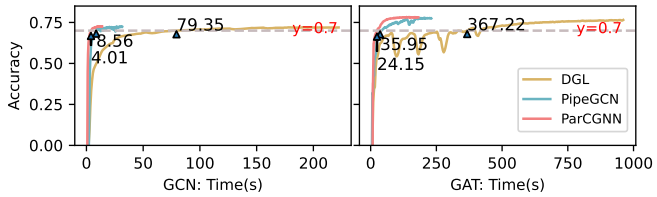


Fig. 10: Time-To-Accuracy for the products dataset. The left subfigure shows GCN's performance, and the right one presents GAT's.

We also provide the Time-To-Accuracy indicator. It can intuitively reflect the efficiency of the model in achieving the expected accuracy and offer a more comprehensive perspective for evaluating the system performance [23]. As shown in Fig. 10, we set the predetermined accuracy as 0.70 on the products dataset. In GCN, ParGNN reaches the target accuracy at 4.01 seconds. PipeGCN and DGL take 8.56 seconds and 79.35 seconds, respectively. In GAT, ParGNN is also the fastest, taking 24.15 seconds. DGL takes 35.95 seconds. Instead, PipeGCN takes the longest time of 367.22 seconds. ParGNN can converge to the predetermined accuracy the fastest on both models.

Overall, ParGNN can not only obtain the highest accuracy but also reach the preset accuracy in the shortest time. This indicates that ParGNN presents a more stable training performance compared to DGL and PipeGCN.

C. Ablation Evaluation of ParGNN

ParGNN is divided into the following four versions for the ablation experiment analysis to analyze the contributions of the two factors, namely PGALB and SP:

(a) ParGNN: This is the full-functional GNN training system that is optimized with both PGALB and SP.

(b) PGALB: This is the version of ParGNN where only the PGALB optimization is enabled.

(c) SP: This is the version of ParGNN where only the SP optimization is enabled.

(d) Vanilla: This is the version of the multi-subgraph distributed GNN training system after discarding both optimization methods.

We normalize the epoch execution time and communication time of each dataset on the specific model based on the execution results of Vanilla to better show the effects of different datasets on different models. The purpose is to fully present the performance improvement of different optimization methods in communication and overall efficiency, as shown in Fig. 9.

The performance improvement brought by PGALB is more concentrated on the GAT model. As stated in Equation (2), the computational load of GAT is greater than that of GCN. When the computational load ratio is heavier, the improvement in computational efficiency brought by load balancing is higher. In Fig. 1, it is shown that the computational time proportion during GAT training is on average 14.25% higher than that of GCN. In the ablation experiments, the average improvement(Vanilla $\rightarrow$ PGALB) in effect brought by PGALB's adaptive load balancing for GAT is 22.95%, while the average improvement in effect obtained on GCN is 12.31%.

The computation and communication overlap provided by SP is also very obvious. SP can reduce the time proportion in GCN from an average of 12.62% to 3.95%, and reduce the time proportion in GAT from an average of 7.99% to 2.49% (PGALB$\rightarrow$ParGNN). The reduction ratios of time are 68.70% and 68.86%, respectively. This gain is in line with the preset situation where each GPU is responsible for an average of four subgraphs. If the communication of three subgraphs is completely overlapped and the communication time for each subgraph is the same, the communication time will be reduced by approximately 75% in an ideal state. And the actual gain is already close to 70%, which shows the communication overlap effect of SP is good.

In general, ParGNN has a 20.0% to 25.6% improvement in performance on GCN and an 18.0% to 47.7% improvement in performance on GAT. Both PGALB and SP can effectively improve the performance of distributed GNN training.

VII. Conclusion

In this paper, we introduce ParGNN, a full-batch GNN training system for multi-GPU platforms. ParGNN adopts the PGALB algorithm with a two-level graph partition to reduce workload imbalance. ParGNN enables multiple subgraphs to be trained on the same GPU and employs the SP algorithm for overlapping computation and communication while maintaining accuracy, outperforming state-of-the-art frameworks such as DGL and PipeGCN.

Acknowledgment

This work is supported by the National Key Research & Development Program of China (Grant No. 2023YFB4502303), and the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDB0500103).

REFERENCES

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[2] H. Lin, M. Yan, X. Ye, D. Fan, S. Pan, W. Chen, and Y. Xie, "A comprehensive survey on distributed training of graph neural networks," *Proceedings of the IEEE*, 2023.

[3] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. M. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, J. Pak, A. Tong, K. Srinivasa, W. Hang, E. Tuncer, Q. V. Le, J. Laudon, R. Ho, R. Carpenter, and J. Dean, "A graph placement methodology for fast chip design," *Nature*, vol. 594, pp. 207–212, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235395490

[4] T. Yao, J. Wang, Y. Wang, P. Zhang, H. Cao, X. Chi, and M. Shi, "Very short-term forecasting of distributed pv power using gstann," *CSEE Journal of Power and Energy Systems*, 2022.

[5] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2020.

[6] D. Zheng, X. Song, C. Yang, D. LaSalle, and G. Karypis, "Distributed hybrid cpu and gpu training for graph neural networks on billion-scale graphs," 2022.

[7] Z. Jia, S. Lin, M. Gao, M. Zaharia, and A. Aiken, "Improving the accuracy, scalability, and performance of graph neural networks with roc," in *Proceedings of Machine Learning and Systems*, I. Dhillon, D. Papailiopoulos, and V. Sze, Eds., pp. 187–198.

[8] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, dec 1998.

[9] S. Yesil, A. Heidarshenas, A. Morrison, and J. Torrellas, "Wise: Predicting the performance of sparse matrix vector multiplication with machine learning," in *Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 329–341. [Online]. Available: https://doi.org/10.1145/3572848.3577506

[10] R. Fan, W. Wang, and X. Chu, "Dtc-spmm: Bridging the gap in accelerating general sparse matrix multiplication with tensor cores," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 253–267.

[11] V. Md, S. Misra, G. Ma, R. Mohanty, E. Georganas, A. Heinecke, D. Kalamkar, N. K. Ahmed, and S. Avancha, "Distgnn: Scalable distributed training for large-scale graph neural networks," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3480856

[12] C. Wan, Y. Li, C. R. Wolfe, A. Kyrillidis, N. S. Kim, and Y. Lin, "Pipegcn: Efficient full-graph training of graph convolutional networks with pipelined feature communication," 2022.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[14] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," *International Conference on Learning Representations*, 2018.

[15] N. Ketkar, J. Moolayil, N. Ketkar, and J. Moolayil, "Automatic differentiation in deep learning," *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, pp. 133–145, 2021.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[17] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 257–266. [Online]. Available: https://doi.org/10.1145/3292500.3330925

[18] D. Szklarczyk, A. L. Gable, D. Lyon, A. Junge, S. Wyder, J. Huerta-Cepas, M. Simonovic, N. T. Doncheva, J. H. Morris, P. Bork *et al.*, "String v11: protein–protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets," *Nucleic acids research*, vol. 47, no. D1, pp. D607–D613, 2019.

[19] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, "Graphsaint: Graph sampling based inductive learning method," *arXiv preprint arXiv:1907.04931*, 2019.

[20] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[21] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:6844431

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[23] C. Coleman, D. Kang, D. Narayanan, L. Nardi, T. Zhao, J. Zhang, P. Bailis, K. Olukotun, C. Ré, and M. Zaharia, "Analysis of the time-to-accuracy metric and entries in the dawnbench deep learning benchmark."

Energy-Efficient Large-Scale Vector Similarity Search in NAND-Flash via Hybrid Matching

Chih-Yu Hu[1§], Chi-Tse Huang[2§], Hao-Wei Chiang[2], Hsiang-Yun Cheng[3],
Po-Hao Tseng[4], Ming-Hsiu Lee[4], An-Yeu (Andy) Wu[2]
[1]*Department of Mechanical Engineering, National Taiwan University, Taipei, Taiwan*
[2]*Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan*
[3]*Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan*
[4]*Macronix International Co. Ltd, 16 Li-Hsin Road, Hsinchu Science Park, Taiwan*
b09502137@ntu.edu.tw, {rickhuang, jackson}@access.ee.ntu.edu.tw, hycheng@citi.sinica.edu.tw,
pohaotseng@mxic.com.tw, Ericlee@mxic.com.tw, andywu@ntu.edu.tw

***Abstract*—Vector similarity search (VSS) is crucial in many AI applications, such as few-shot learning (FSL) and approximate nearest-neighbor search (ANNS), but it demands significant memory capacity and incurs substantial energy costs for data transfers during large-scale comparisons. Various in-memory search technologies have been developed to improve energy efficiency, with NAND-based multi-bit content-addressable memory (MCAM) standing out as a promising solution for its high density and large capacity. MCAM can operate in exact-search (ES) mode, supporting only perfect matches with low energy cost, or in approximate-search (AS) mode, enabling flexible VSS. However, AS mode incurs significant energy waste when comparing queries with non-target stored vectors. To address this issue, we propose Hybrid-M, a 3D NAND-based in-memory VSS architecture that integrates both modes into a single hybrid matching process, using ES mode as a filter to reduce redundant searches for AS mode. We apply three techniques to optimize this integration: range encoding for multi-level cells (MLC) to enhance filtering, search voltage shifts to mitigate the impact on AS accuracy and reduce matching currents, and a filtering-aware training method to further improve reliability and energy efficiency. Results show that Hybrid-M achieves comparable accuracy while reducing energy consumption by 67% to 83% compared to MACM-based VSS using only AS mode, across various many-class FSL and ANNS workloads.**

I. Introduction

Vector similarity search (VSS), which compares a query with stored vectors using specific similarity metrics, is a cornerstone of many applications. It has become crucial in AI, where neural networks and machine learning models encode the semantics of images, videos, and documents as high-dimensional vectors—often called embeddings or features—and use their similarities for predictions. In practice, VSS is often scaled to accommodate large datasets. Prominent examples include many-class few-shot learning (FSL) [1], [2], which classifies unseen queries from numerous classes, and approximate nearest-neighbor search (ANNS) [3], [4], extensively used in recommendation systems and retrieval-augmented generation (RAG) [5] for efficient semantic searches.

Despite its pivotal role in modern AI applications, VSS often faces performance and energy-efficiency bottlenecks due to substantial memory demands and frequent data transfers [6], [7]. For instance, production-level recommendation systems, such as Microsoft Bing [8] and Alibaba e-commerce platforms [9], need tens of TB memory for ANNS with billions of 100+ dimensional vectors. The sheer volume of high-dimensional data makes VSS memory- and bandwidth-intensive, requiring significant energy efficiency optimizations.

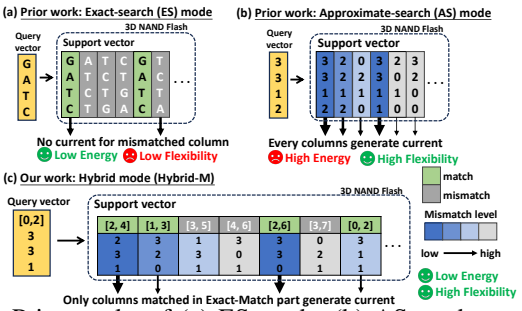


Fig. 1: Prior works of (a) ES mode, (b) AS mode, and (c) the proposed work with Hybrid-M.

Researchers have developed in-memory search (IMS) architectures using various memory technologies to tackle this challenge [10]–[12]. Given the significant memory demands of large-scale VSS, 3D NAND-based multi-bit content-addressable memory (MCAM) emerges as a promising solution due to its high density, large storage capacity, and support for highly parallel search operations. Studies have shown that 3D NAND-based MCAM, with appropriate search voltage settings, supports two modes: exact-search (ES) [13] and approximate-search (AS) [14]. The low-energy ES mode, depicted in Fig. 1(a), identifies only vectors identical to the query, while the AS mode, shown in Fig. 1(b), flexibly identifies vectors similar to the query, with currents correlating to their distance from the query. However, the AS mode incurs substantial energy waste during comparisons between the query and non-target vectors. Our analysis reveals that nearly 98.07-99.61% of energy is wasted in many-class FSL and ANNS scenarios. This trade-off between energy efficiency and search flexibility highlights the need for optimized search flows in 3D NAND-based IMS architectures to achieve energy-efficient large-scale VSS.

To this end, we propose Hybrid-M, a 3D NAND-based IMS architecture that seamlessly integrates ES and AS modes into a single hybrid matching process, as shown in Fig. 1(c). The

[§]Equal contribution

ES mode acts as a filter to reduce redundant searches for the AS mode. Unlike other IMS designs that use separate arrays for each mode [15], Hybrid-M leverages the serial-connection structure of 3D NAND flash to concatenate both modes within the same NAND string. By applying different search voltages to NAND cells associated with each mode, Hybrid-M enables searches in a single hybrid matching operation, significantly reducing latency and energy consumption.

Integrating these two modes within the same NAND string poses several design challenges. First, allocating a portion of NAND cells to the ES mode reduces the available storage for the AS mode, limiting the vector dimensions and potentially impacting accuracy. To avoid significantly compromising AS performance, only a limited number of NAND cells can be dedicated to the ES mode. Within this constraint, the filtering dimensions must be carefully selected and efficiently encoded to maximize the filtering rate. Second, the presence of the ES mode may alter the currents in both modes, which disrupts the correlation between current levels and the distance to the query.

In this paper, we apply four techniques to address these challenges. First, we select the representative dimension with the maximum inter-class variance and program it with ES mode, enabling filtering based on the L_∞ similarity metric [16]. Second, we design a range encoding scheme for multi-level cells (MLC) to improve the filtering rate with limited cells for ES mode. Third, to mitigate the impact of the hybrid-matching mechanism on AS reliability, we lower the search voltage. This adjustment also reduces the overall current in each NAND string, thereby lowering search energy. Finally, we introduce a filtering-aware training method that incorporates the hybrid-matching mechanism with filtering into the training process. This approach further improves VSS accuracy and energy efficiency, particularly for applications like many-class FSL, where feature vectors are generated using neural networks. The key contributions of this work are as follows:

1) **Combining ES and AS modes to filter redundant current during VSS in 3D NAND flash without overhead** : Hybrid-M reduces 20% to 52% in energy consumption by applying filtering in the hybrid matching process during VSS, as compared to using the AS mode alone.
2) **Enhancing filtering capability via MLC range encoding under WLs constraints in 3D NAND flash** : Filtering with MLC range encoding achieves 10% to 30% additional energy savings compared to SLC range encoding.
3) **Mitigating accuracy loss in hybrid mode via search voltage shifting** : Shifting the search voltage to a lower level recovers accuracy by 10% to 15% and further reduces energy consumption by 10% to 20%.
4) **Filtering-aware training (FAT) incorporates VSS and filtering during training to reduce energy consumption** : Hybrid-M with FAT further achieves a 3% to 8% in energy consumption and a 0.5% to 1% improvement in accuracy compared to Hybrid-M without FAT.

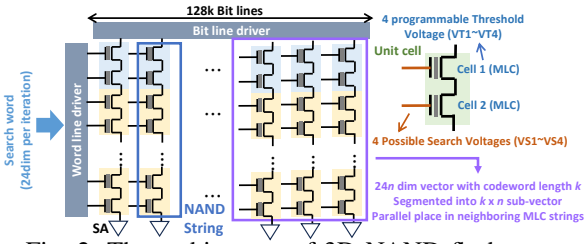


Fig. 2: The architecture of 3D NAND flash array.

II. Background and Motivation

A. Vector Similarity Search (VSS)

Vector similarity search (VSS) compares a query vector with stored vectors to identify the closest vector with similarity metrics. Common similarity measures include cosine similarity for angular comparison, as well as the L_1 [10] and L_∞ [17] norms for distance. VSS is integral to a wide range of AI applications, particularly in many-class few-shot learning (FSL) and approximate nearest-neighbor search (ANNS).

In many-class FSL, models are tasked with making classification using minimal labeled data. This is typically framed as N-way K-shot setting, where N classes with K labeled examples for each class are provided. VSS matches novel inputs (queries) against stored support vectors to infer class based on the closest matches. While cosine similarity is widely adopted in software-based implementations, its high computational complexity limits its scalability in hardware-based implementations. As a result, hardware-friendly alternatives such as the L_1 and L_∞ norms are gaining traction, offering reduced computational overhead while maintaining classification accuracy. On the other hand, ANNS [3], [4] is widely applied in scenarios requiring efficient retrieval from large, high-dimensional datasets. A critical performance metric for ANNS is the recall rate, expressed as N @ M. This metric reflects the proportion of top-N nearest-neighbors correctly identified within M retrieved vectors. High recall rates are essential for maintaining accuracy while leveraging approximation techniques to optimize computational efficiency. This paper explores VSS applications within both many-class FSL and ANNS, focusing on optimization for efficient and scalable implementations in extensive AI tasks.

B. 3D NAND Flash Architecture

Fig. 2 depicts the architecture of 3D NAND flash. Two cells are connected in series to form a word, and multiple words are concatenated within a NAND string. Each multi-level cell (MLC) word encodes two bits via complementary threshold voltages (VT). Search voltages (VS) applied to word-line gates enable parallel search across MLCs in the same row. 3D NAND flash supports two modes for in-memory search (IMS) : exact-search (ES) [13] and approximate-search (AS) [14]. The primary distinction between them lies in the voltage levels of VS. Table I lists the VT and VS combinations for MLC words, and Fig. 3(a) shows their voltage distributions.

In ES mode, VSi is carefully positioned above VTi but below the next VT (e.g., VS1 is above VT1 yet below VT2). Under mismatch, at least one VS falls below its corresponding VT, rendering the cell non-conducting and eliminating string current. As a result, ES mode is highly energy-efficient and

TABLE I: Quinary state representation for MLC.

Quinary State	Threshold Voltage		Search Voltage	
	Cell 1	Cell 2	Cell 1	Cell 2
00	VT1	VT4	VS1	VS4
01	VT2	VT3	VS2	VS3
10	VT3	VT2	VS3	VS2
11	VT4	VT1	VS4	VS1
xx	VT1	VT1	VS4	VS4

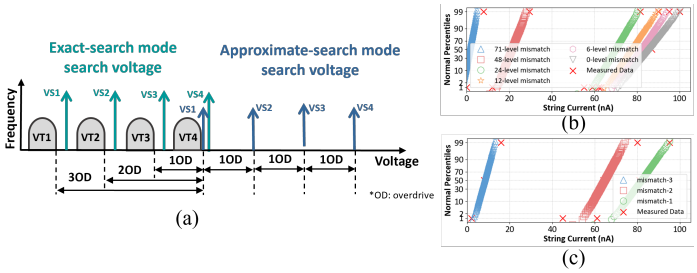


Fig. 3: (a) VS and VT settings for MLC in both ES and AS modes. Simulated current for AS mode (b) with varying string mismatch levels and (c) with identical string mismatch levels but differing maximum cell mismatches.

particularly suitable for applications like DNA sequence matching [13]. In contrast, AS mode sets all VS values above the maximum VT (e.g., VS1 is above VT4). In the presence of mismatches, the reduced voltage difference between VS and VT results in lower overdrives (ODs), decreasing the matching current. This behavior enables NAND flash to generate currents to approximate similarity in AS mode. The current is proportional to the ODs between search voltages and threshold voltages of NAND strings. Fig. 3(b) shows the current distribution of a 48-layer NAND string with mismatch levels ranging from 0 to 72 (24 words with up to 3 mismatch levels per word). As the mismatch level increases, the matching current decreases, reflecting greater distances between the query and stored vectors. Furthermore, Fig. 3(c) demonstrates that even with the same total mismatch levels, a single large mismatch in one cell (mismatch-n) can significantly reduce the overall current. To address this issue, prior work [18] proposed the multi-bit thermometer code (MTMC), which distributes mismatches more evenly across cells to improve reliability and accuracy. In this work, we also adopt MTMC in AS mode for VSS.

C. Motivational Experiments in Large-scale VSS

The high storage density of 3D NAND flash makes it ideal for IMS on large datasets. Prior work [14], [18] operates 3D NAND flash in AS mode to reduce data movement for large-scale VSS. However, 3D NAND flash incurs significant energy overhead since NAND strings always conduct regardless of mismatch levels in AS mode. Although comparisons of irrelevant data result in higher mismatch levels and smaller

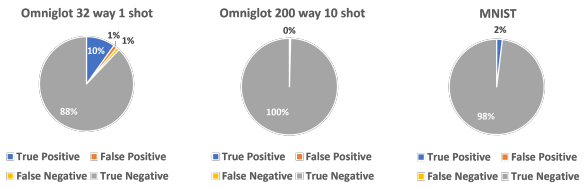


Fig. 4: Current breakdown in VSS using only AS mode.

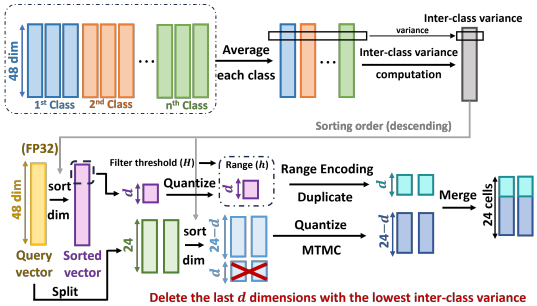


Fig. 5: Vector generation with d dimensions for filtering.

currents, their large volume leads to substantial unnecessary energy consumption. This inefficiency becomes critical in extensive comparison scenarios such as many-class FSL and ANNS, where redundant matching currents dominate energy consumption, as shown in Fig. 4.

Energy consumption in VSS can be categorized into four cases. "True Positive" occurs when the class with the largest current correctly corresponds to the target class, representing necessary energy usage. "False Positive" and "False Negative" indicate classification errors: "False Positive" occurs when the largest current corresponds to an incorrect class, leading to unnecessary energy usage; "False Negative" happens when the correct class does not yield the largest current, resulting in misclassifications despite the energy expended. The "True Negative" case, constituting the majority of currents, includes currents that neither correspond to the correct class nor represent the largest similarity, signifying substantial and unnecessary energy consumption.

Fig. 4 compares traditional FSL (left) and many-class FSL (middle), highlighting substantial energy inefficiency in the latter. As class count grows, a larger protion of energy is wasted on "True Negative" matching currents, exacerbating energy challenges and emphasizing the need for scalable, energy-efficient designs. To mitigate this, prior work [15], [19] applied filtering as a separate stage to reduce the energy of VSS. In this approach, filtering is performed on TCAM using the L_∞ norm, followed by VSS in compute-in-memory (CIM) using L_1 norm computation. While effective, this approach introduces additional hardware complexity and overhead. In contrast, designing a hybrid process with 3D NAND Flash using ES mode for filtering and AS mode for VSS can eliminate dedicated filtering hardware and reduce both energy and complexity without performance loss.

III. Proposed Method

In this paper, we propose techniques used in Hybrid-M for reducing VSS energy in 3D NAND flash. In Section III.A, we introduce an ES mode filtering mechanism that uses SLC range encoding, eliminating unnecessary currents by combining ES and AS modes. In Section III.B, we extend range encoding to MLC, increasing filtering capacity without increasing codeword length. In Section III.C, we lower VS by one OD in AS mode, counteracting the accuracy loss introduced by ES mode while further reducing current magnitudes. Finally, Section III.D presents a filtering-aware training strategy to improve accuracy and enhance energy efficiency.

A. Hybrid Matching Process Combining ES and AS modes

To mitigate the high energy consumption of AS mode without incurring additional area overhead, we propose a hybrid matching scheme that leverages both ES and AS modes in 3D NAND flash. We program a portion of the WLs in ES mode to filter irrelevant vectors by selectively disabling string currents, thereby reducing energy consumption. The remaining WLs operate in AS mode, performing VSS for similarity evaluation. As illustrated in Fig. 1(c), NAND cells in AS mode produce currents proportional to mismatch levels, enabling precise VSS, while NAND cells in ES mode filter irrelevant vectors, yielding zero current for the corresponding NAND strings. This hybrid approach achieves low-power operation and ensures that only relevant strings engage in VSS.

Vector generation with d-dimensional filtering operates as shown in Fig. 5. Each NAND string contains 24 MLC words with the first d words programmed in ES mode. To minimize the number of conducting strings after filtering, the top-d dimensions with the highest inter-class variance are selected as feature dimensions for filtering. A vector is considered potentially similar only if the difference in its corresponding input falls within a predefined threshold H. To enable threshold-based filtering in NAND cells using ES mode, each selected dimension is represented as a range and encoded using range encoding [15]–[17], [20]. For AS mode, we adopt MTMC for VSS, which converts each dimension to several codewords. These codewords are mapped across multiple NAND strings. To ensure consistent filtering of the vectors, identical d-dimensional filtering dimensions are applied to all strings of the same vector. If any filter mismatch occurs, the vector is considered dissimilar, and all associated strings are deactivated. Only the remaining active strings undergo VSS in AS mode.

B. Range Encoding for MLC

Programming WLs to ES mode in a fixed-length NAND string inevitably sacrifices the capacity available for AS mode, making extensive use of ES mode undesirable. Given the limited number of WLs, improving filtering effectiveness becomes a critical issue. A naive approach is to concatenate pairs of codewords from SLC-based range encoding and map them to MLC states. However, current MLC programming schemes cannot represent all 2-bit combinations of 0, 1, and x. Prior work has demonstrated that MLC words can represent quinary states (00, 01, 10, 11, xx) [13], as shown in Table I. We extend the programmable states to include 0x, x1, and 1x for enhancing the flexibility of codeword matching, as shown in Table II. The state x0 is infeasible due to VT and VS constraints, preventing direct support for all 2-bit SLC codewords in MLC.

Building on the approach of BORE [17], which introduces constrained programming (CP) to solve constraint satisfaction in range encoding, we apply CP to generate MLC codewords represented in octal states, enabling more flexible and precise matching. Due to limited storage per string, only one MLC word can be programmed for each dimension. CP produces codewords that are prioritized based on importance, and we select the most representative one for programming. When the threshold H is large, the resulting value range becomes wider, increasing the number of "don't care" (xx) bits and reducing filtering capability. To address this, we partition H into $h_s = \lceil \frac{H}{2} \rceil$ for stored vectors and $h_q = \lfloor \frac{H}{2} \rfloor$ for query vectors. This transformation converts the original point-in-range comparison into a range-to-range matching scheme, which maintains the effective matching interval for filtering while reducing the required codeword length.

TABLE II: Octal state representation for MLC.

Octal State	Threshold Voltage		Search Voltage	
	Cell 1	Cell 2	Cell 1	Cell 2
00	VT1	VT4	VS1	VS4
0x	VT1	VT3	VS2	VS4
01	VT2	VT3	VS2	VS3
x1	VT2	VT2	VS3	VS3
11	VT3	VT2	VS3	VS2
1x	VT3	VT1	VS4	VS2
10	VT4	VT1	VS4	VS1
xx	VT1	VT1	VS4	VS4

TABLE III: Mismatch level of AS mode.

w/o VS Shifting					**w/ VS Shifting**				
state	**1**	**2**	**3**	**4**	**state**	**1**	**2**	**3**	**4**
1	0	1	2	3	**1**	1	2	3	
2	1	0	1	2	**2**	2	1	2	3
3	2	1	0	1	**3**	3	2	1	2
4	3	2	1	0	**4**		3	2	1

TABLE IV: Mismatch level of ES mode.

Octal state	**00**	**0x**	**01**	**x1**	**11**	**1x**	**10**	**xx**
00	3	3						3
0x	3	2	3	3				2
01		3	3	3				2
x1		3	3	2	3	3		1
11				3	3	3		2
1x				3	3	2	3	2
10						3	3	3
xx	3	2	2	1	2	2	3	0

C. Shifting of Search Voltage in AS mode

In the hybrid matching process, active NAND strings are affected by mismatch levels from both ES and AS modes. The mismatch level is defined as the minimum OD between VT and VS within an MLC word, where a smaller OD corresponds to a higher mismatch level, with 0–3 OD mapped to levels 3–0, respectively. Table III and Table IV show mismatch levels introduced by AS and ES modes, respectively. As shown in Fig. 3(b)(c), larger maximum mismatch level—particularly level 3—significantly reduces matching current. Although NAND cells in AS mode produce currents proportional to similarity, the mismatch levels 2 and 3 from NAND cells in ES mode disrupt this correlation, degrading VSS accuracy. The effect of mismatch level 3 is especially severe, causing disproportionate current drops. Furthermore, the "don't care" (xx) codeword in ES mode spans multiple VT intervals, resulting in mismatch levels ranging from 0 to 3 without meaningful similarity correlation. This increases current inconsistency and further degrades matching accuracy.

To address these issues, we first remove the "don't care" from octal states and generate the codeword of range encoding using CP accordingly. We then shift VS in AS mode by one OD to mitigate the impact of high mismatch levels from NAND cells in ES mode, as shown in Fig. 6(a). This raises the baseline mismatch level, reducing the relative influence of ES

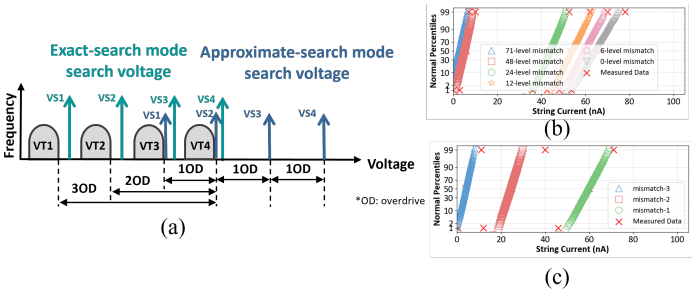


Fig. 6: (a) VS shifting in AS mode with NAND flash. AS mode simulated current during VS shifting for (b) varying string mismatch levels and (c) identical string mismatch levels with differing maximum cell mismatches, both based on [14].

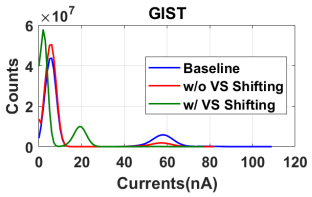


Fig. 7: Current distribution under different AS mode settings.

mismatches on matching current. As demonstrated in Fig. 7, without VS shifting (red), the right-side current peak from AS-only matching (blue) is suppressed, increasing SA errors. In contrast, VS shifting (green) restores the peak, improves current separation, and enhances VSS accuracy. The new mismatch range in AS mode spans levels 1–3, with extreme mismatches denoted by a slash in Table III. This also reduces overall current magnitude, lowering energy consumption without sacrificing performance.

D. Filtering-aware Training (FAT)

In our Hybrid-M approach, substituting AS mode vector dimensions with ES mode introduces two challenges. First, directly omitting AS mode dimensions can degrade accuracy. Second, selecting ES mode dimensions solely by inter-class variance may not yield effective filtering. We propose FAT to address both issues, preserving accuracy through optimized dimension removal while ensuring robust ES mode filtering.

FAT is a training strategy designed for tasks where vectors are extracted by neural networks. We develop FAT based on the hardware-aware training (HAT) framework [18], which incorporates flash characteristics into training for improving robustness under device non-idealities. Specifically, current variations, quantization errors, and MTMC behavior are modeled during meta-training to consider MCAM hardware. We further introduce a third stage—hybrid search–aware training—that accounts for behavioral differences between ES and AS modes, enabling the controller to adapt to hybrid matching and improving both reliability and energy efficiency.

The hybrid search-aware training, illustrated in Fig. 8, integrates L_∞ filtering to generate dimensionally reduced filter vectors that preserve classification accuracy and enhance energy efficiency. Begin by reloading the meta-training stage model and adjusting initial weight and bias values for the filtering mechanism. First, we compute inter-class and intra-class variances from the support vectors. For each NAND string containing 24 MLC words, we define subsequent vector dimensions in multiples of 24, applying a d-dimensional filter. Consequently, in a $D = 24n$ dimensional vector, $n \cdot d$ dimensions with the smallest inter-class variance are eliminated, freeing space for ES mode weights and biases. Next, to improve generalization under distribution shifts between training and testing, we select d filtering dimensions from the top $u \cdot d$ entries with the lowest intra-class variance. The averaging factor u controls selection diversity; larger values are preferred when the training–testing gap is wider. These entries are partitioned into d groups and averaged to derive representative weights, providing a robust initialization aligned with the overall data distribution. During the hybrid search-aware training, we introduce a learnable threshold $\mathbf{T}_{ES}$ and gating function $\mathbf{G}_{ES}(\cdot)$ to simulate the L_∞ filtering effect. Given the ES mode distance $\mathbf{D}_{ES}$ and AS mode similarity $\mathbf{S}_{AS}$, the final similarity outcome $\mathbf{S}_O$ is specified by:

$$\mathbf{S}_O = \mathbf{S}_{AS} \cdot \mathbf{G}_{ES}(\mathbf{D}_{ES} \le \mathbf{T}_{ES}) = \begin{cases} \mathbf{S}_{AS}, & \mathbf{D}_{ES} \le \mathbf{T}_{ES} \\ 0, & \mathbf{D}_{ES} > \mathbf{T}_{ES} \end{cases}$$

since the gating function is non-differentiable around the threshold, it leads to gradient vanishing during backpropagation. To address this, a sigmoid function is applied during training to preserve gradient flow. By leveraging FAT, the integration of ES and AS modes yields robust, hardware-aware solutions for large-scale VSS.

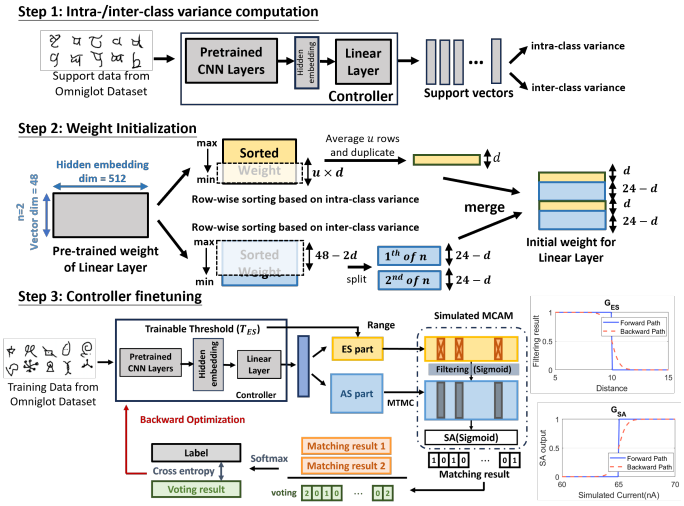


Fig. 8: Training Flow of proposed Filtering-aware Training.

IV. Evaluation

A. Experimental Setup

We evaluate our design on two FSL tasks: Omniglot [22] and CUB-200-2011 (CUB) [23]. For Omniglot, we employ a Conv4 [24] model with 48 dimensions and set the average number $u = 3$ under a 200-way 10-shot setting, which requires 128k NAND strings with a codeword of length 31 for AS mode. For CUB, we adopt a ResNet12 [25] model with 480 dimensions and $u = 10$ under a 50-way 5-shot setting, which needs 125k NAND strings and a codeword of length 25 for AS mode.

We also evaluate three ANNS tasks—Fashion-MNIST [26], MNIST [27], and GIST [28]—using L_2 norm as ground truth [3], [4] to retrieve the top 100 nearest neighbors. Fashion-MNIST and MNIST have 60,000 training and 10,000 test samples with 784 dimensions, while GIST has 1,000,000 training

TABLE V: Setting and statistic of MCAM simulation.

Device	3D NAND Flash
Capacity (KB)	750
Number of WL	48
Number of BL	128k
Search time(us)	50
Supply Voltage(V)	0.5
VT1 - VT4 (V)	[2, 3, 4, 5]
AS mode VS1 - VS4(V)	[4.5, 5.5, 6.5, 7.5]

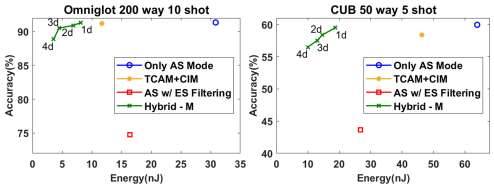


Fig. 9: Energy evaluation of many-class FSL.

and 1,000 test samples with 960 dimensions. Fashion-MNIST and MNIST employ a codeword of length 5 (requiring 9,900k NAND strings), whereas GIST uses a codeword of length 9 for AS mode (360,000k NAND strings). Since these datasets are composed of pre-trained vectors, FAT is not applied.

The estimates of energy consumption for NAND flash are based on measured current distributions [14]. The setting and statistics of MCAM simulations are listed in Table V.

B. Reduction of Energy

Fig. 9 and Fig. 10 show evaluations on many-class FSL and ANNS tasks. Blue dots indicate the AS-only baseline. Red dots replace one AS mode dimension with ES mode filtering, using the highest inter-class variance, but without VS shifting, resulting in accuracy loss due to mismatch levels 2 and 3 disturbing current distributions. For ANNS, the green curve shows that shifting VS mitigates current suppression and improves accuracy. For FSL, it combines VS shifting with FAT to reduce energy without accuracy loss. We denote nd as n-dimensional L_∞ filtering per string (e.g., 1d means one-dimensional filtering). Compared to the AS-only baseline, 1d filtering saves 74% and 71% energy on Omniglot and CUB with under 0.5% accuracy loss. On Fashion-MNIST, MNIST, and GIST, it reduces energy by 77%, 83%, and 67%, with under 0.01 recall drop. Compared to TCAM+CIM [15], which filters with TCAM-based L_∞ and computes L_1 norms in CIM, our

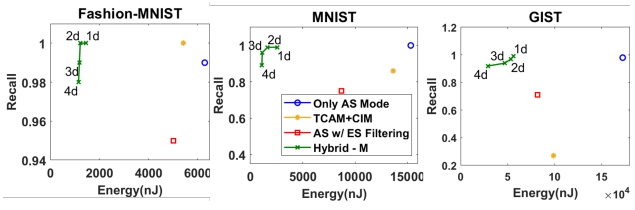


Fig. 10: Energy evaluation of ANNS.

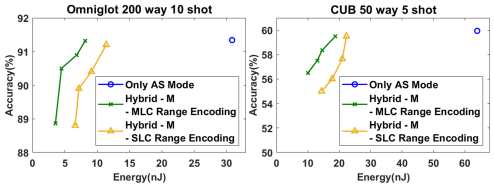


Fig. 11: SLC and MLC range encoding for many-class FSL.

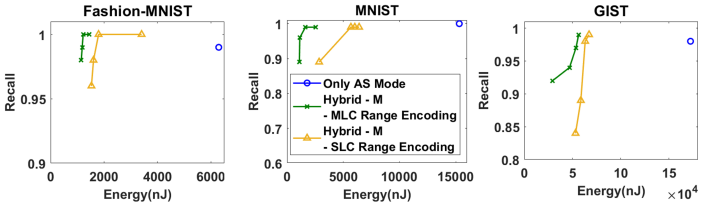


Fig. 12: SLC and MLC range encoding for ANNS.

TABLE VI: Comparison of one-dimensional filtering with VS shifting and the integration of FAT.

Dataset	**Omniglot**		
	Only AS Mode	VS Shifting	FAT
Accuracy	91.34%	90.8%	91.32%
Filtered energy	0%	66%	74%
Dataset	**CUB**		
	Only AS Mode	VS Shifting	FAT
Accuracy	59.94%	58.5%	59.5%
Filtered energy	0%	68%	71%

1d filtering design in Hybrid-M achieves higher accuracy and 10–70% better energy efficiency, demonstrating its advantage in NAND-based MCAM systems.

C. Comparison of Range Encoding between SLC and MLC

Fig. 11 and 12 compare our MLC range encoding with SLC-style methods that use only a subset of MLC states. These approaches underutilize the capacity of MLC, weakening filtering and accuracy. In contrast, our method achieves stronger filtering and higher accuracy, balancing energy efficiency and reliability.

D. Effectiveness of FAT under hybrid matching process

Table VI shows VSS accuracy and energy under different settings. Without training, VS shifting alone causes accuracy degradation. FAT improves both accuracy and energy by jointly learning filtering and VSS. FAT is applicable to trainable scenarios, such as FSL, where the controller is adapted to the hybrid matching process.

V. Conclusion

This work introduces Hybrid-M, a 3D NAND-based in-memory VSS architecture that addresses energy inefficiencies in large-scale VSS. By integrating ES and AS modes, Hybrid-M leverages ES mode filtering to remove redundant comparisons in AS mode, significantly reducing energy consumption while preserving accuracy. We propose three key enhancements–MLC range encoding, VS shifting, and FAT–to optimize hybrid search. Experimental results show that Hybrid-M lowers energy usage by 67–83% compared to MCAM-based VSS using only AS mode in 3D NAND flash, while maintaining accuracy across many-class FSL and ANNS workloads. These findings confirm the scalability and efficiency of Hybrid-M, making it a promising solution for large-scale VSS applications.

Acknowledgment

This work was supported by the National Science and Technology Council of Taiwan under Grants of MOST 111-2218-E-002-018-MBK and NSTC 112-2218-E-002-025-MBK.

REFERENCES

[1] Lu Liu, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang, "Many-Class Few-Shot Learning on Multi-Granularity Class Hierarchy," IEEE Transactions on Knowledge and Data Engineering, vol. 34, no. 5, pp. 2293-2305, 2022.

[2] Jiawei Dang, Yu Zhou, Ruirui Zheng, and Jianjun He, "A Large-Class Few-Shot Learning Method Based on High-Dimensional Features," Applied Sciences, vol. 13, no. 23, pp. 12843, 2023.

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull, "ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms," Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

[4] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull, "Reproducibility Protocol for ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Search Algorithms," Information Systems, 2020.

[5] Lewis, Patrick, *et al*., "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Syst. (NeurIPS)*, pp. 9459-9474, 2020.

[6] Ashish Ranjan, Shubham Jain, Jacob R. Stevens, Dipankar Das, Bharat Kaul, and Anand Raghunathan, "X-mann: A crossbar based architecture for memory augmented neural networks," Proceedings of the 56th Annual Design Automation Conference 2019, pp. 1–6, 2019.

[7] Han-Wen Hu, Wei-Chen Wang, Yuan-Hao Chang, Yung-Chun Lee, Bo-Rong Lin, Huai-Mu Wang, Yen-Po Lin, Yu-Ming Huang, Chong-Ying Lee, Tzu-Hsiang Su, and others, "Ice: An intelligent cognition engine with 3D NAND-based in-memory computing for vector similarity search acceleration," 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 763–783, 2022.

[8] Harsha Simhadri. Research talk: Approximate nearest neighbor search systems at scale. https://www.youtube.com/watch?v=BnYNdSIKibQ&list=PLD7HFcN7LXReJTWFKYqwMcCc1nZKIXBo9&index=9, 2021.

[9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. Proceedings of the VLDB Endowment, 2019.

[10] Haitong Li, Wei-Chen Chen, Akash Levy, Ching-Hua Wang, Hongjie Wang, Po-Han Chen, Weier Wan, Win-San Khwa, Harry Chuang, Y.-D. Chih, Meng-Fan Chang, H.-S. Philip Wong, and Priyanka Raina, "SAPIENS: A 64-kb RRAM-Based Non-Volatile Associative Memory for One-Shot Learning and Inference at the Edge," IEEE Transactions on Electron Devices, vol. 68, no. 12, pp. 6637-6643, 2021.

[11] Haitong Li, Wei-Chen Chen, Akash Levy, Ching-Hua Wang, Hongjie Wang, PoHan Chen, Weier Wan, H-S Philip Wong, and Priyanka Raina, "One-shot learning with memory-augmented neural networks using a 64-kbit, 118 GOPS/W RRAMbased non-volatile associative memory," 2021 Symposium on VLSI Technology, pp. 1–2, 2021.

[12] Mohsen Imani, Xunzhao Yin, John Messerly, Saransh Gupta, Michael Niemier, Xiaobo Sharon Hu, and Tajana Rosing, "SearcHD: A memory-centric hyperdimensional computing with stochastic training," IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol. 39, no. 10, pp. 2422–2433, 2019.

[13] Po-Hao Tseng, Feng-Ming Lee, Yung-Chun Li, Yu-Hsuan Lin, Chih-Wei Hu, Chih-Chang Hsieh, Ming-Hsiu Lee, Kuang-Yeu Hsieh, Keh-Chung Wang, and Chih-Yuan Lu, "Multilevel In-Memory-Searching in 3D NAND-Flash Memory," 2022.

[14] Po-Hao Tseng, Tian-Cig Bo, Yu-Hsuan Lin, Yu-Chao Lin, Jhe-Yi Liao, Feng-Ming Lee, Yu-Yu Lin, Ming-Hsiu Lee, Kuang-Yeu Hsieh, Keh-Chung Wang, and Chih Yuan Lu, "SLC and MLC In-Memory-Approximate-Search Solutions in Commercial 48-layer and 96-layer 3D-NAND Flash Memories," the 2022 International Conference on Solid State Devices and Materials, pp. 443-444, 2022.

[15] Laguna, A. F., Yin, X., Reis, D., Niemier, M. & Hu, X. S. Ferroelectric FET based In-Memory Computing for Few-Shot Learning. Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI, pp. 373-378, 2019.

[16] Anat Bremler-Barr, Yotam Harchol, David Hay and Yacov Hel-Or, "Encoding Short Ranges in TCAM Without Expansion: Efficient Algorithm and Applications," IEEE/ACM Transactions on Networking, vol. 26, no. 2, pp. 835-850, 2018.

[17] Chi-Tse Huang, Cheng-Yang Chang, Hsiang-Yun Cheng and An-Yeu Wu, "BORE: Energy-Efficient Banded Vector Similarity Search with Optimized Range Encoding for Memory-Augmented Neural Network," 2024 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6, 2024

[18] Chiang, Hao-Wei, Chi-Tse Huang, Hsiang-Yun Cheng, Po-Hao Tseng, Ming-Hsiu Lee and An-Yeu Wu. "Efficient and Reliable Vector Similarity Search Using Asymmetric Encoding with NAND-Flash for Many-Class Few-Shot Learning." ArXiv abs/2409.07832 (2024).

[19] Reis, D., Laguna, A. F., Niemier, M. & Hu, X. S. Attention-In-Memory for Few-Shot Learning with Configurable Ferroelectric FET Arrays. In Proceedings of the 26th Asia and South Pacific Design Automation Conference, pp. 49–54, 2021.

[20] Ann Franchesca Laguna, Michael Niemier and X. Sharon Hu, "Design of Hardware-Friendly Memory Enhanced Neural Networks," 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 2019, pp. 1583-1586.

[21] K. J. Eldredge, F. F. Roohparvar, L. De Santis, and T. Vali, "Apparatus and methods of operating memory for exact and inexact searching of feature vectors," U.S. Patent 9,595,334 B2, Mar. 14, 2017.

[22] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum, "Human-level concept learning through probabilistic program induction," Science, vol. 350, no. 6266, pp. 1332–1338, 2015.

[23] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The Caltech-UCSD Birds-200-2011 Dataset," California Institute of Technology, Technical Report CNS-TR-2011-001, 2011.

[24] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, and others, "Matching networks for one shot learning," Advances in Neural Information Processing Systems, vol. 29, 2016.

[25] Boris Oreshkin, Pau Rodríguez López, and Alexandre Lacoste, "Tadam: Task dependent adaptive metric for improved few-shot learning," Advances in Neural Information Processing Systems, vol. 31, 2018.

[26] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms," arXiv preprint arXiv:1708.07747, 2017.

[27] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012.

[28] H. Jégou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 1, pp. 117-128, 2011.

April: Accuracy-Improved Floating-Point Approximation For Neural Network Accelerators

Yonghao Chen[1], Jiaxiang Zou[2,†], Xinyu Chen[1,*]

[1]The Hong Kong University of Science and Technology (Guangzhou)
[2]University of Electronic Science and Technology of China

***Abstract*—Neural Networks (NNs) have achieved breakthroughs in computer vision and natural language processing. However, modern models are computationally expensive, with floating-point operations posing a major bottleneck. Floating-point approximation, such as Mitchell's logarithm, enables floating-point multiplication using simpler integer additions, thereby improving hardware efficiency. However, its practical adoption is hindered by challenges such as precision degradation, efficient hardware integrations, and management of trade-offs between accuracy and resource efficiency.**

In this paper, we propose a hardware-efficient down-sampling-based compensation method to mitigate precision loss and a flexible bias mechanism to accommodate diverse data distributions in NN models. Building on this foundation, we design configurable systolic arrays optimized for NN accelerators. To further support practical adoption, we introduce April, a co-design framework that balances the accuracy and resource usage of generated synthesizable systolic arrays. Our FPGA-based evaluations demonstrate that April-generated systolic arrays reduce root mean square error (RMSE) by up to 96% and achieve 34%-52% area reduction even compared to INT8-based implementations while maintaining comparable or improved model accuracy. Our design is open-sourced at https://github.com/CLabGit/April.

***Index Terms*—Floating-point approximation, FPGAs**

I. Introduction

Neural Networks (NNs), including convolutional neural networks [1]–[3], vision transformers [4]–[6], and large language models [7]–[9], have achieved state-of-the-art performance across various applications, such as computer vision and natural language processing. However, their exceptional performance comes at the cost of high computational complexity, with floating-point multiplications emerging as a major bottleneck. While floating-point arithmetic ensures numerical precision, its implementation demands significant chip area.

Recent advancements in floating-point approximation have demonstrated great potential to reduce the hardware costs of multiplication operations. Based on Mitchell's logarithm approximation [10] that a floating-point number can be interpreted as a number in a logarithmic number system, recent works [11]–[13] theoretically prove that floating-point multiplication can be performed with integer additions. This suggests that *expensive floating-point multiplication units can be replaced with simpler integer addition units*, unlocking substantial resource-saving opportunities for heavy computations of NNs.

However, a significant gap remains in applying the aforementioned floating-point multiplication approximation (FPMA) to real hardware. First, the transition from a floating-point number to a logarithmic number inevitably introduces precision degradation. As a result, an effective and hardware-efficient compensation solution is essential for maintaining the model's accuracy. Second, comprehensive hardware designs for NNs that seamlessly integrate both approximation and compensation methods are necessary to quantitatively evaluate the benefits of FPMA in real-world settings. Third, since compensation methods inherently involve trade-offs between precision and hardware costs, a flexible framework is needed to guide these design decisions based on application-specific requirements. In this paper, we address these challenges and make the following contributions:

- We propose a hardware-efficient down-sampling-based compensation method, together with a flexible bias mechanism to FPMA to improve model accuracy.
- We develop configurable systolic arrays that integrate proposed accuracy-improved FPMA.
- We introduce April, a framework that balances model accuracy and hardware cost for systolic array generation.
- Extensive FPGA evaluations demonstrate that April generated systolic array (FP8) reduces RMSE by up to 96% and saves 34%-52% hardware resources compared to INT8-based systolic arrays while offering better model accuracy.

II. Background and Motivation

A. Approximating FP Multiplication with Integer Addition

According to the IEEE 754-2019 Standard [14], a normalized FP (*Floating-Point*) number x can be represented as,

$$x = (-1)^{S_x} \times 2^{E_x - b} \times (1 + M_x),\ 0 \le M_x < 1, \quad (1)$$

with S_x as a sign bit, E_x as an W_E-bit exponent, $b = 2^{W_E-1} - 1$ as a bias, and M_x as a W_M-bit mantissa without the leading one. Although FP provides a wide dynamic range, operations like multiplication are hardware-intensive due to the complexity of managing the sign, exponent, mantissa, and additional steps like rounding and normalization.

While integer-based quantization has been widely used to reduce computation overhead by replacing floating-point

*Corresponding author: Xinyu Chen
†Work conducted during internship at HKUST(GZ)

multiplication with simpler integer multiplication [15], [16], Gustafsson et al. [11] further simplifies FP multiplication by approximating it as an integer addition. Specifically, based on Mitchell's logarithmic approximation [10], a floating-point number x can be mapped to the logarithmic domain as:

$$\log_2(x) \approx (E_x - b) + M_x = X - B, \tag{2}$$

where $X = \{E_x, M_x\}$ represents the binary form of x, and $B = b \ll W_M$ is the shifted bias. While approximation from $\log_2(1 + M_x)$ to M_x introduces some errors, it enables the Floating-Point Multiplication Approximation (FPMA). The binary representation of the multiplication result R can be expressed as:

$$R \approx log2(x \times y) + B = X + Y - B, \tag{3}$$

where X and Y denote the binary representations of the two operands. This method simplifies the complexity of floating-point multiplication by reducing it to integer addition.

B. Challenges of Applying FPMA to NN Accelerators

Approximation techniques, while promising for improving hardware efficiency, often result in precision loss [17], [18], which can severely degrade model accuracy; therefore, we first study the impact of FPMA on model accuracy. We compared the accuracy of MobileNetV2 and other models under the original FP8 data type (golden accuracy) and various approximated FP8 formats. Table I summarizes the results. The results reveal that naively applying FPMA can lead to unacceptable accuracy drops. For example, MobileNetV2's accuracy decreases from 66.28% (golden) to 58.37% under FP8(E4M3), representing an 8% reduction. Similarly, FastViT-t12 suffers a drastic decline from 76.59% to 30.47% under FP8(E3M4). While some configurations, such as FP8(E2M5) with a mantissa width of 2, introduce no error, others exhibit significant accuracy degradation, particularly with wider mantissa bit-widths. These findings highlight two critical challenges in applying FPMA to hardware accelerators for neural networks.

TABLE I: Top-1 accuracy comparison of MobileNetV2, FastViT, DeiT-T and MobileOne on the ImageNet-1k dataset, with original FP8 or FPMA.

NN Model	Accuracy	E2M5	E3M4	E4M3	E5M2
MobileNetV2 [19]	‡Golden	71.08%	70.30%	66.28%	48.11%
	FPMA	64.65%	62.76%	58.37%	†48.11%
FastViT-t12 [20]	‡Golden	65.38%	76.59%	74.04%	56.30%
	FPMA	0.64%	30.47%	67.02%	†56.30%
DeiT-T [21]	‡Golden	68.40%	69.10%	68.02%	64.54%
	FPMA	0.08%	0.27%	66.75%	†64.54%
MobileOne [22]	‡Golden	67.74%	67.84%	41.63%	0.46%
	FPMA	3.52%	16.94%	14.58%	†0.46%

† FPMA will not introduce error for M2 (mantissa width of 2) [23].
‡ Golden accuracy on FP8 obtained with post-training quantization [24].

Challenge 1: Hardware efficient error correction for high model accuracy. The accuracy loss introduced by FPMA, as shown in Table I, highlights the need for practical and hardware-efficient error correction methods. Existing works [11], [23] explore fine-grained or single-value compensation methods to mitigate precision loss. However, they either suffer from high hardware costs or insufficient accuracy. Balancing effective error correction with minimal hardware overhead remains a key challenge for deploying FPMA in neural network accelerators.

Challenge 2: Efficient NN hardware design with FPMA. To our knowledge, no existing hardware accelerators incorporate FPMA for NNs, leaving a significant gap in architecture design. On the other hand, error correction usually introduces trade-offs between computational efficiency and model accuracy. Consequently, hardware design with FPMA introduces new architectural considerations, such as selecting the right error correction granularity based on application requirements, making hardware support of FPMA nontrivial.

C. Our Solutions Toward Practical FPMA for NNs

Solution 1: Improved Precision with hardware-efficient compensation and flexible bias. To address the precision loss introduced by FPMA, we propose a hardware-efficient compensation method combined with flexible bias. The compensation mechanism uses localized averaging through a downsampling approach, reducing hardware costs while maintaining accuracy. This method dynamically adjusts the error compensation window size to balance precision and efficiency. Additionally, a flexible bias mechanism allows adaptive bias settings for activations and weights to accommodate their distinct distribution patterns. This ensures a wider representable range and significantly enhances model accuracy, overcoming the limitations of fixed-bias FPMA designs.

Solution 2: A model accuracy-aware framework for FPMA-enabled systolic array generation. We present April, a co-design framework for generating systolic arrays with accuracy-enhanced FPMA. This framework integrates error correction and format conversion, enabling optimized compensation window sizes and seamless data format adjustments. It also supports customizable systolic array designs tailored to user-defined accuracy and hardware cost requirements. By leveraging April, this solution effectively optimizes the trade-offs between hardware efficiency and computational accuracy, offering a practical FPMA integration to NN accelerators.

III. Hardware-Efficient Error Correction

A. Error Distribution Analysis

To design an effective error correction strategy, we first analyze the error characteristics in detail. Figure 1 presents the error distribution of $X \times Y$ across approximated FP8 formats (E4M3, E3M4, and E2M5). Since the error arises solely from the mantissa (as approximated by $log_2(1+M_x) \approx M_x$), we use the mantissa values of X and Y (denoted as X_{mant} and Y_{mant}) as coordinates of the heatmap figures. For instance, in Figure 1a, E4M3 has three mantissa bits, so X_{mant} and Y_{mant} range from 0 to 7. Each heatmap enumerates all possible combinations of X_{mant} and Y_{mant} for a given data format, ensuring comprehensive coverage of potential errors. The heatmap values reflect the difference between the golden and approximate results at these specific mantissa values and are

measured in units of last place (ulp), a widely adopted metric for error analysis [11], [25].

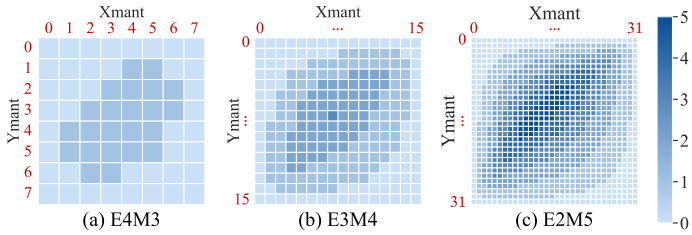


Fig. 1: Visualization of error distribution of $X \times Y$ with various approximated FP8 formats. Values measured in units of the last place (ulp) given the mantissa values of X and Y.

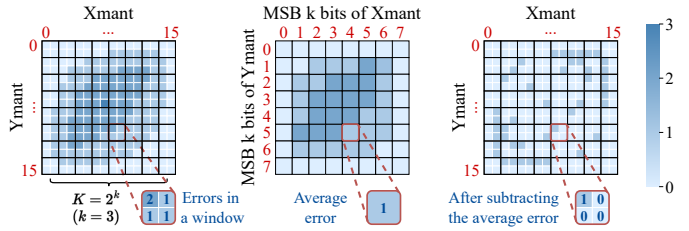
The analysis reveals three key findings. First, the errors are nonuniformly distributed, making it difficult to establish a simple arithmetic relationship between errors and mantissa values for direct compensation [23]. Second, error values vary significantly in heatmaps. As a result, applying a constant compensation value for each X_{mant}-Y_{mant} combination (i.e., a point (X_{mant}, Y_{mant}) in the heatmap) yields poor accuracy [11]. Third, as the number of mantissa bits increases (from E4M3 to E2M5), the error distribution transitions from sparse to dense. This leads to fine-grained compensation (i.e., each X_{mant}-Y_{mant} combination has an individual compensation value) impractical due to the high on-chip storage cost required [23].

Despite these challenges, we identify an opportunity in the strong locality of the error distribution. Adjacent error data points (X_{mant}, Y_{mant}) often exhibit similar values within small regions of the heatmap. This suggests that errors in these regions can be compensated with a single representative value, significantly reducing hardware complexity and storage requirements. Leveraging this observation, we propose a down-sampling-based compensation method that exploits the locality of the error distribution, providing a hardware-efficient solution for FPMA error correction.

B. Down-Sampling-Based Compensation Method

We propose a novel down-sampling-based compensation method that balances numerical accuracy and hardware cost. *The core idea is to average the errors of mantissa combinations (X_{mant},Y_{mant}) within a defined window and apply the averaged value as compensation.* Smaller window sizes provide higher compensation accuracy, while larger windows trade some accuracy for better hardware efficiency. Furthermore, our approach is versatile and can be effectively applied to any FP format such as BF16, FP16, and FP32, as it is determined by the data format and is independent of the input data distribution.

Figure 2 provides a step-by-step visual representation of how the original error heatmap of E3M4 is transformed into a compensated version through down-sampling. First, the original error map is divided into $K \times K$ windows, where $K = 2^k$ and k is the down-sampling factor. A larger k corresponds to smaller window sizes, allowing for finer granularity. In the example shown in Figure 2a, $k = 3$ results in $2^3 \times 2^3$ windows, each of size 2×2. Second, the errors within each window are averaged to generate a down-sampled error heatmap, as illustrated in Figure 2b, where the size is reduced to $K \times K$. Third, each X_{mant}-Y_{mant} combination uses the most significant k bits of X_{mant} and Y_{mant} to lookup the averaged value for compensation. Figure 2c shows error distribution after applying the down-sampling-based compensation method, demonstrating significantly reduced errors.



(a) Divide the original error heatmap into $K \times K$ sub-block. (b) Each sub-block is down-sampled. (c) After compensation. Error has been redued significantly.

Fig. 2: Transformation of the original error heatmap of E3M4 using the down-sampling-based compensation method.

This approach only requires storing the compact down-sampled heatmap, minimizing hardware demands while maintaining sufficient compensation accuracy. The trade-off between hardware efficiency and numerical precision can be tuned by adjusting the k, which also served as a compensation factor. The compensation factor k is usually chosen within the range $[3, W_M]$, where W_M is the total bit width of the mantissa. When $W_M = 2$, no error is introduced by FPMA [23]. For practical implementations, k is typically set to $k \geq 3$ to effectively capture the error distribution while minimizing the index size. On current FPGA architectures, with $k = 3$, the down-sampled heatmap with the size of 8×8 can be stored in one LUT6 [26] that has 2^6 entries. If k is increased, additional LUT6 units are required to accommodate the larger input space.

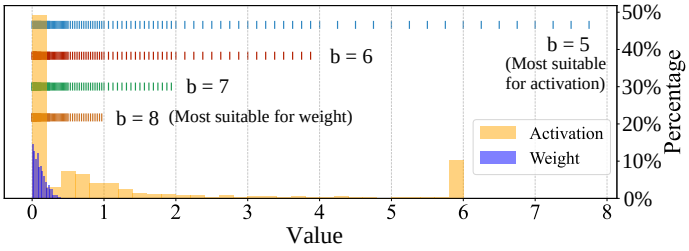


Fig. 3: Distribution of activation and weight values of MobileNetV2 network and numerical representation space of E3M4 under different biases (b).

C. Enabling Flexible Bias to FPMA

For FP formats, the bias (b, as shown in Equation 1) not only simplifies the representation of exponents but also enables flexibility to adjust the representation range of FP numbers. The original FPMA [11] uses a uniform bias, providing the same range of representable numbers for inputs and outputs (i.e., X, Y, and R). However, the distributions of activations and weights of NNs can vary significantly. Figure 3 illustrates the distribution of activations (orange) and weights (purple) in

a linear layer of MobileNetV2 and the numerical space of the E3M4 FP8 format under different bias values (b = 5, 6, 7, 8), all in absolute value. The horizontal axis represents the numerical range of FP numbers determined by the bias, while the vertical axis shows the percentage of activations and weights within specific ranges. The results show that bias b = 5 is most suitable for activations, as their distribution aligns closely with the representable range, whereas b = 8 best accommodates weights for the same reason. Selecting an appropriate bias for one often leads to suboptimal representation for the other, potentially degrading model accuracy. This motivates the need for flexible or adaptive bias strategies.

In this paper, we propose a flexible bias design for FPMA and provide theoretical proof of its correctness. By scaling the original input x with a factor of $2^b/2^{b_1}$, Equation 2 adjusts to:

$$\begin{aligned} log_2(x\times\frac{2^b}{2^{b_1}}) &= log_2(2^{E_x-b+b-b_1}\times(1+M_x)) \\ &= (E_x-b_1)+log_2(1+M_x)\approx(E_x-b_1)+M_x \\ &= X-B_1, \end{aligned} \tag{4}$$

where $B_1 = b_1 << W_M$. This adjustment enables the binary representation X to represent a custom value space scaled by $2^b/2^{b_1}$. Similarly, for scaling y by $2^b/2^{b_2}$, we derive:

$$log_2(y\times\frac{2^b}{2^{b_2}})\approx Y-B_2. \tag{5}$$

When the two scaled input operands are multiplied, their product can be expressed as:

$$\begin{aligned} &log_2(x\times\frac{2^b}{2^{b_1}}\times y\times\frac{2^b}{2^{b_2}}) = log_2(x\times y\times\frac{2^{2b}}{2^{b_1+b_2}}) \\ &= log_2(x\times y\times\frac{2^b}{2^{b_1+b_2-b}})\approx R-B_3, \end{aligned} \tag{6}$$

where $B_3 = (b_1+b_2-b) << W_M = B_1+B_2-B$ is used for the replacement of B in the FPMA calculation. This flexible bias adjustment ensures that the computation remains accurate and faithful to the scaled value space, enabling FPMA to adapt seamlessly to diverse data range distributions in NNs.

IV. Accuracy-aware Systolic Array Generation

A systolic array is a hardware architecture commonly used for accelerating matrix operations, particularly in the linear layers of neural networks. By integrating accuracy-improved FPMA into a systolic array, our design largely reduces hardware complexity while maintaining high model accuracy. Furthermore, we propose April, a comprehensive framework that determines the optimal window size for error compensation, performs data format conversion, and generates a synthesizable systolic array tailored to the user's model accuracy and hardware cost requirements.

A. Integrating Accuracy-Improved FPMA to Systolic Arrays

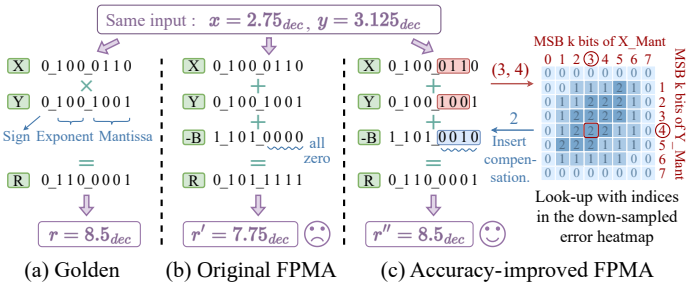
Figure 4 illustrates the multiplication process of $x\times y$ using three methods: standard floating-point multiplication (Golden), original FPMA, and the proposed accuracy-improved FPMA, demonstrated in the FP8(E3M4) format. While the original FPMA performs multiplication using only two integer additions, it introduces a noticeable error (8.5_{golden} vs. 7.75). In contrast, the accuracy-improved FPMA compensates for this error through an additional table lookup operation. Specifically, the most significant three bits (MSBs) of the mantissa of X_{mant} and Y_{mant} are extracted and used as indices to query the precomputed error compensation table. The retrieved compensation value is then added to the result, correcting the error and producing an accurate output ($r'' = 8.5$). This demonstrates that our approach effectively reduces errors with minimal additional computational and hardware overhead.

Figure 5 illustrates the proposed configurable systolic array architecture integrated with accuracy-improved FPMA for efficient linear layer computations. It utilizes an output-stationary dataflow strategy. The systolic array (a) consists of interconnected MAC units that process inputs (Iact, Wght, CascO) and output results (Oact, PassO), enabling pipelined matrix operations. Each MAC unit (b) incorporates an optimized FPMA module that performs lightweight floating-point multiplication with error compensation and flexible bias. The FPMA module (c) decomposes inputs (Iact, Wght) into sign, exponent, and mantissa components, extracts the most significant k bits of the mantissa, and uses them to query precomputed compensation values stored in LUT6 units. The retrieved values are directly appended to the lower vacant bits of the flexible bias (Flexible.B) and then added with Iact and Wght through an area-efficient ternary adder [27].

B. A Software-Hardware Co-designed Framework - April

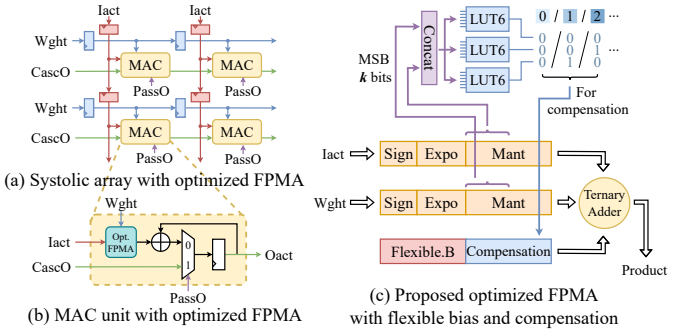
The April framework is a hardware-software co-design solution designed to balance the trade-offs between accuracy and resource consumption for FPMA-based systolic arrays. Essentially, given the model accuracy and hardware cost constraints from users, the framework could determine the most suitable compensation factor k and biases for each layer and generate the synthesizable hardware code for systolic arrays. Figure 6 outlines its workflow.

Step ① involves **Flexible Bias Selection**, where the framework selects the optimal bias for each tensor. This is achieved by analyzing the distribution of activation ($Iact$) and weight ($Wght$) values and aligning them with the numerical space of the floating-point (FP) format. This step ensures accurate and efficient representation through flexible bias adjustment.



(a) Golden (b) Original FPMA (c) Accuracy-improved FPMA

Fig. 4: Comparison of multiplication methods in FP8(E3M4) format: (a) Standard floating-point multiplication (Golden), (b) Original FPMA, and (c) Accuracy-improved FPMA.



(a) Systolic array with optimized FPMA

(b) MAC unit with optimized FPMA

(c) Proposed optimized FPMA with flexible bias and compensation

Fig. 5: Design of the proposed configurable systolic array with accuracy-improved FPMA for linear layers.

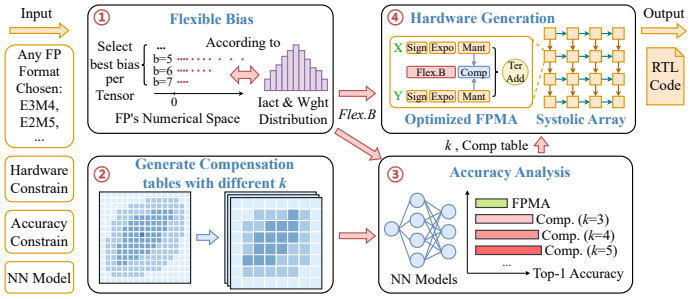


Fig. 6: The proposed April framework.

Step ② performs **Compensation Table Generation** by generating error compensation tables for various compensation factors (k) using down-sampling techniques. These precomputed tables correct FPMA errors while maintaining hardware efficiency.

Step ③ conducts **Accuracy Analysis**, where neural network models are simulated with FPMA to evaluate the impact of different compensation configurations ($k = 3, 4, 5$, etc.) on accuracy. This step identifies the optimal trade-off between accuracy and resource usage, allowing developers to choose parameters based on their optimization priorities, such as resource efficiency or accuracy maximization.

Step ④ involves **Hardware Generation**. The optimized compensation table and flexible bias settings are integrated into the systolic array design. The hardware generator produces RTL code for systolic arrays, incorporating components such as the ternary adder, flexible bias logic, and compensation modules to enable efficient FPMA computation.

The proposed April framework seamlessly integrates model-level accuracy tuning with hardware design space exploration, enabling resource-efficient and numerically accurate systolic arrays for neural network hardware accelerators.

V. Evaluation

A. Experimental Setup

For accuracy evaluation, experiments were conducted on NVIDIA GeForce RTX 3090 GPU, Torch 2.4.1, and CUDA 12.6, leveraging an open-source FP8 quantization tool from Qualcomm [24]. For resource evaluation, designs are implemented on an AMD Alveo U250 Accelerator Card with 1,728K LUTs and 3,456K registers, using Vivado 2022.2.

B. Evaluation on Down-Sampling based Compensation

1) Error Analysis: Table II evaluates the error reduction capability of the proposed compensation method for FPMA across formats such as FP8, BF16, and FP16, using metrics such as Mean Absolute Error (MeanAbsErr) and Maximum Absolute Error (MaxAbsErr) in units of last place (ulp). The results demonstrate significant error reductions across all formats, with MeanAbsErr reductions ranging from 74.1% (FP8_E3M4) to 81.1% (FP16_E5M10) and MaxAbsErr reductions ranging from 60.0% (FP8_E2M5) to 77.2% (BF16_E8M7). Errors were fully eliminated for FP8_E4M3 (100% reduction). These results highlight the method's effectiveness in minimizing errors for improved model accuracy.

TABLE II: Error analysis of the proposed compensation method across different floating-point formats.

FPMA	MeanAbsErr (ulp)			MaxAbsErr (ulp)		
Formats	Base.	Comp.	Reduction	Base.	Comp.	Reduction
FP8_E5M2	0	0	/	0	0	/
FP8_E4M3	0.36	0	**↓100%**	1	0	**↓100%**
FP8_E3M4	0.85	0.22	↓74.1%	3	1	↓66.6%
FP8_E2M5	1.79	0.48	↓73.2%	5	2	↓60.0%
FP12_E5M6	3.62	0.77	↓78.7%	11	3	↓72.7%
BF16_E8M7	7.27	1.44	↓80.2%	22	5	**↓77.2%**
FP16_E5M10	58.22	10.98	**↓81.1%**	175	52	↓70.3%

2) Area Analysis: Figure 7 compares area consumption (LUTs) of original FPMA, the proposed accuracy-optimized FPMA, and FPMA with fine-grained compensation [23] across various floating-point (FP) formats. While the overhead of fine-grained compensation method expands dramatically as the mantissa width increases, the proposed accuracy-optimized FPMA exhibits only a linear increase in LUT consumption relative to the original FPMA baseline, demonstrating efficient scalability even with higher bit-widths. The additional area required for compensation remains minimal, ensuring the design's area efficiency while maintaining improved computational accuracy.

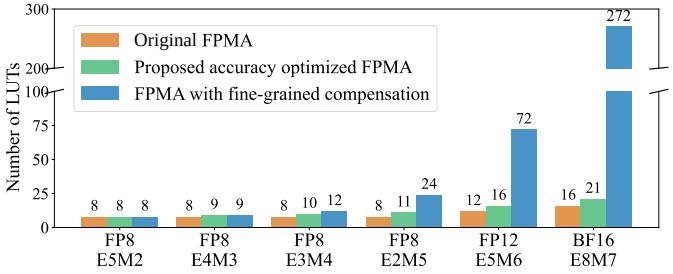


Fig. 7: Area (LUTs) consumption of a single multiplication unit implemented with different methods. The compensation factor k is set to 3 for our accuracy-optimized FPMA.

C. Evaluation on April-generated Systolic Arrays

1) Error Analysis: Figure 8 evaluates the accuracy of April-generated systolic arrays. Random matrices with values uniformly distributed in the range [1, 2) were represented in various formats and multiplied on systolic arrays with different mantissa widths (M) and matrix sizes ($S = 32, 64, 128$). Results were compared to systolic arrays with original FPMA

(FPMA) using Root Mean Square Error (RMSE). First, April-generated systolic arrays demonstrate significant error reductions over baselines (FPMA) across all settings. Second, smaller mantissas (e.g., $M3$, $M4$) showed RMSE reductions of up to 100%, while larger mantissas (e.g., $M7$, $M8$) achieved reductions of up to 96% with larger k.

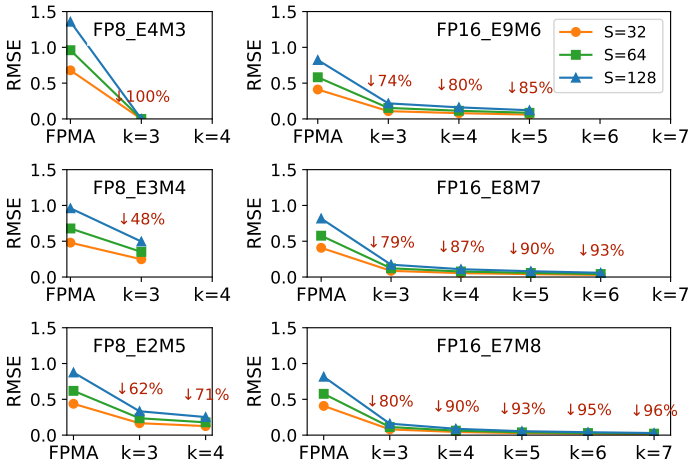


Fig. 8: RMSE analysis of matrix multiplication on April-generated systolic array with various mantissa widths (M), matrix sizes (S), and compensation factors (k). The baseline is the systolic array with original FPMA (FPMA).

2) Area Analysis: Table III compares area consumption of April-generated systolic arrays with INT8 and original FPMA-based systolic arrays. All configurations are matched in terms of parallelism and pipeline depth, with DSP usage intentionally avoided, making LUT utilization the primary metric. The results show that April-generated systolic arrays deliver significant LUT area reductions across all configurations, with smaller formats like E5M2 achieving up to 52% reduction consistently across matrix sizes. Formats with larger mantissas (e.g., E2M5) result in slightly smaller reductions of 34%-35%, reflecting a trade-off between hardware cost and precision.

TABLE III: Area (LUT) cost of April-generated (April) and original FPMA (FPMA) based systolic arrays compared to INT8-based (INT8) systolic arrays. Vivado strategy is set to Flow_AreaOptimized_high, with the frequency of 250MHz.

Formats	Mult	16×16	32×32	64×64
INT8	Exact	23808	95303	380291
E5M2	FPMA	11630	46183	184126
	April	†11630 (↓51%)	†46183 (↓52%)	†184126 (↓52%)
E4M3	FPMA	13446	52974	210956
	April	13774 (↓42%)	53629 (↓44%)	214169 (↓44%)
E3M4	FPMA	13724	55429	221075
	April	14241 (↓40%)	57786 (↓39%)	229274 (↓40%)
E2M5	FPMA	14887	59145	236195
	April	15662 (↓34%)	62145 (↓35%)	248628 (↓35%)

† FPMA does not introduce errors when mantissa equals 2.

D. End-to-end Model Accuracy Evaluation

The experiment in Figure 9 evaluates the end-to-end top-1 accuracy of four neural networks on the ImageNet-1k dataset under different configurations, including Golden INT8 (post-training quantization with INT8), Golden FP8 (post-training quantization with FP8), original FPMA (FPMA), FPMA with the proposed compensation method (FPMA + Comp), and the April (i.e., FPMA + Comp + flexible bias) with varying compensation factor ($k = 3, 4, 5$).

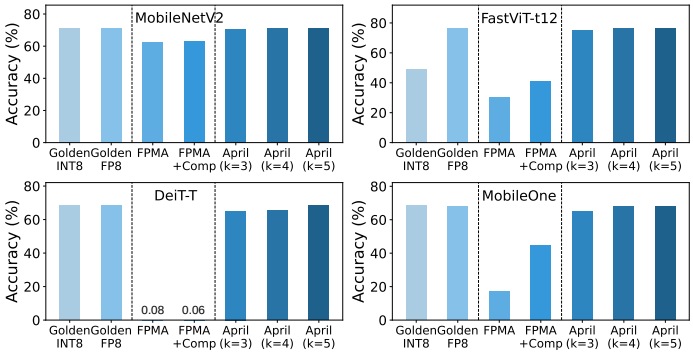


Fig. 9: Comparison of top-1 accuracy for neural networks on the ImageNet-1k dataset under different configurations.

The results show that original FPMA significantly reduces accuracy compared to both Golden FP8 and Golden INT8, highlighting the need for effective error correction. FPMA with compensation recovers most of the accuracy loss, and the April framework further improves accuracy, often matching the Golden FP8 baseline and achieving comparable or better results than Golden INT8, particularly with larger compensation factors (e.g., $k = 5$). Different models exhibit varying sensitivity to FPMA, with lightweight models like MobileOne benefiting substantially from the April framework. While earlier experiments showed significant resource reduction from accuracy-improved FPMA, these findings confirm its ability to maintain end-to-end model performance, making it practical for neural network accelerators.

VI. Conclusion

This paper addressed the challenges in adopting FPMA to neural network accelerators. We proposed accuracy-improved FPMA with down-sampling-based compensation and flexible bias methods to mitigate precision loss in FPMA. We introduced April, a co-design framework that balances accuracy and resource efficiency for systolic array generation. April-generated systolic arrays demonstrated up to 52% area reduction compared to INT8-based ones while maintaining better accuracy, and achieved up to 96% RMSE reduction compared to the original FPMA, making it a practical and efficient solution for deploying FPMA in neural network accelerators. This work bridges the gap between floating-point approximation and practical hardware implementation.

Acknowledgment

This work is supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No.2025A03J3568). We also thank the AMD Heterogeneous Accelerated Compute Cluster (HACC) Program [28] for providing access to hardware resources.

REFERENCES

[1] Z. Liu *et al.*, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 11 976–11 986.

[2] X. Ding *et al.*, "Repvgg: Making vgg-style convnets great again," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13 728–13 737, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231572790

[3] M. Tan *et al.*, "Efficientnetv2: Smaller models and faster training," in *International conference on machine learning*. PMLR, 2021, pp. 10 096–10 106.

[4] A. Vaswani *et al.*, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[5] A. Dosovitskiy *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *ArXiv*, vol. abs/2010.11929, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:225039882

[6] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9992–10 002, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:232352874

[7] T. Brown *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[8] H. Touvron *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *ArXiv*, vol. abs/2307.09288, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259950998

[9] A. Liu *et al.*, "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," *arXiv preprint arXiv:2405.04434*, 2024.

[10] J. N. Mitchell, "Computer multiplication and division using binary logarithms," *IRE Transactions on Electronic Computers*, no. 4, pp. 512–517, 1962.

[11] O. Gustafsson *et al.*, "Approximate floating-point operations with integer units by processing in the logarithmic domain," in *2021 IEEE 28th Symposium on Computer Arithmetic (ARITH)*. IEEE, 2021, pp. 45–52.

[12] A. Kosson *et al.*, "Multiplication-free transformer training via piecewise affine operations," *Advances in Neural Information Processing Systems*, vol. 36, pp. 8208–8223, 2023.

[13] T. Mogami, "Deep neural network training without multiplications," 2020. [Online]. Available: https://arxiv.org/abs/2012.03458

[14] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

[15] A. Gholami *et al.*, "A survey of quantization methods for efficient neural network inference," *ArXiv*, vol. abs/2103.13630, 2021.

[16] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[17] M. Imani *et al.*, "Approxlp: Approximate multiplication with linearization and iterative error control," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19, 2019.

[18] C. Chen *et al.*, "Pam: A piecewise-linearly-approximated floating-point multiplier with unbiasedness and configurability," *IEEE Transactions on Computers*, vol. 71, no. 10, pp. 2473–2486, 2022.

[19] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] P. K. A. Vasu *et al.*, "Fastvit: A fast hybrid vision transformer using structural reparameterization," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 5785–5795.

[21] H. Touvron *et al.*, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[22] P. K. A. Vasu *et al.*, "Mobileone: An improved one millisecond mobile backbone," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023, pp. 7907–7917.

[23] T. Lindberg *et al.*, "On approximate 8-bit floating-point operations using integer operations," *arXiv preprint arXiv:2406.18441*, 2024.

[24] A. Kuzmin *et al.*, "Fp8 quantization: The power of the exponent," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 651–14 662, 2022.

[25] J.-M. Muller, "On the definition of ulp(x)," 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:264519989

[26] "Amd fpga advantages over competing legacy lut4 architectures," AMD, Tech. Rep., 2024. [Online]. Available: https://docs.amd.com/v/u/en-US/wp558-amd-lut6

[27] M. Kumm *et al.*, "Multiple constant multiplication with ternary adders," in *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–8.

[28] (2025) Heterogeneous accelerated compute cluster (HACC) at NUS. Https://xacchead.d2.comp.nus.edu.sg/.

Towards Uncertainty-aware Robotic Perception via Mixed-signal BNN Engine Leveraging Probabilistic Quantum Tunneling

Likai Pei*, Yu Zhou*, Xingtian Wang*, Xueji Zhao*, Wanxin Huang*, Boyang Cheng*, Halid Mulaosmanovic†, Stefan Duenkel†, Dominik Kleimaier†, Sven Beyer†, Kai Ni*, Mengxue Hou*, Michael Niemier*, and Ningyuan Cao*
*University of Notre Dame, South Bend, IN USA
†GlobalFoundries Module One LLC & Co. KG, Dresden, Germany

***Abstract*—Integrating deep learning with environmental perception enhances robotic adaptability to complex tasks. However, its "black-box" nature, such as the lack of uncertainty quantification, poses challenges for safety-critical applications, particularly in unstructured and noisy environments. Bayesian neural networks (BNNs) offer uncertainty quantification but are limited by high hardware overhead, restricting real-time implementation on resource-constrained robots. This paper presents a mixed-signal hardware accelerator for BNNs, utilizing probabilistic quantum tunneling in fully depleted silicon-on-insulator (FD-SOI) transistors to enable efficient, real-time uncertainty quantification. Device measurements indicate high-quality Gaussian random variable generation, validated through quantile-quantile plot analysis, with a high correlation coefficient (r = 0.997) at 200 fJ/sample. Leveraging such compact randomness, the parallel architecture achieved 10^3–$10^4\times$ latency reduction at less than $2\times$ area cost. Finally, in uncertainty-aware visual localization application of autonomous underwater vehicles, the BNN model effectively distinguishes data noise from model uncertainty, yielding significant information gain and enhancing the resampling efficiency by $4.5\times$ at same accuracy.**

Keywords: robotic perception, uncertainty quantification, mixed-signal, Bayesian neural networks, quantum, active perception, retraining, compute-in-memory

I. Introduction

Robotic perception has advanced with deep learning, improving how robots understand and interact with environment, including visual Simultaneous Localization and Mapping (SLAM) [1] and semantic reconstruction [2]. Despite significant advancements, the "black-box" nature of deep learning remains a major limitation, particularly for uncertainty quantification (UQ) [3]. For example, current learning-based visual SLAM methods fail to account for uncertainty in pose or map estimates, leading to localization errors, especially in unseen or partially known environments [4].

Bayesian Neural Networks (BNNs) uniquely enable uncertainty quantification within deep learning by sampling model weights from learned distributions rather than relying on fixed, deterministic weights. Furthermore, BNNs can separately quantify uncertainty arising from sensory/environmental noise (aleatoric uncertainty) and the model itself (epistemic uncertainty) [5], enabling robots to make informed decisions during normal operation or the learning process. However, implementing such sampling in hardware poses significant challenges. State-of-the-art digital [6], analog [7], and mixed-signal [8] approaches for BNNs often suffer from high energy consumption, increased latency, and limited on-chip programmability [9]. These constraints are particularly restrictive in robotic applications, which have constrained computational resources, and demand rapid decision-making and real-time adaptability.

To tackle these challenges, this paper introduces a mixed-signal BNN accelerator, as shown in Fig. 1, leveraging the probabilistic quantum tunneling properties of fully depleted silicon-on-insulator (FD-SOI) devices. The key contributions are summarized as follows:

- **Compact and energy-efficient GRNG**: The proposed FD-SOI based Gaussian Random Number Generator (GRNG) achieves a compact design (6 F^2/RNG) and ultra-low energy consumption (200 fJ/sample). Leveraging quantum tunneling, it generates high-quality randomness with a correlation coefficient ($r = 0.997$) and low autocorrelation (< 0.2) validated by device measurements.
- **Parallel BNN sampling**: Using the compact GRNG cells, we develop a custom circuit and architecture that supports fully parallel BNN sampling. This design reduces sampling latency by $10^3 - 10^4\times$ with less than $2\times$ on-chip area, enabling real-time and resource-constrained UQ.
- **Hardware-algorithm co-design**: We propose a co-design framework tailored for uncertainty-aware applications. Evaluations on Autonomous Underwater Vehicles (AUVs) visual localization demonstrate significant reductions in retraining data volume ($4.5\times$) and labeling costs, showcasing its effectiveness for practical deployment.

II. Background and Discussions

A. Robotic Perception

Robotic perception refers to a robot's ability to sense, interpret, and understand its environment using data collected from various sensors [10]. This capability enables robots to make reliable decisions and perform tasks autonomously, whether navigating through an area, manipulating objects, or interacting with humans.

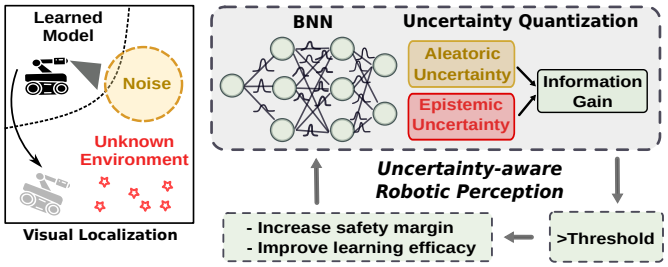


Fig. 1: Overview of robotic perception with uncertainty estimation.

Recent advances in robotic perception have been driven by deep learning models like DeepLab [11] (for semantic segmentation), Deep Visual Odometry [12] (for visual odometry and motion estimation), and PointNet [13] (for 3D point cloud processing), which enhance localization and scene understanding for processing 2D and 3D vision data. These techniques allow robots to map environments more effectively and simplify the perception-to-action process, increasing overall efficiency. End-to-end learning systems further optimize operations by directly mapping sensor inputs to outputs, while multi-modal approaches, which integrate data from RGB cameras [14], [15], LiDAR [16], and radar [17], [18], enhance robustness and reliability in interpreting diverse and complex environments.

B. Uncertainty Quantification

Advancements in learning-based robotic perception emphasize the critical role of UQ in decision-making within dynamic, noisy, and unstructured environments [19]. UQ allows robotic systems to dynamically adjust safety margins and optimize learning strategies, thereby enhancing robustness and performance.

Robotic perception UQ requires distinguishing between two primary types of uncertainty:

- *Aleatoric uncertainty* stems from inherent noise or variability in the data, such as sensor noise or environmental factors like fog or fluctuating lighting. This uncertainty is tied to the data generation process and is irreducible, even with additional data.
- *Epistemic uncertainty* arises from limited knowledge or insufficient training data, reflecting the model's inability to generalize to unseen scenarios. Unlike aleatoric uncertainty, epistemic uncertainty can be reduced by acquiring more diverse data or improving the model architecture.

Separately quantifying these uncertainties enables adaptive measures. For example, when aleatoric uncertainty is high, the robot can utilize temporal sampling or activate additional sensors to improve reliability. Conversely, high epistemic uncertainty signals the need for targeted data acquisition to address unknown structures and expedite retraining with minimal overhead.

To further enhance learning efficiency, information gain can be employed to assess the source of uncertainty. Information gain increases with epistemic uncertainty but decreases with aleatoric uncertainty, offering a unified metric for continual learning in robotic systems.

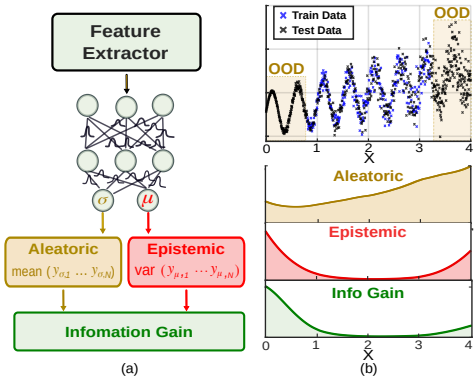


Fig. 2: (a) Overview of BNN-based UQ [20] and (b) example regression task and its UQ results.

C. BNN-based Uncertainty Quantification

BNNs are notable for their capacity to provide probabilistic estimates of uncertainty in predictions [21], [22]. BNNs treat network weights as posterior distribution of the weights, formalized as $P(\mathbf{W}|\mathbf{X},\mathbf{Y}) = \frac{P(\mathbf{Y}|\mathbf{W},\mathbf{X})P(\mathbf{W})}{P(\mathbf{X})}$.Here, $P(\mathbf{W}|\mathbf{X},\mathbf{Y})$ denotes the posterior probability of the weights $\mathbf{W}$ given the input $\mathbf{X}$ and output $\mathbf{Y}$, $P(\mathbf{Y}|\mathbf{W},\mathbf{X})$ represents the likelihood of observing the given outputs for a set of weights and inputs, $P(\mathbf{W})$ is the prior distribution of the weights, and $P(\mathbf{X})$ is the evidence or the probability of observing the input data.

To estimate model uncertainty in a task $y = f(x)$, where x is the input, f is the model, and y is the predicted value, we assume that the model output follows a Gaussian distribution: $y \sim \mathcal{N}(\mu(x), \sigma(x)^2)$. The BNN is trained to generate two outputs, y_μ and y_σ. Here, y_μ represents the predicted value, while y_σ represents the variance of the prediction. Model uncertainty is estimated by sampling the BNN weights and running the model N times.

For aleatoric uncertainty (AU), we compute the mean of y_σ across N samples is $AU = \frac{1}{N}\sum_{n=1}^{N} y_{\sigma,n}$. Epistemic uncertainty (EU) is derived from the variance of y_μ across N samples: $EU = \frac{1}{N}\sum_{n=1}^{N}\left(y_\mu^n - \bar{y}_\mu\right)^2$. To measure the information gain (IG) from the uncertainties, we define $IG = \log\left(1 + \frac{EU}{AU+\delta}\right)$, where δ is a small constant added to prevent division by zero in the denominator [20], [23].

Fig. 2(b) illustrates the BNN UQ with a regression example. The task involves predicting the function $\frac{1}{2}x + \frac{1}{2}\sin(25x) + \mathcal{N}(0, \sigma_{noise})$, where σ_{noise} increases with x. The results demonstrate that the BNN effectively captures the increasing noise with higher aleatoric uncertainty and assigns higher epistemic uncertainty to out-of-distribution (OOD) regions. Additionally, the BNN achieves higher information gain in low-noise OOD regions, highlighting its ability to adaptively quantify uncertainty.

D. BNN Hardware Overhead

Due to the computational complexity of directly approximating the posterior distribution, methods such as variational inference are commonly employed, approximating the posterior with a more tractable Gaussian distribution $q(\mathbf{W}|\theta)$, where

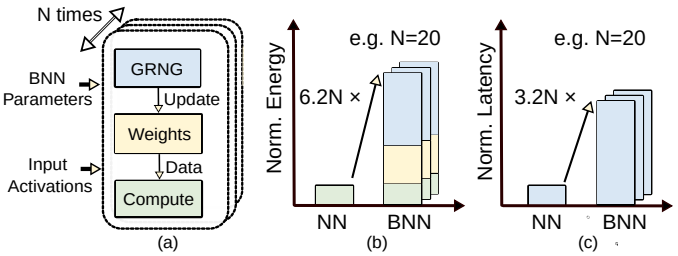


Fig. 3: (a) Overview of conventional BNN hardware implementation. (b) Normalized energy overhead compared to regular neural networks. (c) Normalized latency overhead compared to regular neural networks. These results are based on simulations using a 22nm PDK [6], comparing against an 8-bit digital baseline with a single fully connected layer of size 2048×2.

θ represents the set of variational parameters [24]. However, compared to conventional neural networks, BNNs require the additional step of generating GRNG and performing weight updates for each inference cycle. This sequential generation and update process significantly increases both energy consumption and latency (shown in Fig. 3).

Various hardware designs address sampling overhead in BNNs. Digital approaches using FPGAs and ASICs optimize GRNG with advanced pipeline and data reuse strategies to reduce computational and data transfer costs [25]–[29]. However, these methods still rely on sequential GRNG and weight updates, with extensive sampling iterations leading to high power consumption and latency.

As shown in Table. I, recent compute-in-entropy (CIE) architectures leverage mixed-signal circuits to enable programmable in-memory mixed-signal GRNG and compute-in-memory (CIM) [8]. While energy-efficient, their large GRNG unit cell area limits suitability for fully parallel BNN sampling, which demands substantial on-chip area. Other works utilize emerging non-volatile memory devices, exploiting device fabrication or programming variability for stochastic behavior in BNN [9]. However, these approaches suffer from unfreshable randomness or limited programmability of random variables (e.g., highly correlated mean and variance), making them unsuitable for BNNs.

Feature	[30]	[6]	[9]	[8]	This work
GRNG	Digital	Digital	Analog	MS	MS
Computation	Digital	CIM	CIM	CIM	CIM
No Weight Update	×	✓	✓	✓	✓
Parallel Sampling	×	×	✓	×	✓
Programmability	✓	✓	×	✓	✓

TABLE I: Comparison of different state-of-the-art BNN implementations.

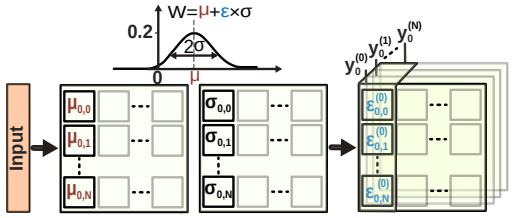


Fig. 4: Overview of Gaussian decomposition and presented parallel BNN sampling architecture.

III. Architecture, Circuit and Device

This section will discuss the parallel sampling BNN architecture, its custom circuit design, and the compact FD-SOI GRNG utilizing probabilistic quantum tunneling.

A. Gaussian Decomposition

To reduce data access and take advantage of CIM and mixed-signal GRNG, we decompose the normally distributed weights, $w_{ij} \sim \mathcal{N}(\mu, \sigma)$ into $w_{ij} = \mu + \epsilon \cdot \sigma$, as demonstrated in the compute-in-entropy (CIE) architecture proposed in [8], where ϵ is drawn from a standard normal distribution $\epsilon \sim \mathcal{N}(0, 1)$, scaled by σ, and shifted by μ, as shown in Fig. 4. This reconfiguration splits the dynamic weight into two static parameters, μ and σ, and a dynamic random variable, ϵ. This decomposition allows the Multiply-Accumulate (MAC) operations to be reformulated as:

$$y_i^{(n)} = \sum_i x_i \cdot w_{ij}^{(n)} = \sum_i x_i \cdot \mu_{ij} + \sum_i x_i \cdot \epsilon_{ij}^{(n)} \cdot \sigma_{ij} \quad \text{for } n = 1, \ldots, N \tag{1}$$

where i, j represents the weight index, n represents sampling index and N is total number of required sampling.

B. Circuit and Architecture for Parallel Bayesian Sampling

With Gaussian decomposition, we arrange computation and GRNG within memory arrays in a CIE architecture. As shown in Fig. 5(a), static parameters μ and σ are stored in SRAM arrays, while a parallel GRNG matrix, composed of ϵ cells, is associated with each σ cell.

μ arrays: The μ array enables vector-matrix multiplication between input activations and the μ parameters of the dynamic weights. This operation is activated only once when the input activations remain unchanged. It follows the standard CIM architecture, as shown in Fig. 5(b). Here, input activations X are represented by time pulses applied on wordlines, controlling the on-off states of access transistors. In this work, μ parameters are represented with 8-bit precision to balance accuracy and hardware efficiency.

σ-ϵ array: The σ-ϵ array in Fig. 5(c) facilitates the computation of the vector-matrix multiplication for the dynamic components of the weights, $\Sigma x_i \cdot \epsilon_{ij} \cdot \sigma_{ij}$. Unlike the CIE architecture in [8], which uses sequentially sampled GRNG using supply voltage noise, this design employs parallel FD-SOI transistors as GRNGs for significantly reduced sampling latency at minimal area cost. It leverages the quantum tunneling effect that can generate holes to be stored in the floating body, thus generating unique threshold voltages during write operations. For each σ, we integrate N parallel FD-SOI devices alongside their respective computation modules. Acting as current sources, these FD-SOI devices perform multiple parallel samples in a single clock cycle, producing all $I_{\varepsilon i,j}$ values required to control the discharge cycle for each σ module. All FD-SOI devices share common control signals, ensuring compactness and efficiency.

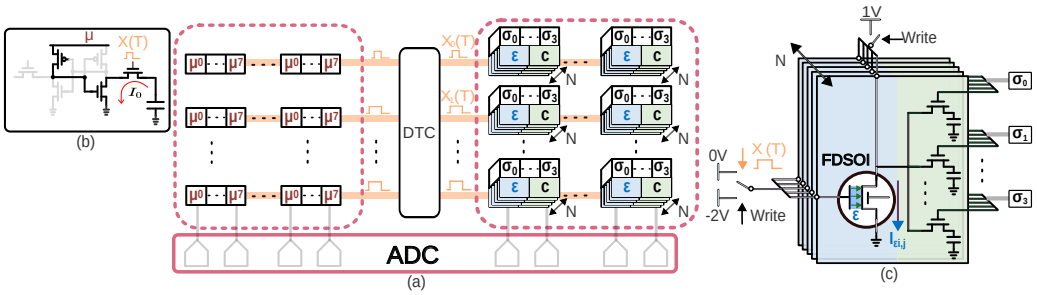


Fig. 5: (a) Architecture based on decomposed dynamic weights, illustrating circuit details for each bit of the μ component and the parallel sampling σ cells. The design leverages dynamic weight decomposition to enable efficient computation, with each (b) μ-bit paired with corresponding (c) σ cells operating in parallel to facilitate simultaneous sampling and enhanced processing efficiency.

The stored σ values in SRAM, combined with time-domain input activations, regulate the charge-discharge cycle and corresponding discharge time. Prior work [8] demonstrates that σ precision can be reduced to 2 bits with negligible performance degradation. In our implementation, we use 4-bit precision to provide additional margin for robustness.

This process is carried out simultaneously by a parallel architecture, with multiple sampled results sequentially processed through an ADC to complete the final calculations. This yields the mean and variance of the predictions, providing a measure of uncertainty.

C. GRNG with Probalistic Quantum Tunneling

To enable parallel GRNG sampling on resource-constrained edge devices, we propose a single-transistor FD-SOI GRNG leveraging probabilistic quantum tunneling. This design minimizes area, energy, and latency while ensuring high randomness quality, long retention, and convenient updates to support BNN functionality.

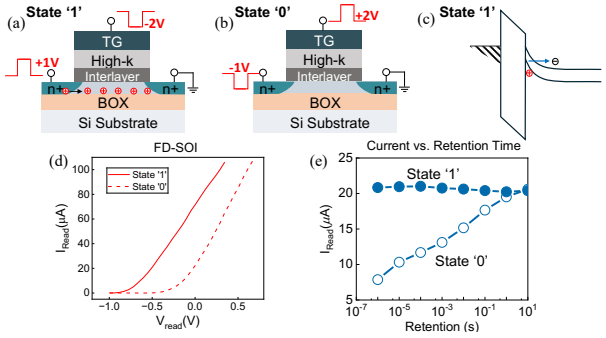


Fig. 6: (a) State '1', (b) State '0', (c) The quantum tunneling of electrons for State '1', (d) Single-step FD-SOI Sensing, (e) FD-SOI retention performance.

FD-SOI based floating body DRAM: In a FD-SOI transistor, the silicon body is floating as it is isolated by the top gate dielectric and also the buried oxide (BOX). Leveraging the floating body effect, many work propose to use this device as a floating body dynamic random access memory (DRAM), where the free charge, in this case holes, is stored in the floating body and could leak away over time [31]. As an example, Fig. 6(a) and (b) shows the biases applied to FD-SOI transistor integrated on the 22nm platform [32]. To write a DRAM "1" state, a negative/positive bias is applied to the gate/drain, respectively. Such a large bias at the gate/drain overlap region will induce band-to-band tunneling (BTBT), such that the electrons in the valence band will tunnel to the conduction band [33], as shown in Fig. 6(c), and leaving behind holes. Since the quantum tunneling process is inherently probabilistic, which is harnessed in this work as GRNG. After electron-hole pair generation, electrons are quickly collected by the drain terminal and the holes will accumulate in the floating body, raise the body potential, and reduce the threshold voltage, as shown in Fig. 6(d). Meanwhile, with a retention duration exceeding 10 ms, as shown in Fig. 6(e), the device enables global GRNG sampling for an entire BNN inference or even multiple inferences without requiring re-sampling. This significantly reduces the average GRNG energy consumption and control overhead.

Conversely, when a positive bias is applied to the gate and the body-drain junction, as shown in Fig. 6(b), holes are depleted from the floating body, thus decreasing the channel potential and hence increasing the threshold voltage. To validate the aformentioned mechanisms of the FD-SOI transistor, Fig. 6(d) shows the measured I_D-V_G curves measured after the write '0' and '1' respectively. The distinct difference in current levels between State '1' (higher current) and State '0' (lower current) confirms the capability of distinguishing these two states.

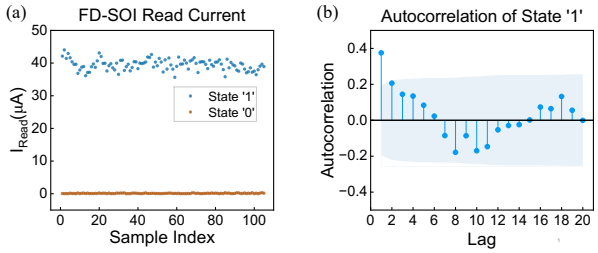


Fig. 7: (a) Current comparison of State '1' and '0', (b) The Autocorrelation Analysis of State '1' Samples.

GRNG Measurements: With measurements on fabricated 22nm FD-SOI devices, we observe a write energy of 200 fJ per GRNG sample, the stochastic behavior of the FD-SOI device is characterized by repeatedly switching between State '0' and State '1' for 105 cycles, with each cycle lasting 1 μs. As shown in Fig. 7(a), in State '1', the floating body accumulates sufficient charge, leading to a higher channel current. Con-

versely, in State '0', the floating body is depleted of charges, resulting in a channel current at the instrument noise floor. This significant difference between the two states ensures reliable detection of the device's state. The independence of the generated currents in State '1' is validated by the autocorrelation plot in Fig. 7(b), which shows minimal correlation between successive samples. Most autocorrelation values fall within the confidence bounds, confirming the statistical independence of the measurements.

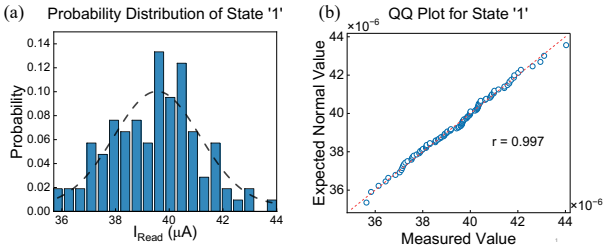


Fig. 8: (a) The histogram of State '1' Samples, (b) The Q-Q plot analysis of State '1' samples for Gaussian distribution.

The statistical properties of the State '1' current are further analyzed to verify its Gaussian nature. The histogram of the measured current, shown in Figure. 8(a), reveals a Gaussian-like distribution. To further confirm this, the Q-Q plot in Fig. 8(b) compares the measured current values to theoretical normal values. The data points align closely along the reference line, with a correlation coefficient of 0.997, indicating that the distribution of the channel currents in State '1' closely follows a Gaussian distribution.

These findings underscore the FD-SOI device's capability to generate high-quality Gaussian random variables efficiently, making it a suitable candidate for real-time uncertainty quantification in Bayesian Neural Network applications.

IV. Uncertainty-aware AUV Localization and Retraining

While many state-of-the-art studies have demonstrated the effectiveness of BNNs for UQ in non-robotic applications, such as image classification [34] and medical analysis [35], end-to-end evaluations in robotic perception remain largely unexplored. In this section, we demonstrate the efficacy of the proposed BNN-based uncertainty-aware approach for AUV localization, showcasing its capabilities in uncertainty-driven sampling and retraining.

Localization for AUVs poses significant challenges compared to mobile or aerial robots due to the difficulty of underwater data collection (e.g., noisy, unstructured environments) and labeling (e.g., surfacing for GPS signals). Additionally, constrained battery life and computational resources necessitate efficient hardware.

In the simulation scenario (Fig. 9(a)), the AUV is equipped with an onboard camera for perception. Given a detectable landmark in the deployment area (e.g., a sunken submarine), the AUV uses camera-captured images (Fig. 9(b)) to estimate its position. The vehicle operates in a 2D horizontal domain, maintaining a constant depth. The localization task involves

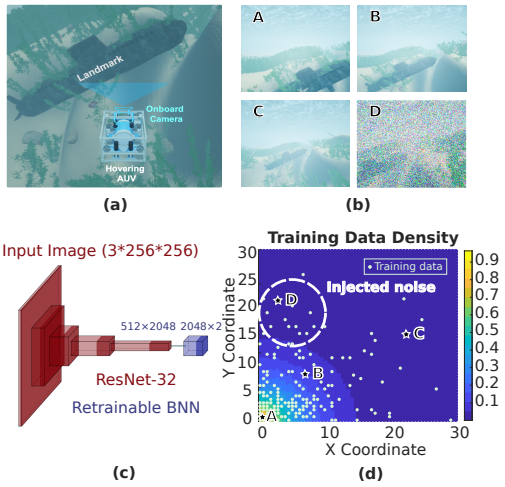


Fig. 9: (a) The AUV uses its onboard camera to capture visual observations of a landmark (the sunken submarine). (b) Test dataset with sampled regions removed from the training dataset. (c) Structure of the proposed retrainable BNN with a fine-tuned ResNet-32 backbone. (d) Probability map generated based on the distance from the origin, used to guide region sampling and create a training dataset.

training a deep learning model to estimate position and quantify uncertainty.

This approach employs ResNet-32 trained on other image tasks as a fixed feature extractor and a two-layer BNN as a classifier and uncertainty quantifier (Fig. 9(c)).

According to nature of the task, we assume training data is denser near a known landmark (origin) and sparser further away (Fig. 9(d)). This limited spatial coverage introduces uncertainty in localization estimates. To improve accuracy, the AUV must actively explore regions not covered in the training data and use real-time observations to retrain the model. However, labeling online data is power-intensive since the AUV must surface for GPS signals. To address this, we introduce an information-theoretic method that selects the most informative samples, enabling efficient data collection and retraining while minimizing energy consumption.

To select the most informative and spatially diverse data points, we define the scoring function $S_i = IG + \alpha \cdot (1 - p_i)$, where IG quantifies the expected reduction in model uncertainty from retraining with the i-th point, and p_i is a spatial penalty discouraging selection of points near previously chosen ones. The weighting parameter α balances information gain and spatial diversity. The spatial penalty p_i is given by $p_i = \exp\left(-\frac{\|\mathbf{r}_i - \mathbf{r}_{\text{nearest}}\|}{\lambda}\right)$,where $\mathbf{r}_i$ is the position of the i-th point, $\mathbf{r}_{\text{nearest}}$ is the closest previously selected point, and λ controls the penalty's decay with distance. By iteratively selecting points with the highest S_i, this method ensures the chosen samples maximize information gain while exploring new regions, reducing redundancy and improving data efficiency.

V. Evaluation

For the AUV task, we evaluate the BNN's ability to: (1) detect epistemic uncertainty in regions with sparse training data, (2) identify aleatoric uncertainty under high sensor

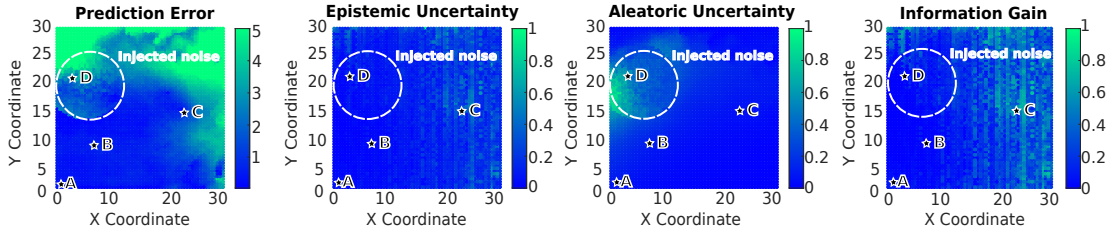


Fig. 10: Illustration of two comparative experiments, with the second experiment incorporating gradually decreasing Gaussian noise within a localized region. Heatmaps display the error distribution across the exploration area, including maps for aleatory and epistemic uncertainties, as well as the resulting information gain. Additionally, the input image from the origin and images from three corners with high information gain are shown to highlight regions of significant insight.

noise, and (3) demonstrate data efficiency in uncertainty-aware retraining compared to conventional convolutional neural networks (CNNs) without BNN layers.

Four representative locations (A, B, C, and D) are selected to illustrate various uncertainty scenarios (Fig. 9(b),(d)). Location A represents the closest point to the landmark, B has a clear view of the landmark, C is a distant location with OOD data, and D contains artificially injected sensor noise.

Figure 10 shows the prediction error, epistemic uncertainty, aleatoric uncertainty, and information gain across the exploration terrain. Prediction error increases with distance from the origin due to sparse training data. Locations A and B exhibit low error because of rich training data and absence of sensor noise. At location C, despite no sensor noise, sparse training data results in high error and epistemic uncertainty. At location D, injected noise increases aleatoric uncertainty without significantly affecting epistemic uncertainty. Information gain correlates strongly with prediction error, highlighting its potential for uncertainty-aware sampling and retraining.

We then apply the resampling method introduced in section IV, using information gain to selecting new data to retrain the AUV localization model, and compare the results with a CNN retrained using random sampling. Fig. 11(a) contrasts random sampling with uncertainty-aware sampling, showing the latter focuses on regions with sparse training data. Comparing CNNs and CNNs with BNN layers, we observe that BNNs outperform CNNs even without resampling. Moreover, uncertainty-aware resampling and retraining with BNNs significantly reduce error, a capability not matched by CNNs (Fig. 11(b)). Additionally, we considered an 8 % device variation, modeled

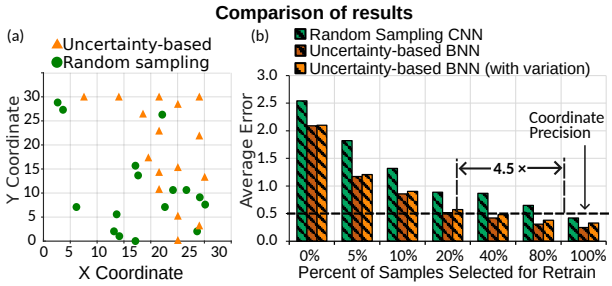


Fig. 11: Comparison of two resampling methods for retraining. (a) New data acquired by each method, with uncertainty-based sampling focusing on sparse data regions. (b) Average error across different retraining data percentages for CNN with random sampling, BNN with uncertainty-based sampling, and BNN with added variation, showing their comparative performance.

through Monte Carlo simulations based on SPICE models., which resulted in only a slight decrease in final prediction accuracy.

Finally, Fig. 11.(b) illustrates the data efficiency of uncertainty-aware BNN resampling compared to random sampling with CNNs. At a target average error of 0.5 (sample coordiate minimum step), BNNs achieve the same accuracy with significantly fewer resamples, saving 4.5× the data.

	[6]	[8]	This Work
Technology	22nm	65nm	22nm (FD-SOI)
Energy efficiency (TOPS/W)	32.3	12	22.2
RNG efficiency (pJ/Sa)	1.08	0.1	0.2
RNG throughput (GSa/s)	7.31	1	20.9 (2.9×)
Area (10^9 F^2)	30	1.8	3.2 (1.77×)
Latency (us)	2250	20000	1.2 (10^3×)

TABLE II: Comparison of the proposed work with state-of-the-art RNG accelerators in terms of technology, efficiency, throughput, energy, area, and latency. F is the transistor feature dimension.

Hardware Benchmark: We benchmarked the parallel mixed-signal BNN engine against state-of-the-art digital [6] and mixed-signal [8] BNN accelerators using SPICE simulations. For BNN layers with 512 input neurons, 2048 output neurons, and 20 weight samples, our design achieves a 10^3–10^4× reduction in latency and a 2.9× increase in GRNG throughput. While on-chip parallelization results in a 1.7× larger area compared to mixed-signal designs, it remains significantly more compact than digital counterparts. Additionally, the engine demonstrates state-of-the-art GRNG efficiency (0.2 pJ/Sa) and computation energy efficiency (22.2 TOPS/W).

VI. Conclusion

This work introduces a mixed-signal BNN accelerator leveraging FD-SOI quantum tunneling to address challenges in energy efficiency, compactness, and real-time uncertainty quantification for robotic perception. By designing a compact and energy-efficient GRNG, we enable high-quality randomness and a parallel BNN sampling architecture, achieving significant latency reductions with minimal area overhead. Furthermore, we propose a hardware-algorithm co-design framework, validated in AUV localization and retraining tasks, demonstrating reduced retraining data requirements and labeling costs. This approach offers a scalable and efficient solution for robotic applications requiring real-time, uncertainty-aware processing.

REFERENCES

[1] Abhishek Gupta and Xavier Fernando. Simultaneous localization and mapping (slam) and data fusion in unmanned aerial vehicles: Recent advances and challenges. *Drones*, 6(4):85, 2022.

[2] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[3] Wenchong He, Zhe Jiang, Tingsong Xiao, Zelin Xu, and Yukun Li. A survey on uncertainty quantification methods for deep learning. *arXiv preprint arXiv:2302.13425*, 2023.

[4] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. Cnn-slam: Real-time dense monocular slam with learned depth prediction. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6243–6252, 2017.

[5] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? *Advances in neural information processing systems*, 30, 2017.

[6] Richard Dorrance, Deepak Dasalukunte, Hechen Wang, Renzhi Liu, and Brent Carlton. An energy-efficient bayesian neural network accelerator with cim and a time-interleaved hadamard digital grng using 22-nm finfet. *IEEE Journal of Solid-State Circuits*, 2023.

[7] Yudeng Lin, Qingtian Zhang, Jianshi Tang, Bin Gao, Chongxuan Li, Peng Yao, Zhengwu Liu, Jun Zhu, Jiwu Lu, Xiaobo Sharon Hu, et al. Bayesian neural network realization by exploiting inherent stochastic characteristics of analog rram. In *2019 IEEE International Electron Devices Meeting (IEDM)*, pages 14–6. IEEE, 2019.

[8] Likai Pei, Yifan Qin, Zephan M. Enciso, Boyang Cheng, Jianbo Liu, Steven Davis, Zhenge Jia, Michael Niemier, Yiyu Shi, X. Sharon Hu, and Ningyuan Cao. Towards uncertainty-quantifiable biomedical intelligence: Mixed-signal compute-in-entropy for bayesian neural networks. In *2024 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2024.

[9] Djohan Bonnet, Tifenn Hirtzlin, Atreya Majumdar, Thomas Dalgaty, Eduardo Esmanhotto, Valentina Meli, Niccolo Castellani, Simon Martin, Jean-François Nodin, Guillaume Bourgeois, et al. Bringing uncertainty quantification to the extreme-edge with memristor-based bayesian neural networks. *Nature Communications*, 14(1):7530, 2023.

[10] Andrea Bonci, Pangcheng David Cen Cheng, Marina Indri, Giacomo Nabissi, and Fiorella Sibona. Human-robot perception in industrial environments: A survey. *Sensors*, 21(5):1571, 2021.

[11] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[12] Nan Yang, Lukas von Stumberg, Rui Wang, and Daniel Cremers. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1281–1292, 2020.

[13] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[14] Liulong Ma, Jiao Chen, et al. Using rgb image as visual input for mapless robot navigation. *arXiv preprint arXiv:1903.09927*, 2019.

[15] Anca Morar, Alin Moldoveanu, Irina Mocanu, Florica Moldoveanu, Ion Emilian Radoi, Victor Asavei, Alexandru Gradinaru, and Alex Butean. A comprehensive survey of indoor localization methods based on computer vision. *Sensors*, 20(9):2641, 2020.

[16] Niclas Vödisch, Ozan Unal, Ke Li, Luc Van Gool, and Dengxin Dai. End-to-end optimization of lidar beam configuration for 3d object detection and localization. *IEEE Robotics and Automation Letters*, 7(2):2242–2249, 2022.

[17] Huan Yin, Runjian Chen, Yue Wang, and Rong Xiong. Rall: end-to-end radar localization on lidar map using differentiable measurement model. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6737–6750, 2021.

[18] Kyle Harlow, Hyesu Jang, Timothy D Barfoot, Ayoung Kim, and Christoffer Heckman. A new wave in robotics: Survey on recent mmwave radar applications in robotics. *IEEE Transactions on Robotics*, 2024.

[19] Guanya Shi, Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, Fabio Ramos, Animashree Anandkumar, and Yuke Zhu. Fast uncertainty quantification for deep object pose estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5200–5207. IEEE, 2021.

[20] Danijar Hafner, Dustin Tran, Timothy Lillicrap, Alex Irpan, and James Davidson. Noise contrastive priors for functional uncertainty. In *Uncertainty in Artificial Intelligence*, pages 905–914. PMLR, 2020.

[21] Kumar Shridhar, Felix Laumann, and Marcus Liwicki. A comprehensive guide to bayesian convolutional neural network with variational inference. *arXiv preprint arXiv:1901.02731*, 2019.

[22] Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, et al. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, 56(Suppl 1):1513–1589, 2023.

[23] Jianxiang Feng, Jongseok Lee, Maximilian Durner, and Rudolph Triebel. Bayesian active learning for sim-to-real robotic perception. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10820–10827. IEEE, 2022.

[24] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American statistical Association*, 112(518):859–877, 2017.

[25] Dong-U Lee, Wayne Luk, John D Villasenor, Guanglie Zhang, and Philip Heng Wai Leong. A hardware gaussian noise generator using the wallace method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(8):911–920, 2005.

[26] David B Thomas and Wayne Luk. Multiplierless algorithm for multivariate gaussian random number generation in fpgas. *IEEE transactions on very large scale integration (VLSI) systems*, 21(12):2193–2205, 2013.

[27] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Xinyu Niu, Miguel Rodrigues, and Wayne Luk. Accelerating bayesian neural networks via algorithmic and hardware optimizations. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):3387–3399, 2022.

[28] Jiachen Xu, Yuyi Shen, Ethan Chen, and Vanessa Chen. Bayesian neural networks for identification and classification of radio frequency transmitters using power amplifiers' nonlinearity signatures. *IEEE Open Journal of Circuits and Systems*, 2:457–471, 2021.

[29] Steven Davis, Jianbo Liu, Boyang Cheng, Muya Chang, and Ningyuan Cao. In-situ privacy via mixed-signal perturbation and hardware-secure data reversibility. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2024.

[30] Hongxiang Fan, Martin Ferianc, Zhiqiang Que, Shuanglong Liu, Xinyu Niu, Miguel RD Rodrigues, and Wayne Luk. Fpga-based acceleration for bayesian convolutional neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(12):5343–5356, 2022.

[31] Xuelian Liu, Aamir Zia, Mitchell R. LeRoy, Srikumar Raman, Ryan Clark, Russell Kraft, and John F. McDonald. A three-dimensional dram using floating body cell in fdsoi devices. In *2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pages 159–162, 2012.

[32] Stefan Dünkel, Martin Trentzsch, Regina Richter, Patrick Moll, Christine Fuchs, Oliver Gehring, Mateusz Majer, Stefan Wittek, B Müller, Thomas Melde, et al. A fefet based super-low-power ultra-fast embedded nvm technology for 22nm fdsoi and beyond. In *2017 IEEE International Electron Devices Meeting (IEDM)*, pages 19–7. IEEE, 2017.

[33] Sophie Puget, Germain Bossu, Claire Fenouiller-Beranger, Pierre Perreau, Pascal Masson, Pascale Mazoyer, Philippe Lorenzini, Jean-Michel Portal, Rachid Bouchakour, and Thomas Skotnicki. Fdsoi floating body cell edram using gate-induced drain-leakage (gidl) write current for high speed and low power applications. In *2009 IEEE International Memory Workshop*, pages 1–2, 2009.

[34] Zain Ul Abideen, Mubeen Ghafoor, Kamran Munir, Madeeha Saqib, Ata Ullah, Tehseen Zia, Syed Ali Tariq, Ghufran Ahmed, and Asma Zahra. Uncertainty assisted robust tuberculosis identification with bayesian convolutional neural networks. *Ieee Access*, 8:22812–22825, 2020.

[35] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. Uncertainty quantification using bayesian neural networks in classification: Application to ischemic stroke lesion segmentation. *Medical Imaging with Deep Learning*, 4:2, 2018.

CaMDN: Enhancing Cache Efficiency for Multi-tenant DNNs on Integrated NPUs

Tianhao Cai[1,2], Liang Wang[1,2,*], Limin Xiao[1,2,*], Meng Han[3], Zeyu Wang[1,2], Lin Sun[4] and Xiaojian Liao[1,2]
[1] State Key Laboratory of CCSE, Beihang University, Beijing, China
[2] School of Computer Science and Engineering, Beihang University, Beijing, China
[3] School of Integrated Circuits, Tsinghua University, Beijing, China
[4] Jiangsu Shuguang Optoelectric Co., Ltd., Yangzhou, China
caitianhao@buaa.edu.cn, lwang20@buaa.edu.cn, xiaolm@buaa.edu.cn

***Abstract*—With the rapid development of DNN applications, multi-tenant execution, where multiple DNNs are co-located on a single SoC, is becoming a prevailing trend. Although many methods are proposed in prior works to improve multi-tenant performance, the impact of shared cache is not well studied. This paper proposes CaMDN, an architecture-scheduling co-design to enhance cache efficiency for multi-tenant DNNs on integrated NPUs. Specifically, a lightweight architecture is proposed to support model-exclusive, NPU-controlled regions inside shared cache to eliminate unexpected cache contention. Moreover, a cache scheduling method is proposed to improve shared cache utilization. In particular, it includes a cache-aware mapping method for adaptability to the varying available cache capacity and a dynamic allocation algorithm to adjust the usage among co-located DNNs at runtime. Compared to prior works, CaMDN reduces the memory access by 33.4% on average and achieves a model speedup of up to 2.56× (1.88× on average).**

***Index Terms*—Cache, Multi-tenant, DNN, NPU, SoC.**

I. Introduction

Deep Neural Networks (DNNs) are increasingly pervasive in various applications. This brings an inevitable demand for efficient multi-tenant DNN execution, where many models are co-located on a single SoC to improve the performance of comprehensive DNN applications. Therefore, modern SoCs tend to integrate Neural Processing Units (NPUs) alongside CPUs to support trending multi-DNN applications [1]–[3].

The key challenge of co-locating DNNs is the intense contention among running tasks over all kinds of shared resources. Methods are proposed to schedule different resources among models through either temporal interleaving [4]–[6] or spatial partitioning [7]–[14]. However, these methods mainly focus on memory bandwidth [7], [8], NPU cores [9], [10], or both [11]–[13]. The performance impact of shared cache with multi-tenant DNNs is still not well studied.

Cache efficiency can be severely compromised in multi-tenancy due to the contention among running programs. Our experiment shows that cache hit rate drops by up to 59.7%, and memory access increases by up to 64.1% with 32 co-located DNNs, as shown in Section II-C. Since multi-tenant DNNs are generally memory-bound, the increase in memory access directly results in severe model slowdown. According to our statistical analyses, a large portion of data in multi-tenant DNNs is either non-reusable or reused over long distances on shared cache, which is a primary cause of the inefficiency.

Prior works manage to improve cache efficiency with different types of multi-tenant workloads other than DNNs, such as CPU workloads [15]–[18] and accelerator workloads [19]–[21]. However, the performance improvements brought by these methods are limited due to either the need to keep cache transparent to workloads, or lack of consideration on special characteristics of trending multi-tenant DNNs.

For multi-tenant DNNs, both architecture and scheduling require specialized optimization to improve shared cache efficiency. In terms of architecture, cache is required to be able to bypass non-reusable data and retain data with long reuse distances. In terms of scheduling, model mapping is required to be aware that cache will be shared among multiple models, and resource allocation is required to dynamically respond to changes of cache usage during multi-tenant execution.

Therefore, this paper proposes CaMDN, an architecture-scheduling co-design to enhance cache efficiency for multi-tenant DNNs on integrated NPUs. Specifically, CaMDN introduces a lightweight architecture to support model-exclusive, NPU-controlled regions inside shared cache. This allows NPUs to manage data storage and replacement on cache and eliminates unexpected contention among models. To fully utilize the architectural advantages, CaMDN provides a cache scheduling method including cache-aware mapping and dynamic cache allocation. In particular, cache-aware mapping generates multiple mapping candidates that target different cache usage levels to provide adaptability to the varying available capacity. Dynamic cache allocation adjusts cache usage among co-located DNNs at runtime. It responds quickly to changes of cache usage to improve cache utilization. In summary, this paper makes the following main contributions:

- We propose a lightweight architecture to support model-exclusive, NPU-controlled regions inside shared cache.
- We propose a cache scheduling method that contains cache-aware DNN mapping and dynamic cache allocation to improve shared cache utilization.
- We demonstrate that, compared to prior works, CaMDN reduces the memory access by 33.4% and achieves a model speedup of up to 2.56× (1.88× on average).

This work was supported by the National Key R&D Program of China under Grant No. 2023YFB4503100, the National Natural Science Foundation of China under Grant No. 62272026 and No.62104014, the Fundamental Research Funds for the Central Universities, State Key Laboratory of Complex & Critical Software Environment under Grant No. CCSE-2024ZX-10, and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

* Liang Wang and Limin Xiao are corresponding authors of this paper.

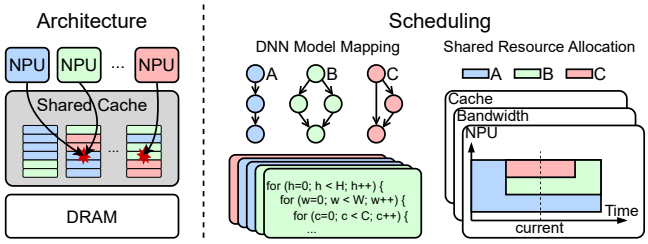


Fig. 1: An overview of architecture and scheduling for multi-tenant DNN execution on integrated NPUs. Cache contention occurs between co-located DNNs.

II. Background and Motivation

A. Multi-tenant DNN Execution

As DNNs increasingly pervade various applications, multi-tenant execution becomes an inevitable requirement for multiple reasons. First, complex applications such as AR/VR are likely to apply more than one DNN. Second, multiple DNN applications may run on a single device like personal computers and edge devices [22]. Third, cloud servers need to support multi-tenancy for modern AI services [23].

Figure 1 illustrates an overview of architecture and scheduling for multi-tenant DNN execution on a single SoC. Architecturally, to meet the trending demand for multi-tenant DNNs, the chip industry has started to integrate NPUs alongside CPUs on their SoCs [1]–[3]. Such architectures leverage the benefits of shared memory subsystem, including higher interconnect bandwidth, reduced host-device data movement, and additional on-chip storage provided by shared cache.

Scheduling refers to the process of deploying DNN models on target hardware. In particular, model mapping is to generate hardware-specific programs from high-level model descriptions. Mapping is demonstrated to have a significant impact on model performance, and many optimizations are proposed in prior works [24]–[26]. At runtime, shared resources, such as NPUs and bandwidth, are allocated to models according to their requirements as well as target deadlines.

B. Scheduling Multi-tenant DNNs

Recently, methods are proposed in prior works to schedule multi-tenant DNNs on integrated NPUs. Some of them use temporal interleaving to execute tasks in turn [4]–[6]. These methods suffer from under-utilization as resources scale richer. Others use spatial partitioning to fully utilize shared resources. We categorize these methods according to the resources they mainly focus on.

1) Bandwidth: MAGMA [7] and MoCA [8] allocate memory bandwidth usage among co-located DNNs according to their memory access requirements.

2) Compute: Planaria [9] manages to co-locate multiple models on a single large systolic array-based NPU. DREAM [10] dynamically schedules multiple NPUs for multi-tenant DNNs in real-time scenarios.

3) Bandwidth & Compute: RELMAS [11] considers bandwidth contention for NPU scheduling but does not employ any hardware constraints on bandwidth usage. HDA [12] and AuRORA [13] leverage static and dynamic scheduling methods, respectively, to co-allocate bandwidth and NPUs.

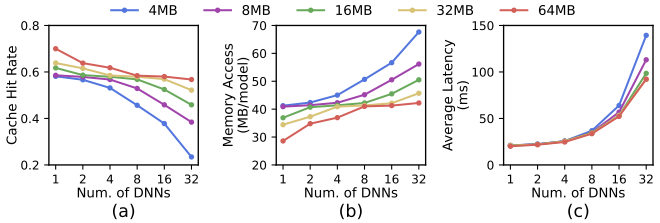


Fig. 2: Cache hit rate, memory access and average latency with different settings on number of DNNs and cache capacity.

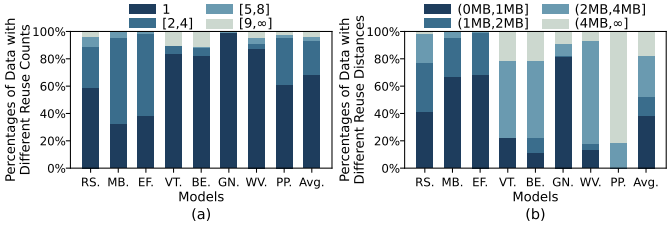


Fig. 3: Percentages of data with different reuse counts and reuse distances on shared cache in benchmark DNN models.

Prior works above manage to schedule memory bandwidth and/or NPUs to achieve higher performance. However, they fail to consider the performance impact of shared cache with multi-tenant DNNs. Veltair [14] uses cache performance counters for contention monitoring, but does not utilize any hardware method to address the issue. Additionally, Veltair targets server-level many-core CPUs rather than NPUs.

C. Cache Inefficiency with Multi-tenant DNNs

To understand the performance impact of shared cache with multi-tenant DNNs, we conduct an experiment that randomly dispatches different DNN models on an NPU-integrated SoC. The experiment is performed with different settings on the number of co-located models and shared cache capacity. The detailed experimental setup is described in Section IV-A. Figure 2 shows the results of the experiment. The cache hit rate drops by 18.9% to 59.7%, and the memory access increases by 32.7% to 64.1% as the number of DNNs reaches 32. This is due to the severe cache contention among models, which occurs when a process frequently replaces cache lines belonging to other co-running processes, as illustrated in the left part of Figure 1. As a result, the average model latency increases by 3.46× to 5.65×. Since multi-tenant DNNs are generally memory-bound, the increase in memory access directly contributes to the slowdown.

To further understand how DNN workload characteristics incur the inefficiency of shared cache, we perform the following statistical analyses on benchmark DNNs in Table I. Figure 3(a) shows the percentages of data with different reuse counts. A reuse count refers to the expected number of repeated cache accesses to a piece of data. On average, 68.0% of data have no future reuse but still occupy cache space, which results in a large amount of reusable data being early replaced off-chip. Figure 3(b) shows the percentages of intermediate data with different reuse distances when accessed across DNN layers. A reuse distance refers to the amount of accesses to other data that occur between two repeated accesses to the same piece of data. On average, 61.8% of intermediate data have reuse

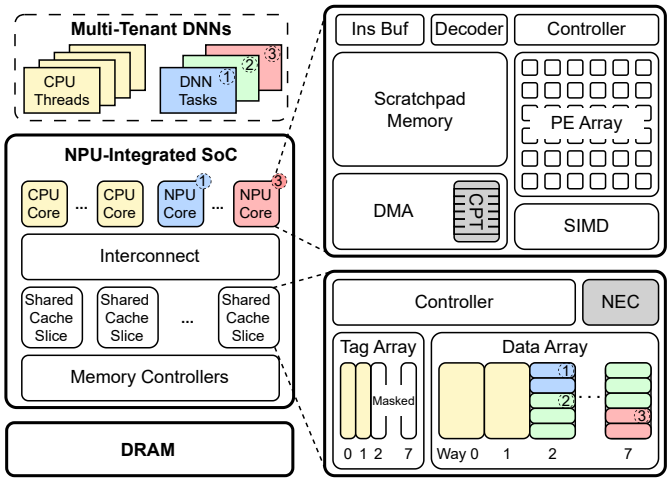


Fig. 4: The overview of CaMDN architecture. CaMDN installs an NEC in each shared cache slice and a CPT in each NPU.

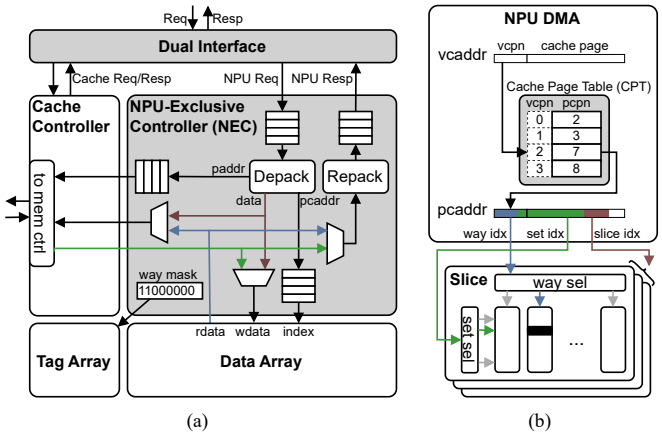


Fig. 5: Implementation details of CaMDN architecture. (a) NEC handles NPU-specific requests. (b) CPT translates *vcaddr* into *pcaddr* that indexes the targeted cache line in data arrays.

distances more than 1MB, and 47.9% of them more than 2MB. Such large reuse distances make it difficult for transparent cache to retain these data for future reuse, especially when multiple models are competing for cache space. In summary, a large portion of data in DNNs is either non-reusable or reused over long distances. Shared cache shows evident inefficiency to handle such data, therefore significantly degrading the performance of multi-tenant DNN execution.

D. Enhancing Cache Efficiency

Prior works manage to enhance cache efficiency for different multi-tenant workloads other than DNNs. For CPU workloads, cache partitioning and related techniques are proposed to limit the cache usage of each process [15]–[18]. However, these methods offer limited performance improvement, since cache must be hardware-managed and transparent to CPU workloads. For accelerator workloads, methods are purposed that either instantiate accelerator-controlled buffers inside shared cache [19], [20], or dynamically change cache coherence mode for each workload at runtime [21]. However, these methods lack consideration for the special characteristics of multi-tenant DNNs.

For multi-tenant DNNs, both architecture and scheduling require specialized optimization to improve shared cache efficiency. In terms of architecture, cache is required to be able to bypass non-reusable data and retain data with long reuse distances. In terms of scheduling, mapping is required to be aware that cache will be shared among multiple models, and resource allocation is required to dynamically respond to changes of cache usage during multi-tenant execution. Therefore, this paper proposes CaMDN, an architecture-scheduling co-design to enhance cache efficiency for multi-tenant DNNs on integrated NPUs. Specifically, CaMDN contains a lightweight architecture to support model-exclusive, NPU-controlled regions inside shared cache, a cache-aware DNN mapping method for adaptability to the varying available cache capacity, and a dynamic cache allocation algorithm to adjust the cache usage among co-located DNNs at runtime.

III. CaMDN System

A. Overview

The architecture-scheduling co-design of CaMDN contains the following components. (1) An NPU-controlled cache architecture that supports model-exclusive, NPU-controlled regions inside shared cache by adding an NPU-exclusive controller (NEC) in each cache slice and a cache page table (CPT) in each NPU (Section III-B). (2) A cache-aware mapping method that provides adaptability to varying available cache size by generating multiple mapping candidates that target different cache usage levels (Section III-C). (3) A dynamic cache allocation algorithm that adjusts cache usage among models by predicting near-future cache usage and selecting mapping candidates accordingly (Section III-D).

B. NPU-controlled Cache Architecture

The NPU-controlled cache architecture of CaMDN supports the instantiation of model-exclusive, NPU-controlled regions inside shared cache. Specifically, (1) shared cache is divided into a general-purpose subspace and an NPU subspace by way partitioning. (2) An NEC is added in each cache slice to take control of the NPU subspace and provide NPU-controlled cache access semantics. (3) The NPU subspace is then divided into model-exclusive regions by paging, which is implemented by a hardware-based CPT inside each NPU.

1) Way-Partitioned NPU Subspace: CaMDN divides shared cache into a general-purpose subspace and an NPU subspace by way partitioning. This eliminates unexpected contention between CPU applications and NPU tasks. CaMDN implements way partitioning by adding a way mask register in each cache slice to mask off the ways allocated to the NPU subspace. As shown in the bottom-right part of Figure 4, ways 0-1 are reserved for CPU applications, and ways 2-7 are assigned to the NPU subspace. Different proportions of partitioning can be adapted for different application scenarios.

2) NPU-controlled Access: To support NPU-controlled cache access, CaMDN adds a lightweight NEC and a dual interface in each cache slice to take control of the NPU subspace, as shown in Figure 5(a). The dual interface provides

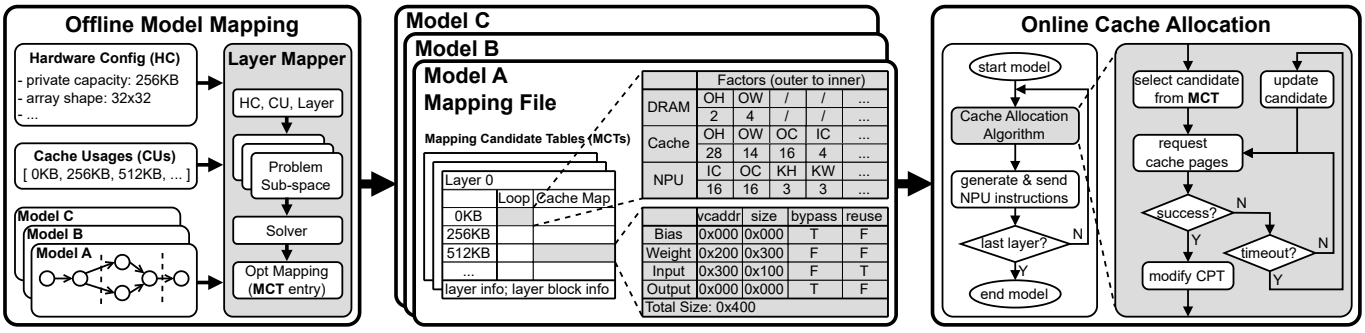


Fig. 6: CaMDN cache scheduling includes offline cache-aware mapping and online cache allocation.

a uniform interface for normal cache requests and NPU-specific requests. NEC handles different types of NPU requests and performs operations including reading/writing cache lines and generating memory requests to memory controllers.

NEC implements NPU-controlled cache access semantics with cache line granularity. Replacing the original hardware-managed cache control, these semantics provide NPU programs with full control over data movement, storage and replacement. In addition to basic semantics which implement memory-cache and cache-NPU data movements, NEC introduces advanced semantics for performance improvement: bypass and multicast. Bypass semantics support bypassing non-reusable data around shared cache, reserving cache space for reusable data to improve utilization. This includes (1) **bypass-read** a line from memory to NPU; (2) **bypass-write** a line from NPU to memory. Multicast semantics help reduce bandwidth pressure on both memory and NoC by combining identical requests from a group of NPUs that are allocated to the same model. This includes (3) **multicast-read** a line from cache to a group of NPUs; (4) **multicast-bypass-read** a line from memory to a group of NPUs.

3) Virtual Cache Addressing: To further make the NPU subspace divisible and addressable for co-located models, CaMDN utilizes hardware-based paging to create isolated model-exclusive regions and provide an independent virtual address space for every model. As shown in the bottom-right part of Figure 4, the NPU subspace is divided into pages of identical size and assigned to different models. A hardware-based CPT is installed in each NPU to translate virtual cache address (*vcaddr*) into physical cache address (*pcaddr*). Considering that NPUs usually operate on data chunks of at least dozens of KB, we set the page size to 32KB for a 16MB shared cache. Thus, a hardware-based CPT has at most 512 entries, each of which needs at most 3 bytes to store a physical cache page number (*pcpn*) and a valid bit, resulting in a total 1.5KB SRAM overhead, which is negligible as demonstrated in Section IV-B4.

Figure 5(b) illustrates the address translation and cache line indexing. The virtual cache page number (*vcpn*) of a *vcaddr* is used to index the CPT for *pcpn*, which is then used to compose *pcaddr*. *pcaddr* is divided into four bit-fields: byte offset, slice index, set index and way index, from lower to higher, which uniquely determine the targeted cache line. In this indexing manner, consecutive data lines are distributed among all slices for higher cache bandwidth utilization.

C. Cache-aware Mapping

The cache-aware mapping method of CaMDN provides adaptability to varying available cache capacity by generating multiple mapping candidates that target different cache usage levels. As shown in the left part of Figure 6, it takes a hardware configuration, a list of cache usage limitations, and model structures as inputs. For each layer, the layer mapper generates a mapping candidate within each usage limitation. All candidates of the layer are written into a mapping candidate table (MCT) and all MCTs of the model are stored in a model mapping file as the output of the mapping phase.

1) Heuristic-solver-hybrid Layer Mapper: To reduce the additional time cost of generating multiple mapping candidates, CaMDN introduces a heuristic-solver-hybrid layer mapper that combines heuristic rules and an available optimization solver to speed up the mapping process while maintaining high performance of results. Specifically, it first shrinks the problem space according to a set of heuristic rules. These rules improve the utilization of cache line, NPU-private storage and compute resource, and reduce the choices of loop permutation [26]. Second, it constructs a set of disjoint problem subspaces, each of which is an integer programming problem that takes minimal DRAM access as the optimization objective. Third, these problems are solved by the solver, and the result with the minimal DRAM access is selected as the mapping candidate for the layer within a certain cache usage limitation.

2) Layer-block Mapping for Data Reuse: In addition to layer-wise mapping (LWM) mentioned above, CaMDN also uses layer-block mapping (LBM) to store intermediate data between layers fully in cache and allocate zero memory space to these data. LBM significantly reduces memory access during multi-tenant DNN execution. To prevent a model from occupying too much cache space for too long, models are segmented into layer blocks and LBM happens only inside each block. For each layer, an additional LBM candidate is generated alongside multiple LWM candidates.

3) Mapping Candidate Table: Mapping candidate tables (MCTs) are the major outputs of the offline mapping phase. An MCT records all mapping candidates and detailed information of a layer in a compact format instead of unrolled NPU instructions. In this way, much less storage is needed for multiple mapping candidates. As shown in the middle of Figure 6, for each candidate, there is a loop table that records loop permutations and factors, and a cache map table that indicates how tensors are mapped in vcaddr space.

Algorithm 1: Predict near-future shared cache usage and select mapping candidate accordingly for a layer.

Input: t_{cur}: current task with its model information;
MCT_{cur}: MCT for current layer of t_{cur}.
Output: M_{cur}: mapping candidate selected for t_{cur};
P_{cur}: shared cache pages required by M_{cur};
T_{ahead}: timeout threshold for waiting.
Data: T_{next}[numTasks]: profiling-based predicted next reallocation time for each task;
P_{next}[numTasks]: predicted pages needed at next reallocation for each task;
P_{alloc}[numTasks]: allocated pages for each task.

```
// Predict future available pages.
1  Func predAvailPages(T_ahead, t_cur) : P_ahead
2      P_ahead = idlePages();
3      for t_i : runningTasks do
4          if t_i ≠ t_cur and T_next[t_i] < T_ahead then
5              P_ahead += P_alloc[t_i] − P_next[t_i];
6      return P_ahead
// Check if LBM is available.
7  if hasEnabledLBM(t_cur) then
8      M_cur = MCT_cur.LBM;
9      return M_cur, M_cur.P_need, ∞;
10 else if isHeadLayerOfBlock(t_cur.layer_cur) then
11     T_ahead = T_cur + t_cur.layerBlock_cur.T_est × 0.2;
12     P_ahead = predAvailPages(T_ahead, t_cur);
13     if MCT_cur.LBM.P_need < P_ahead then
14         M_cur = MCT_cur.LBM;
15         return M_cur, M_cur.P_need, T_ahead;
// Select a LWM candidate from MCT.
16 T_ahead = T_cur + t_cur.layer_cur.T_est × 0.2;
17 P_ahead = predAvailPages(T_ahead, t_cur);
18 M_cur = MCT_cur.LWMs[0];
19 for M_i : MCT_cur.LWMs do
20     if M_cur.P_need < M_i.P_need ≤ P_ahead then
21         M_cur = M_i;
22 return M_cur, M_cur.P_need, T_ahead;
```

D. Dynamic Cache Allocation Algorithm

The dynamic cache allocation algorithm of CaMDN adjusts cache usage among co-located DNNs by predicting near-future cache usage and selecting mapping candidates accordingly.

As shown in the right part of Figure 6, the algorithm is invoked at the beginning of each layer. First, it predicts the cache usage among tasks in the near future, estimates the available capacity for the layer, and selects the mapping candidate that best fits the available capacity by Algorithm 1. Second, it tries to request the cache pages needed. If the pages become available within a threshold time, the CPTs are modified and the layer is executed with the selected mapping. Otherwise, every time a timeout occurs, it updates the candidate to the one that require fewer pages.

Algorithm 1 describes the detailed process of predicting shared cache usage and selecting a mapping candidate for a layer. The algorithm uses function predAvailPages to predict the available pages P_{ahead} in future time T_{ahead} for task t_{cur} (lines 1-6). The function is based on three global variables, T_{next}, P_{next} and P_{alloc}, which are updated at the end of each layer. The algorithm takes the current task t_{cur} and the current layer's MCT MCT_{cur} as inputs, and produces a mapping candidate M_{cur}, a number of required pages P_{cur}, and a timeout threshold T_{ahead} as outputs. Specifically, it first checks whether LBM is already enabled or can be enabled according to predicted near-future cache usage (lines 7-15). If available, LBM is enabled for this layer block. Otherwise, the LWM candidate that best fits the near-future available capacity is selected (lines 16-22). The timeout threshold T_{ahead} is calculated twice for both steps, using a profiling-based estimation of execution latency $t_{cur}.layer_{cur}.T_{est}$.

TABLE I: Benchmark Models for Multi-tenant Execution

Domain	Model	Abbr.	Type	QoS(ms)
Computer Vision	ResNet50 [27]	RS.	Conv	6.7
	MobileNet-v2 [28]	MB.	DwConv	2.8
	EfficientNet-b0 [29]	EF.	DwConv	2.8
	ViT-base-16 [30]	VT.	Trans	40.0
Natural Language Processing	BERT-base [31]	BE.	Trans	40.0
	GNMT [32]	GN.	LSTM	6.7
Audio Processing	Wav2Vec2-base [33]	WV.	Trans	16.7
Point Cloud	PointPillars [34]	PP.	Conv	100.0

TABLE II: Configuration of the NPU-Integrated SoC

Parameter		Value
NPU	PE Array (per core)	32x32
	Scratchpad (per core)	256KB
	Cores	16
Shared Cache	Total Capacity	16MB
	NPU Ways / Total Ways	12 / 16
	Slices	8
DRAM	Total Bandwidth	102.4GB/s
	Channels	4
Frequency		1GHz

IV. Evaluation

A. Experimental Setup

1) Implementation: The architecture of CaMDN is implemented based on Gemmini [35] in Verilog HDL, and synthesized by Synopsys Design Compiler in a 45nm process technology. OpenRAM [36] is used to generate SRAM macros. For performance evaluation, an in-door cycle-accurate simulator built on DRAMsim3 [37] is used, which is capable to simulate multi-tenant DNNs running on multiple NPUs with a sliced shared cache.

2) Benchmark: Table I shows the eight models chosen to compose the multi-tenant benchmark that feature various domains (CV, NLP, audio processing and point cloud) and different model types (Convolution, Depth-wise Convolution, Transformer and LSTM).

3) Baseline: MoCA [8] and AuRORA [13] are chosen as baseline: MoCA dynamically schedules memory bandwidth; AuRORA dynamically schedules NPUs and memory bandwidth. For fairness, MoCA and AuRORA are implemented with the same hardware configuration as CaMDN. Two versions of CaMDN are set for comparison: CaMDN(HW-only) equally allocates cache capacity among NPUs without dynamic cache scheduling; CaMDN(Full) refers to the full system of the architecture-scheduling co-design.

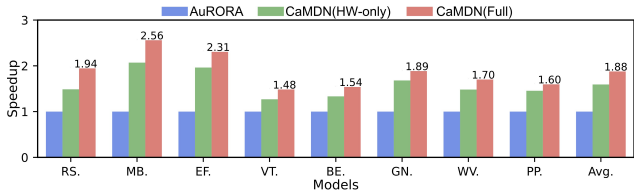


Fig. 7: The model-wise speedup achieved by CaMDN.

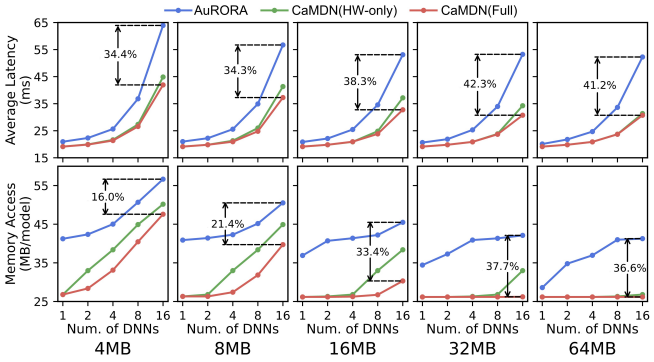


Fig. 8: The average latency and memory access with different scales. The numbers indicate the percentage reduction achieved by CaMDN(Full) compared to the baseline.

4) Experiments: To fully understand the performance improvement, we evaluate CaMDN from three aspects: speedup, scaling and quality-of-service(QoS).

Speedup experiment is conducted with configuration in Table II. we randomly dispatch each model task to one NPU as soon as it finishes its current task. All NPUs are kept busy to maximize the shared cache contention.

Scaling experiment is similar to speedup experiment, but with different cache sizes from 4MB to 64MB, and different numbers of co-located DNNs from 1 to 16.

QoS experiment is conducted with configuration in Table II. CaMDN is integrated with the same memory bandwidth and NPU allocation algorithms as AuRORA. We set tighter latency targets than prior work, as shown in Table I. Following [13], we setup three different QoS levels: QoS-H, QoS-M and QoS-L, respectively referring to 0.8×, 1.0× and 1.2× latency targets. The following metrics are used in comparison: Service Level Agreement (SLA) satisfaction rate indicates the percentage of tasks that meet their deadline; System Throughput (STP) indicates the overall throughput; Fairness indicates the equality degree of the progresses of co-running tasks. Detailed definitions are described in [13].

B. Results Analysis

1) Speedup: Figure 7 shows the model-wise speedup of CaMDN. Since the methods in MoCA and AuRORA are essentially for improving QoS rather than speedup, and they show similar results in this experiment, we choose AuRORA as a representative. CaMDN(Full) achieves an averagely 1.88× speedup, demonstrating that enhancing cache efficiency significantly improves the model speed. CaMDN(Full) surpasses CaMDN(HW-only) by an averagely 1.18× speedup, proving the effectiveness of dynamic cache allocation. CaMDN reaches higher speedup on MobileNet-v2 and EfficientNet-b0 because these models possess larger proportions of intermediate data and CaMDN successfully reduces the memory access by storing intermediate data entirely in cache.

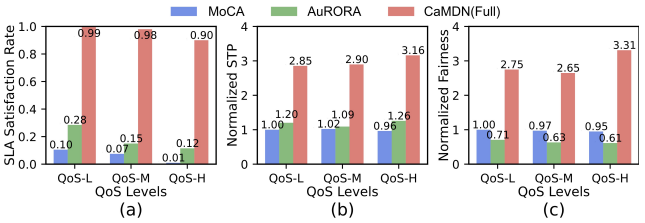


Fig. 9: The QoS improvement achieved by CaMDN.

TABLE III: Area Breakdown of the Architecture of CaMDN

Component	Area(μm^2)	(%)	Component	Area(μm^2)	(%)
NPU	7905k	100.0	Cache Slice	24676k	100.0
Scatchpad	6302k	79.7	Data Array	21878k	88.7
PE Array	1302k	16.5	Tag Array	2398k	9.7
CPT	**73k**	**0.9**	**NEC**	**66k**	**0.3**
others	228k	2.9	others	334k	1.3

2) Scaling: Figure 8 shows the average model latency and memory access with different system scales. Generally, CaMDN(Full) achieves 34.3% to 42.3% latency reductions and 16.0% to 37.7% memory access reductions. Since multi-tenant DNNs are memory-bound, the reduction in memory access contributes a significant portion of performance improvement. As the number of co-located DNNs increases, CaMDN significantly alleviates the performance degradation. As the cache becomes larger, CaMDN generally shows larger enhancement.

3) QoS: As shown in Figure 9, CaMDN achieves averagely 5.9×, 2.5× and 3.0× improvements on SLA satisfaction rate, STP and fairness respectively. This demonstrates that CaMDN can meet more rigorous QoS requirements by improving cache efficiency to reduce the model latency. Notably, CaMDN achieves even higher STP and fairness in QoS-H. The reason is that bandwidth and NPUs are more frequently allocated to slow tasks in QoS-H, while in QoS-L and QoS-M, the improvement of CaMDN largely reduces the need for bandwidth and NPU allocation. Additionally, AuRORA shows lower fairness than MoCA in our experiment, because our QoS targets are much tighter and AuRORA manages to reach higher SLA satisfaction rate at the cost of worsening fairness. Nonetheless, CaMDN can achieve higher performance without sacrificing fairness.

4) Area Overhead: Table III shows the negligible area overhead of an NPU and a cache slice with the configuration in Table II. A CPT contributes 0.9% of the total NPU area. An NEC contributes 0.3% of the total cache slice area.

V. Conclusion

We proposed CaMDN, an architecture-scheduling co-design to enhance cache efficiency for multi-tenant DNNs on integrated NPUs. Compared to prior works that focus on memory bandwidth and/or compute resources, CaMDN is dedicated to solving the inefficiency issue of cache shared by multiple NPUs. CaMDN combines a lightweight architecture, a cache-aware mapping method and a dynamic cache allocation algorithm. CaMDN reduces the memory access by 33.4% and achieves a model speedup of up to 2.56× (1.88× on average). For future work, we believe more efficient scheduling methods could be proposed that take both multi-tenant DNNs and general-purposed programs into consideration.

REFERENCES

[1] G. Williams, "Qualcomm oryon™ cpu," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–21.

[2] B. Cohen, M. Subramony, and M. Clark, "Next generation "zen 5" core," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–27.

[3] P. Mosur, "Built for the edge: The intel® xeon® 6 soc," in *2024 IEEE Hot Chips 36 Symposium (HCS)*. IEEE Computer Society, 2024, pp. 1–28.

[4] Y. H. Oh, S. Kim, Y. Jin, S. Son, J. Bae, J. Lee, Y. Park, D. U. Kim, T. J. Ham, and J. W. Lee, "Layerweaver: Maximizing resource utilization of neural processing units via layer-wise scheduling," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 584–597.

[5] Y. Choi and M. Rhu, "Prema: A predictive multi-task scheduling algorithm for preemptible neural processing units," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020, pp. 220–233.

[6] E. Baek, D. Kwon, and J. Kim, "A multi-neural network acceleration architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 940–953.

[7] S.-C. Kao and T. Krishna, "Magma: An optimization framework for mapping multiple dnns on multiple accelerator cores," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 814–830.

[8] S. Kim, H. Genc, V. V. Nikiforov, K. Asanović, B. Nikolić, and Y. S. Shao, "Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 828–841.

[9] S. Ghodrati, B. H. Ahn, J. K. Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim *et al.*, "Planaria: Dynamic architecture fission for spatial multi-tenant acceleration of deep neural networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 681–697.

[10] S. Kim, H. Kwon, J. Song, J. Jo, Y.-H. Chen, L. Lai, and V. Chandra, "Dream: A dynamic scheduler for dynamic real-time multi-model ml workloads," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Volume 4*, 2023, pp. 73–86.

[11] F. G. Blanco, E. Russo, M. Palesi, D. Patti, G. Ascia, and V. Catania, "A deep reinforcement learning based online scheduling policy for deep neural network multi-tenant multi-accelerator systems," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[12] H. Kwon, L. Lai, M. Pellauer, T. Krishna, Y.-H. Chen, and V. Chandra, "Heterogeneous dataflow accelerators for multi-dnn workloads," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 71–83.

[13] S. Kim, J. Zhao, K. Asanovic, B. Nikolic, and Y. S. Shao, "Aurora: Virtualized accelerator orchestration for multi-tenant workloads," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023, pp. 62–76.

[14] Z. Liu, J. Leng, Z. Zhang, Q. Chen, C. Li, and M. Guo, "Veltair: towards high-performance multi-tenant deep learning services via adaptive compilation and scheduling," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2022, pp. 388–401.

[15] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2006, pp. 423–432.

[16] D. Sanchez and C. Kozyrakis, "Vantage: Scalable and efficient fine-grain cache partitioning," in *Proceedings of the 38th annual international symposium on Computer architecture*, 2011, pp. 57–68.

[17] A. Herdrich, E. Verplanke, P. Autee, R. Illikkal, C. Gianos, R. Singhal, and R. Iyer, "Cache qos: From concept to reality in the intel® xeon® processor e5-2600 v3 product family," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 657–668.

[18] J. Park, S. Park, and W. Baek, "Copart: Coordinated partitioning of last-level cache and memory bandwidth for fairness-aware workload consolidation on commodity servers," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–16.

[19] C. F. Fajardo, Z. Fang, R. Iyer, G. F. Garcia, S. E. Lee, and L. Zhao, "Buffer-integrated-cache: A cost-effective sram architecture for handheld and embedded platforms," in *Proceedings of the 48th Design Automation Conference (DAC)*, 2011, pp. 966–971.

[20] J. Cong, M. A. Ghodrat, M. Gill, C. Liu, and G. Reinman, "Bin: A buffer-in-nuca scheme for accelerator-rich cmps," in *Proceedings of the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2012, pp. 225–230.

[21] J. Zuckerman, D. Giri, J. Kwon, P. Mantovani, and L. P. Carloni, "Cohmeleon: Learning-based orchestration of accelerator coherence in heterogeneous socs," in *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2021, pp. 350–365.

[22] A. Karatzas and I. Anagnostopoulos, "Omniboost: Boosting throughput of heterogeneous embedded devices under multi-dnn workload," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[23] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 1–14.

[24] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A systematic approach to dnn accelerator evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2019, pp. 304–315.

[25] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. S. Shao, "Cosa: Scheduling by constrained optimization for spatial accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 554–566.

[26] S. Dave, Y. Kim, S. Avancha, K. Lee, and A. Shrivastava, "Dmazerunner: Executing perfectly nested loops on dataflow accelerators," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–27, 2019.

[27] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4510–4520.

[29] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International Conference on Machine Learning (ICML)*. PMLR, 2019, pp. 6105–6114.

[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *9th International Conference on Learning Representations, ICLR 2021*. OpenReview.net, 2021.

[31] J. D. M.-W. C. Kenton and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of naacL-HLT*, vol. 1. Minneapolis, Minnesota, 2019, p. 2.

[32] Y. Wu, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[33] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, pp. 12 449–12 460, 2020.

[34] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 697–12 705.

[35] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 769–774.

[36] M. R. Guthaus, J. E. Stine, S. Ataei, B. Chen, B. Wu, and M. Sarwar, "Openram: An open-source memory compiler," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2016, pp. 1–6.

[37] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "Dramsim3: A cycle-accurate, thermal-capable dram simulator," *IEEE Computer Architecture Letters*, vol. 19, no. 2, pp. 106–109, 2020.

LearnGraph: A Learning-Based Architecture for Dynamic Graph Processing

Lingling Zhang*, Yijian Wu, Hong Jiang‡, Ziyu Zhou, Tiancheng Lu
Capital Normal University, ‡University of Texas at Arlington
*Corresponding author: 7089@cnu.edu.cn

***Abstract*—Dynamic graph processing systems using conventional array-based architectures face significant throughput limitations due to inefficient memory access and index management. While learned indexes improve data structure access, they struggle with interconnected graph data. We present LearnGraph, a novel architecture with an adaptive tree-based memory manager that dynamically optimizes for graph topology and access patterns. Our design integrates two key components: a hierarchical learned index optimized for graph topology to predict vertex and edge locations, and an adaptive tree structure that automatically reorganizes memory regions based on access patterns. Evaluation results demonstrate that LearnGraph outperforms state-of-the-art dynamic graph systems, achieving 3.4× higher throughput on average and reducing processing time by 1.7× to 11× across standard graph workloads.**

I. Introduction

Graphs are powerful tools for expressing object relationships through vertices and edges [1], [2]. With the exponential growth of dynamic graph data, traditional architectural approaches to graph storage and access are becoming increasingly inadequate [3], [4]. Efficient graph processing architectures are crucial for various applications [5], [6], from recommendation systems to natural language processing. Current graph processing architectures rely on conventional index structures using sorted arrays - for instance, Teseo [7] employs hash table-based adjacency lists, while Sortledton [8] uses compressed sparse row-like structures. However, these traditional architectures face fundamental limitations: first, maintaining sorted arrays becomes computationally prohibitive when processing millions of edge mutations per second [9]; second, accessing vertices or edges in large, dense arrays inherently incurs significant latency [10], [11]. These architectural constraints create performance bottlenecks in dynamic graph processing, suggesting the need for a fundamentally new approach to system design.

Learning-based architectures have emerged as a promising direction for overcoming traditional graph processing limitations, offering both space efficiency and rapid access performance [12], [13]. These architectures integrate learned models to predict data positions, replacing conventional index structures with intelligent position-mapping functions [14], [15]. By learning to map keys directly to memory locations within densely packed arrays, these architectures fundamentally reimagine data structure design. Compared to traditional B-tree architectures, learning-enhanced designs have demonstrated significant performance advantages, achieving at least 1.5× higher throughput on large datasets [16].

However, existing learning-based architectures prove inadequate for dynamic graphs due to two fundamental architectural limitations. First, these architectures are designed for independent data placement [17], [18], lacking mechanisms to optimize position prediction for interconnected data references. This architectural constraint fails to support efficient operations on connected data [19], which is crucial for graph applications where algorithms fundamentally depend on vertex relationships [2]. Second, current architectures employ static learning models [20], [21], making them unsuitable for dynamic graphs where vertex and edge distributions continuously evolve. These architectural limitations highlight the need for a new learning-enhanced system design specifically optimized for dynamic graph processing.

In this paper, we present LearnGraph, a novel graph processing architecture that integrates learned models into its fundamental data organization and access patterns. Our architecture departs from traditional sorted array structures by using learned models to predict and optimize data placement, access patterns, and memory management decisions. The system architecture is built around a learning-enhanced adaptive tree structure that continuously optimizes the trade-off between memory efficiency and access performance. Specifically, this paper makes the following contributions:

- **Learning-based hierarchical architecture:** LearnGraph employs a two-level learned model architecture - vertex position prediction and edge location optimization. The design combines lightweight neural models for vertex access with learned sparse arrays and bitmap-based indexing for edge position prediction, creating a learning-driven memory hierarchy.
- **Adaptive learning-guided memory management:** The architecture features a self-optimizing tree structure where learning models guide both data organization and access patterns. This learned tree maintains logical adjacency list views while dynamically adapting its structure through a cost-sensitive model that learns optimal node split/merge decisions. The learning-based adaptation ensures efficient memory utilization without compromising access performance or algorithm compatibility.
- **Comprehensive evaluation:** We systematically evaluate LearnGraph's architectural design using real-world and synthetic graph datasets, measuring operation throughput and end-to-end analytical processing performance. Our experimental

results demonstrate that LearnGraph's learning-based architecture achieves 3.4× higher throughput and reduces execution time by 3.5× on average compared to state-of-the-art systems Teseo [7] and Sortledton [8].

II. Background and Motivation

A. Graph Structure

A graph G is represented by $G(V, E)$, where V is the set of vertices and E is the set of edges. Real-world graphs exhibit sparse connectivity patterns where $|E| << |V|^2$ [22], [23], a property that influences our learning model design. Traditional adjacency lists organize these relationships by grouping edges by vertex, forming neighborhood structures. As illustrated in Figure 1, (a) shows an example graph while (b) demonstrates its adjacency list representation, where each vertex maintains references to its edge array sized by vertex degree. While graph systems may use different physical storage strategies, they preserve this logical adjacency view [7] to support algorithm execution. Understanding these structural properties is crucial for designing learning models that can efficiently predict and optimize both vertex and edge access patterns.

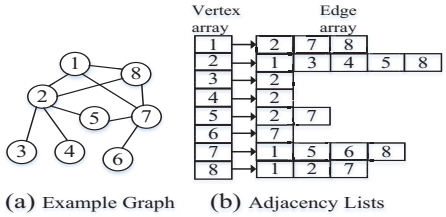


Fig. 1. Example graph and its adjacency lists.

B. Related Work

Research on dynamic graph processing has explored various fundamental data structures: CSR combined with sparse arrays [24], trees [25], [26], and hash tables [27]. However, these basic approaches achieve limited throughput, particularly when handling frequent updates. To address this limitation, hybrid solutions emerged - systems like Terrace [28] and GraphOne [2] adopted multiple data structures to balance graph update and analytics performance. More recently, LSGraph [6] focused on data locality optimization in real-world streaming scenarios but at the cost of increased memory overhead and system complexity. Graph processing algorithms (e.g., PageRank and BFS) in modern streaming applications require both efficient sequential and random access patterns to vertices and edges [29]. To meet these requirements, various optimization strategies have been proposed: hybrid edge-vertex structures [2], sparse arrays with tree-based indexing [7], [26], and CSR-like organizations [8]. While these approaches achieve reasonable lookup performance, they face scalability challenges with large, dynamic graphs due to their reliance on maintaining dense array structures. These limitations motivate exploring learning-based approaches for dynamic graph processing.

C. Motivation

Learning-based indexing represents an architectural innovation that achieves efficient data access through position prediction models. However, current architectures are fundamentally designed for independent data placement, lacking mechanisms to handle interconnected data structures. This architectural limitation prevents effective verification and navigation of relationships between data elements. Graph processing presents unique challenges for these architectures due to its inherently connected nature, where edges create complex relationships between vertices that must be efficiently accessed and maintained. While adapting existing architectures to treat edges as independent keys might seem viable, such an approach fails to capture the fundamental requirements of graph processing.

This architectural approach fails to efficiently support neighborhood access patterns, as treating edges independently fragments the critical vertex-neighbor relationships. Such a design cannot maintain coherent adjacency list views that graph algorithms fundamentally require. To quantify these architectural limitations, we evaluated a representative learning-based system, ALEX [30], against Teseo [7], a specialized graph processing architecture. Using five in-memory graphs (detailed in Section IV), our experiments in Figure 2 demonstrate that ALEX's architecture not only fails to provide benefits but actually degrades lookup throughput by up to 4× on large graphs compared to Teseo's graph-aware design. These results highlight the need for a learning-based architecture specifically designed to optimize neighborhood access patterns.

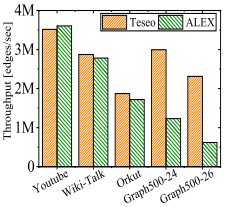


Fig. 2. Lookup throughput

Beyond their limitations with connected data, current learning-based architectures employ model designs unsuitable for dynamic graph processing. These architectures typically follow two approaches: (1) CDF-based designs that learn static data distributions [17], and (2) RMI-based hierarchical architectures where models form a fixed prediction hierarchy [30], [31]. Both architectural approaches have fundamental limitations: CDF-based designs assume fixed-length data structures, while RMI architectures restrict learning to single property mappings. Neither design can effectively adapt to the dynamic nature of graph data, where vertex and edge distributions continuously evolve. This architectural mismatch motivates the need for a new learning-enhanced design specifically optimized for dynamic graphs—one that maintains high-throughput operations on logical adjacency lists while adapting to evolving graph properties.

III. LearnGraph Framework

A. Overview

LearnGraph implements a learning-driven two-layer architecture that optimizes graph processing through intelligent position prediction and adaptive memory management. As illustrated in Figure 3, at the top, the Model Layer integrates

two specialized learned models: a Vertex Model that learns and predicts storage positions by adapting to vertex insertion patterns, and an Edge Model that learns position prediction based on dynamic neighborhood relationships. The Data Layer beneath implements a learning-guided hierarchical tree structure with three specialized node types: Internal Nodes that maintain model parameters and learned routing information, Vertex Nodes using learned position-aware fixed-size arrays, and Edge Nodes employing fixed-size arrays and learned sparse arrays with bitmap tracking for optimized edge operations. This hierarchical design enables the learning models to continuously adapt and optimize both data placement and access patterns throughout the graph structure.

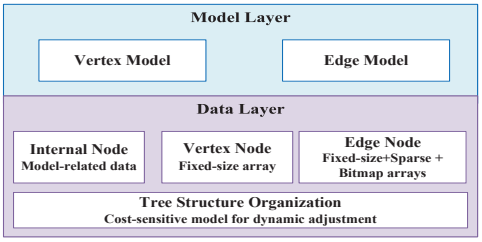
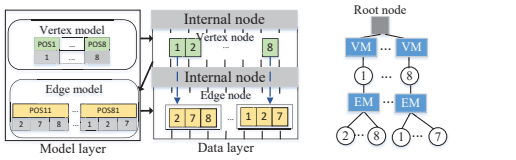


Fig. 3. LearnGraph Architecture.

Figure 4(a) illustrates LearnGraph's memory architecture using the example graph from Figure 1(a). The architecture uses learned models at two levels: a vertex model for position prediction and edge models for neighbor access optimization. Figure 4(b) shows the logical tree structure, where the root node employs the learned vertex model for indexing, while vertex nodes maintain edge models for efficient neighbor management. This architecture enables efficient graph access through continuous model adaptation. The following subsections detail how our two-layer design achieves high performance through learned vertex prediction and neighbor access optimization. To ensure usability, we provide modular interfaces that allow existing graph algorithms to leverage LearnGraph's optimizations without modification.



(a) In-memory structure of the example graph using LearnGraph (b) Logical view of LearnGraph

Fig. 4. The in-memory architecture and logic view of the example graph in Figure 1(a) using LearnGraph.

B. Model Layer

LearnGraph introduces a novel learning model architecture for dynamic graph processing, as traditional learned indexes like recursive model indexes (RMI) and CDF-based approaches prove insufficient for dynamic graphs (Section II). Our architecture builds upon RMI principles due to their inherent adaptability to growing and evolving data, but fundamentally reimagines their implementation for graph workloads. The architecture employs specialized linear regression models ($F(key) = a(key) + b$) for RMI-based insertions, with two key architectural innovations over existing RMI designs:

Graph-aware parameter optimization: Unlike existing RMI implementations like ALEX [30] that use arbitrary parameters leading to non-uniform memory distribution, LearnGraph's architecture carefully determines a and b based on graph properties. This design ensures uniform data distribution, reducing memory management overhead and improving access patterns.

Training-free model adaptation: While traditional RMI approaches rely heavily on training data selection and distribution, our architecture eliminates training set dependencies entirely. Instead, the vertex model learns from insertion order patterns, while the edge model adapts based on current neighborhood structures.

To support efficient graph operations, the architecture maintains two coordinated learning models: a vertex position predictor using insertion patterns and an edge locator leveraging neighborhood information. This dual-model design enables accurate position prediction for both vertices and edges while maintaining adaptivity to graph mutations.

1) Vertex model: For a vertex μ to be inserted, let $Num(\mu)$ represent its insertion order and D denote the capacity of a vertex or edge node. The vertex model determines the target vertex node for inserting μ using a function $F(\mu)$ as defined below.

$$F(\mu) = \lfloor \frac{Num(\mu)}{D} \rfloor + \lceil \frac{Num(\mu)}{D} \rceil \times D \tag{1}$$

2) Edge model: Edges fundamentally represent vertex neighborhoods. In undirected graphs, inserting an edge involves adding each vertex as a neighbor of the other, while in directed graphs, it requires adding just one vertex as a neighbor. Let $CurDeg(\mu)$ denote the current neighbor count of vertex μ. The edge model determines which edge node will store μ's neighbor ν, using a function $M[e(\mu, \nu), \nu]$ defined in Equation (2).

$$M[e(\mu, \nu), \nu] = \lfloor \frac{CurDeg(\mu)}{D} \rfloor + \lceil \frac{CurDeg(\mu)}{D} \rceil \times D \tag{2}$$

3) The specific insertion position: Vertex insertion follows a two-step process: first determining the target vertex node, then identifying the specific array position within that node. For a vertex μ, if $\lfloor \frac{Num(\mu)}{D} \rfloor = T$, then $Num(\mu)$ falls within $[TD, (T+1)D)$. This means μ will be stored in the Tth vertex node, which has capacity D. After determining the vertex node, the specific array position is calculated using $Mod(\frac{Num(\mu)}{D})$, which gives the remainder when $Num(\mu)$ is divided by D. Edge insertion follows a similar process for determining the target edge node. Once the edge node is identified, exponential search locates the specific insertion position as detailed in Section III-D2.

C. Data Layer

LearnGraph organizes in-memory graph data using a tree structure comprising internal nodes, vertex nodes, and edge

nodes. As shown in Figure 5, each vertex or edge node contains a model value ($'MV'$) used to determine element positions within that node. The data structure employs vertex arrays, sparse arrays, and bitmap arrays to store both vertex-related and edge-related data. To represent vertex neighborhoods, vertices in vertex nodes maintain neighbor reference arrays that point to their corresponding edge nodes.

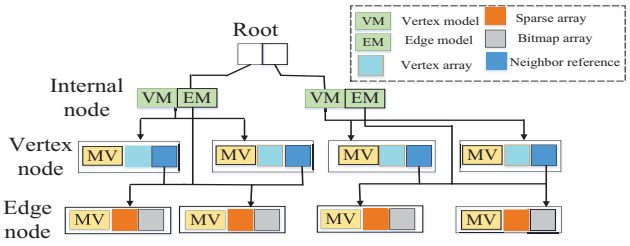


Fig. 5. The structure of the data layer in LearnGraph.

1) Internal node: An internal node maintains references to vertex nodes, edge nodes, and other internal nodes as its children. The model value range of a parent internal node encompasses those of all its child data nodes. Beginning from the second level of the tree (with total height K), we constrain the maximum number of internal nodes at the ith level to t^{i-1}, where t is an integer ≥ 2 and $2 \leq i \leq K-2$. Internal nodes at the same level maintain an equal number of references to next-level nodes, with this reference count being dynamically configurable in LearnGraph. These internal nodes serve to implement recursive model indexes, providing flexible management of vertex and edge nodes.

2) Vertex node and edge node: Vertex and edge nodes store model values for position prediction. Each vertex node maintains a fixed-size array of capacity D to store vertices and a separate array for neighbor references. Since many nodes have few neighbors in a graph, we use a fixed-size array of capacity 10 to store the earliest arriving neighbors in sorted order. When neighbor counts exceed this fixed array size, edge nodes switch to sparse arrays that contain deliberate gaps to accommodate insertions. Following established practices [7], [30], we maintain 75% occupancy in sparse arrays, reserving the remaining space as gaps to optimize search operations. Each gap is populated with its closest right neighbor, enabling efficient exponential search with $O(\log D)$ time complexity for vertex lookups, where D is the sparse array size [30]. Edge nodes also maintain a bitmap array matching the sparse array's size to mark occupied positions, allowing searches to skip gaps efficiently.

D. Basic algorithms

The algorithms for vertex operations (lookups, insertions, and deletions) in LearnGraph are straightforward since vertex positions can be directly computed from their arrival order. Edge operations, particularly in learned sparse arrays with bitmap tracking, require more detailed examination. While operations on neighbors stored in fixed-size arrays are efficiently handled using quicksort, we focus on the algorithms for edge lookup, insertion, and deletion in learned sparse arrays.

1) Lookup: Edge lookup begins by verifying that both vertices of the edge exist in vertex nodes. If confirmed, we locate the root node of vertex neighbors using one vertex's starting pointer. Since LearnGraph implements model-based insertions, we first check the edge's model value. A non-existent model value indicates the edge hasn't been inserted, resulting in a null record return. Otherwise, edge models are recursively applied from the root node, predicting positions at each level until reaching the edge node. If the edge is found at the predicted position, the lookup returns that record. If not, exponential search is initiated from the predicted position to locate the edge's actual position.

2) Insert: The edge insertion process in LearnGraph follows a structured approach utilizing both sparse arrays and bitmap arrays. As shown in Figure 6, starting from the initial state, where the sparse array maintains ordered destination vertex IDs of edges (e.g., edges (0, 2), (0, 5), (0, 8), (0, 9), where Vertex 0 is the source vertex) with intentional gaps and a corresponding bitmap array (1, 1, 0, 1, 0, 1) tracking occupied positions, the LearnGraph system employs a four-step process for edge insertion: First, the system verifies both vertices exist in vertex nodes using vertex models, and inserts any missing vertices into vertex nodes at positions determined according to Section III-B3. Second, the edge model is applied to compute the model value and predict the target position - insertion proceeds only if no model value exists. Third, for each neighbor vertex insertion (one for directed graphs, two for undirected graphs), the system traverses internal nodes following edge models from the root node to reach the target edge node.

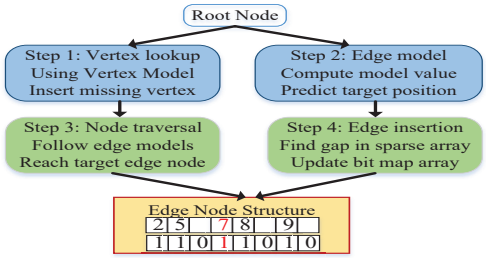


Fig. 6. Insertion process using LearnGraph.

Finally, when inserting an edge, e.g., (0, 7), representing a connection from vertex 0 to vertex 7, if inserting at the predicted position maintains the edge node's sparse array ordering, the insertion succeeds immediately; otherwise, exponential search locates the correct insertion position. When inserting at the final position between existing destination vertices (between 5 and 8), the system identifies an available gap using the bitmap array (0 indicates available gap) for direct placement, or shifts elements to the nearest rightward gap if the position is occupied. Once the position is determined, the destination vertex ID 7 is inserted into the sparse array, and the corresponding bitmap position is updated from 0 to 1 to reflect the occupied status. If a model value falls outside internal

node ranges, the ranges and references must be adjusted, though this is typically prevented by setting sufficiently large initial ranges. This process maintains both the sorted order of edge destinations and the target 75% utilization rate, while preserving gaps for future insertions, ensuring efficient edge search and insertion operations through the combination of model-based prediction and bitmap-assisted position tracking.

3) Delete and update: Edge deletion employs the lookup algorithm to locate edge positions within the edge node. Once found, the position is converted to a gap by clearing it and setting the corresponding bitmap position to '0', followed by removing the edge's model values. Edge updates are straightforward, simply writing new edge values into positions identified by the lookup algorithm.

E. Cost-sensitive model

First, we analyze the time complexity and overhead of constructing our learning-based architecture before considering the cost-sensitive model. This analysis applies to any input graph with $|V|$ nodes and $|E|$ edges. The time complexity is primarily determined by data access operations. Our tree-based data structure requires $O(\log K)$ time for internal lookups and $O(\log D)$ time for vertex searches. In terms of space complexity, internal nodes consume $O((|V|+2|E|)/H)$ space, while vertex and edge nodes require $O((|V|+2|E|)/(0.75D))$ space, plus negligible overhead from a few small fixed-size arrays that store the first-arrival neighboring nodes. Here, K denotes the tree height, D represents the fixed-size array capacity, and H (64MB) specifies the maximum size of each internal node's subtree.

Since vertex operations involve simple computations with minimal overhead, we focus our cost-sensitive model on edge operations. The model relies on historical statistics of edge lookups and insertions. For any vertex μ, LearnGraph monitors two key metrics: $numTra_\mu$, which counts nodes traversed from root to edge nodes, and $numSearch_\mu$, which counts exponential searches in edge nodes. The ratio between these metrics guides our dynamic tree structure adjustments, with $numTra_\mu$ indicating memory usage for storing μ's neighbors and tree height, while $numSearch_\mu$ reflects the computational costs of neighbor operations and tree width.

Using a threshold $\epsilon = 0.2$, we implement a dynamic adjustment strategy. When $\frac{numTra_\mu}{numSearch_\mu} \leq \epsilon$, we optimize for computation time by increasing the RMI-based edge models' height from K to $K+1$ and evenly distributing the Kth level node references across new $(K+1)^{th}$ level nodes. Conversely, when the ratio exceeds ϵ, we consolidate the structure by removing the last level of internal nodes and connecting second-last level nodes directly to edge nodes.

IV. Evaluation

We perform experiments on a machine equipped with Intel(R) Xeon(R) Silver 4110 CPU @2.10GHz and 256GB memory. Our evaluation examines both operation throughputs (lookups, inserts, and deletes) and processing times for five common graph algorithms: breadth-first search (BFS), PageRank (PR), local triangle counting (LCC), weakly connected components (WCC), and single-source shortest path (SSSP). We compare **LearnGraph** against two state-of-the-art graph structures: **Teseo** [7] and **Sortledton** [8], using three real-world datasets and two synthetic graphs as shown in Table I. While all three systems support multithreading, we evaluate LearnGraph (with maximum array size 100 for vertex and edge nodes) and Teseo using single-thread execution to simplify comparisons and minimize hardware configuration effects. Sortledton, being specifically designed for multithreaded environments, is evaluated with multiple threads for optimal performance.

TABLE I
SUMMARY OF GRAPH DATASETS, WHERE $|V|$ DENOTES THE NUMBER OF THE VERTICES IN THE VERTEX SET V, $|E|$ THE TOTAL NUMBER OF THE EDGES IN THE EDGE SET E.

Graph	$\|V\|$	$\|E\|$
Youtube	1,134,890	2,987,624
Wiki-Talk	2,394,385	5,021,410
Orkut	3,072,441	117,184,899
Graph500-24	8,870,942	26,037,952
Graph500-26	32,804,978	1,051,922,853

A. Throughput

Lookup throughput. We evaluate lookup performance by measuring the total time required to search all edges in random order across each input graph. As shown in Figure 7, LearnGraph achieves an average $2.1X$ higher throughput compared to Teseo and Sortledton across the five datasets. This performance advantage stems from LearnGraph's architectural approach: while Teseo and Sortledton rely on a two-step process of vertex table lookup followed by binary search through adjacency list-like structures, LearnGraph employs learning models to directly predict vertex and edge positions, using exponential search only for final verification.

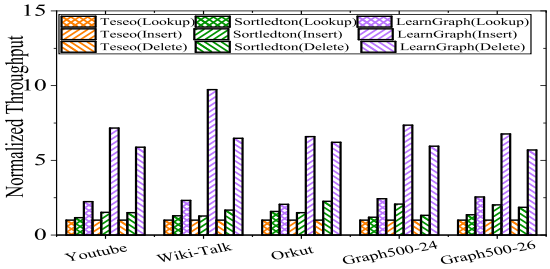
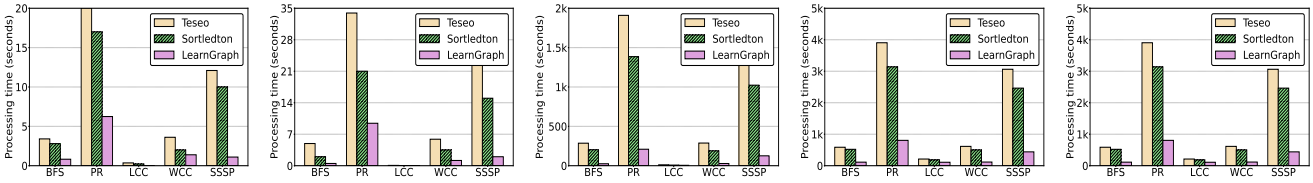


Fig. 7. The throughput of LearnGraph compared to competitors.

Insert throughput. We evaluate insertion performance by measuring random-order edge insertions into the input graph. Figure 7 demonstrates that LearnGraph achieves at least $7X$ higher insertion throughput than Teseo and Sortledton across the five datasets. This significant performance gap arises from fundamental structural differences: Teseo utilizes a modified B+ tree (fat tree) with leaf nodes storing the vertex table alongside a hash table, while Sortledton employs dual-level



(a) Youtube (b) Wiki-Talk (c) Orkut (d) Graph500-24 (e) Graph500-26

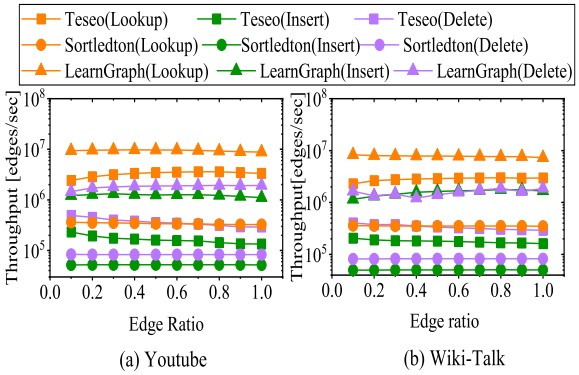
Fig. 8. Processing time of graph algorithms with different analytic systems.

adjacency list-like and CSR-like structures. Both competitors require extensive structure traversal to locate proper insertion positions, whereas LearnGraph leverages vertex and edge models to directly predict insertion positions, resulting in substantially shorter traversal paths.

Delete throughput. Figure 7 demonstrates that LearnGraph achieves at least $5X$ higher deletion throughput compared to existing graph analytic systems. This superior performance is attributed to LearnGraph's learning model approach, which quickly predicts vertex and edge positions, in contrast to Teseo and Sortledton's requirement to traverse vertex tables and associated structures for element location.

B. Processing time of Graph analysis

Figure 8 demonstrates that LearnGraph processes the five algorithms on average $3.5X$ faster and reduces processing time by 1.7× to 11× over Teseo and Sortledton across all datasets. Most notably, LearnGraph executes BFS up to 10X faster than its competitors, as BFS and LCC operations require extensive vertex and neighbor traversal, which LearnGraph accomplishes in $O(\log D)$ time using learning models, while Sortledton requires $O(D)$ time with vertex table scans. For PR and WCC, which also depend heavily on vertex and neighbor access, LearnGraph achieves an average 2.5× speedup over Teseo and Sortledton, whose performance is limited by hashmap lookup and skip list traversal costs. SSSP execution is at least $5X$ faster in LearnGraph, benefiting from its superior lookup throughput. These performance improvements across various graph algorithms demonstrate LearnGraph's effectiveness in accelerating operation throughput.



(a) Youtube (b) Wiki-Talk

Fig. 9. LearnGraph keeps high and steady throughput with a growing ratio of edges.

C. Evaluation on Cost-sensitive Model

We evaluate our cost-sensitive model's effectiveness through scalability analysis, measuring edge-process throughput as a function of edge ratio in dynamic graph processing. Edge-process throughput represents operations (lookups, insertions, and deletions) per second on sequentially processed edges, while edge ratio indicates the proportion of processed edges to total graph edges. Figure 9 demonstrates that LearnGraph achieves 3.4× higher throughput on average compared to Teseo and Sortledton across increasing edge ratios on Youtube and Wiki-Talk datasets. This performance advantage stems from our cost-sensitive model's ability to adapt to varying workload characteristics. Particularly in Youtube and Wiki-Talk graphs, where neighbor counts [32], [33] exceed the capacity of an edge node rooted at an internal node, the cost-sensitive model stabilizes throughput fluctuations during edge insertions and deletions by dynamically adjusting data structure configurations and optimizing memory allocation patterns. Teseo achieves moderately higher throughput than Sortledton, primarily because Sortledton's versioning-based concurrency mechanisms introduce unnecessary overhead during sequential edge processing. Overall, Figure 9 validates that LearnGraph's cost-sensitive model enables consistent and superior operation throughput when handling dynamic graphs through balanced operation costs across different graph regions.

V. Conclusion

This paper introduces LearnGraph, a learning-based architecture for dynamic graph processing. Our system integrates learned models into the system architecture through a hierarchical approach: a vertex prediction model for optimal data placement and an edge model for efficient neighbor access, both built into an adaptive tree structure. The architecture employs a learning-based cost model that continuously optimizes tree organization based on operation patterns. Experimental evaluation across five diverse graph datasets demonstrates that LearnGraph's architecture outperforms state-of-the-art systems Teseo and Sortledton, achieving throughput improvements of 2.1× for lookups, 7× for insertions, and 5× for deletions. When evaluating five common graph algorithms, LearnGraph's learning-enhanced architecture delivers performance gains of 3.5× over existing competitors.

VI. Acknowledgement

We would like to thank the anonymous reviewers. This work is supported by the NSFC (Natural Science Foundation of China) under Grant No.62302043.

REFERENCES

[1] Zhuo *et al.*, "Graphq: Scalable pim-based graph processing," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

[2] Kumar *et al.*, "Graphone: A data store for real-time analytics on evolving graphs," in *ACM Transactions on Storage*, 2020.

[3] K. Ammar, F. McSherry, S. Salihoglu, and M. Joglekar, "Distributed evaluation of subgraph queries using worst-case optimal low-memory dataflows," *Proceedings of the VLDB Endowment*, vol. 11, no. 6, pp. 691–704, 2018.

[4] M. Besta, M. Fischer, V. Kalavri, M. Kapralov, and T. Hoefler, "Practice of streaming processing of dynamic graphs: Concepts, models, and systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 6, pp. 1860–1876, 2021.

[5] Asgari *et al.*, "Fafnir: Accelerating sparse gathering by using efficient near-memory intelligent reduction," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2021.

[6] H. Qi, Y. Wu, L. He, Y. Zhang, K. Luo, M. Cai, H. Jin, Z. Zhang, and J. Zhao, "Lsgraph: a locality-centric high-performance streaming graph engine," in *Proceedings of the Nineteenth European Conference on Computer Systems*, 2024, pp. 33–49.

[7] Leo *et al.*, "Teseo and the analysis of structural dynamic graphs," in *Proceedings of the VLDB Endowment*, 2021.

[8] Fuchs *et al.*, "Sortledton: a universal, transactional graph data structure," in *Proceedings of the VLDB Endowment*, 2022.

[9] Sahu *et al.*, "The ubiquity of large graphs and surprising challenges of graph processing," in *Proceedings of the VLDB Endowment*, 2017.

[10] M. N. Bojnordi and F. Nasrullah, "Retagger: An efficient controller for dram cache architectures," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[11] X. Chen, T. Huang, S. Xu, T. Bourgeat, C. Chung, and A. Arvind, "Flexminer: A pattern-aware accelerator for graph pattern mining," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 581–594.

[12] P. Li, Y. Hua, P. Zuo, Z. Chen, and J. Sheng, "{ROLEX}: A scalable {RDMA-oriented} learned {Key-Value} store for disaggregated memory systems," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023, pp. 99–114.

[13] H. Qi, Y. Zhang, L. He, K. Luo, J. Huang, H. Lu, J. Zhao, and H. Jin, "Psminer: A pattern-aware accelerator for high-performance streaming graph pattern mining," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[14] Nathan *et al.*, "Learning multi-dimensional indexes," in *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, 2020.

[15] Wu *et al.*, "Updatable learned index with precise positions," in *arXiv preprint arXiv:2104.05520*, 2021.

[16] Kraska *et al.*, "The case for learned index structures," in *Proceedings of the 2018 international conference on management of data*, 2018.

[17] Tang *et al.*, "Xindex: a scalable learned index for multicore data storage," in *Proceedings of the 25th ACM SIGPLAN symposium on principles and practice of parallel programming*, 2020.

[18] Wei *et al.*, "Fast {RDMA-based} ordered {Key-Value} store using remote learned cache," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, 2020.

[19] Ma *et al.*, "Film: A fully learned index for larger-than-memory databases," in *Proceedings of the VLDB Endowment*, 2022.

[20] Yu *et al.*, "Treeline: an update-in-place key-value store for modern storage," in *Proceedings of the VLDB Endowment*, 2022.

[21] Yang *et al.*, "{GL-Cache}: Group-level learning for efficient and high-performance caching," in *21st USENIX Conference on File and Storage Technologies (FAST 23)*, 2023.

[22] Zhang *et al.*, "Depgraph: A dependency-driven accelerator for efficient iterative graph processing," in *2021 IEEE International Symposium on High-Performance Computer Architecture*, 2021.

[23] P. Kumar and H. H. Huang, "G-store: high-performance graph store for trillion-edge processing," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 830–841.

[24] M. A. Bender and H. Hu, "An adaptive packed-memory array," *ACM Transactions on Database Systems (TODS)*, vol. 32, no. 4, pp. 26–es, 2007.

[25] M. Besta, M. Fischer, V. Kalavri, M. Kapralov, and T. Hoefler, "Practice of streaming processing of dynamic graphs: Concepts, models, and systems," *arXiv preprint arXiv:1912.12740*, 2019.

[26] L. Dhulipala, G. E. Blelloch, and J. Shun, "Low-latency graph streaming using compressed purely-functional trees," in *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, 2019, pp. 918–934.

[27] M. A. Awad, S. Ashkiani, S. D. Porumbescu, and J. D. Owens, "Dynamic graphs on the gpu," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2020, pp. 739–748.

[28] P. Pandey, B. Wheatman, H. Xu, and A. Buluc, "Terrace: A hierarchical graph container for skewed dynamic graphs," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 1372–1385.

[29] Zhao *et al.*, "Tdgraph: a topology-driven accelerator for high-performance streaming graph processing," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022.

[30] Ding *et al.*, "Alex: an updatable adaptive learned index," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

[31] Li *et al.*, "Finedex: a fine-grained learned index scheme for scalable and concurrent memory systems," in *Proceedings of the VLDB Endowment*, 2021.

[32] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proceedings of the ACM SIGKDD workshop on mining data semantics*, 2012, pp. 1–8.

[33] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th international conference on World wide web*, 2010, pp. 641–650.

DAWN: Accelerating Point Cloud Object Detection via Object-Aware Partitioning and 3D Similarity-Based Filtering

Dongdong Tang[1], Yu Mao[1]*,Weilan Wang[1], Nan Guan[1], Tei-Wei Kuo[2], and Chun Jason Xue[3]
[1]City University of Hong Kong, [2]National Taiwan University, [3]MBZUAI

***Abstract*—As a fundamental perception task, 3D point cloud detection has become essential for applications in autonomous driving and robotics. However, point cloud detection faces significant challenges of high computational cost due to complex point processing operations. To address this issue, we propose DAWN, an acceleration framework for point cloud object detection that identifies partial similarities between adjacent frames and reduces computational cost by filtering redundant points. DAWN uses object-aware partitioning that defines boundaries based on previous detection results for localized similarity analysis. Additionally, it applies axis-sorted point selection to refine partitioning for point clouds with non-uniform distribution. An efficient 3D similarity algorithm then filters redundant points to reduce computational load. DAWN enables flexible latency-accuracy trade-offs by tuning point filtering ratios. Experimental results show that DAWN achieves a $1.59\times$ average speedup and up to $1.70\times$ on state-of-the-art detection networks by filtering more than 50% of points on average, with negligible impact on accuracy.**

I. Introduction

With the rapid advancement of 3D sensing technology, point cloud data has become increasingly popular for its detailed 3D geometric information [1], [2]. A point cloud is a set of 3D points generated by sensing devices such as LiDAR (Light Detection and Ranging) sensors [3], [4]. Point cloud data processing requires complex computations, posing significant challenges for real-time autonomous driving and robotics implemented on embedded devices. To address this, we propose DAWN, an acceleration framework that reduces computation by filtering redundant points, decreasing processing latency while preserving accuracy.

Point cloud detection can be broadly categorized into voxel-based and point-based methods. Voxel-based detection methods convert the point cloud to 3D grids for efficient 3D CNN processing [5], [6] but suffer from loss of 3D spatial information during the voxelization process. Point-based detection methods process raw points directly, achieving higher accuracy by preserving fine-grained patterns and local structures. However, point-based detection methods suffer from significant processing latency due to substantial computational costs. Point-voxel based detection networks process both voxel-based and point-based features for detection [10], [26]. Some of these methods achieve high accuracy but face the same latency challenges as point-based methods due to intensive point processing.

*Corresponding author

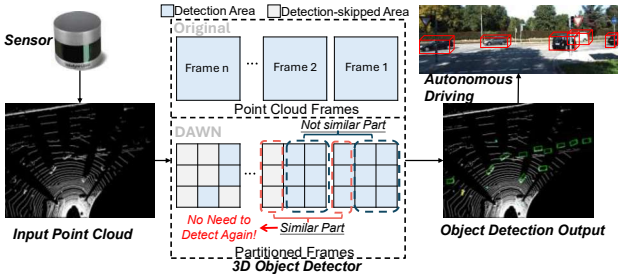


Fig. 1: Original point cloud detection processes entire frames. DAWN accelerates detection by identifying partial similarities and filtering redundant points.

This paper proposes DAWN, an acceleration framework that reduces the inference latency of point cloud detection methods while preserving accuracy. DAWN is based on the observation that points in similar areas across adjacent frames typically produce similar detection outcomes. As shown in Figure 1, DAWN partitions frames and identifies areas that are similar to their counterparts in the previous frame through localized similarity analysis. DAWN skips detection for these areas by reusing previous results and only processes those with significant changes. Unlike traditional methods that perform detection on entire frames, DAWN's selective processing strategy reduces points for computation. However, implementing this strategy presents three key challenges. First, DAWN requires frame partitioning for localized similarity analysis, but defining partitioning boundaries is non-trivial. Naive solutions without considering object positions can fragment objects across partitions, leading to detection errors. Second, while DAWN needs fine-grained partitions for partial similarity analysis, the non-uniform point cloud distribution poses challenges. The partitioning result may be skewed, where some partitions are empty while others become overcrowded with points. Third, DAWN should precisely identify truly redundant points that can be safely skipped. Otherwise, the speedup would come at the cost of significant accuracy loss.

To address these challenges, we propose three key design elements. First, we propose a dynamic object-aware partitioning that defines boundaries based on previous detection results to avoid object fragmentation. Second, we present the axis-sorted point selection to refine the partitioning, enabling localized frame comparisons for better partial similarity analysis. Third,

we introduce a GPU-efficient 3D similarity algorithm that accurately filters redundant points through parallel computation of point distances. Based on the experiment results, DAWN achieves a $1.59\times$ average speedup and up to $1.70\times$ on state-of-the-art detection networks, with minimal accuracy loss. Additionally, DAWN enables flexible latency-accuracy trade-offs to meet different system constraints and application requirements. Our work makes the following contributions:

- We propose a dynamic object-aware partitioning method that maintains object integrity by defining boundaries based on previous detection results.
- We introduce axis-sorted point selection to further refine the partitioning for more effective partial similarity analysis between adjacent frames.
- We present an efficient 3D similarity algorithm to reduce detection workload by accurately filtering the redundant points.
- We implement DAWN, a framework providing flexible latency-accuracy trade-offs, on three state-of-the-art detection networks. Experiment results show that DAWN accelerates these networks by $1.59\times$ on average and up to $1.70\times$ by filtering more than 50% of points on average, with negligible impact on accuracy.

II. Background and Related Work

A. Point Cloud Object Detection

A point cloud is a set of sparse, unstructured 3D data points defined by their (X, Y, Z) coordinates. Point cloud data is widely used in various fields, including autonomous driving, robotics, and remote sensing [1]. Among point cloud applications, object detection is a crucial perception task. It identifies and locates objects for environmental perception. Point cloud detection methods can be categorized into point-based and voxel-based detection networks. Voxel-based detection networks transform point clouds into regular 3D grids for efficient 3D Convolutional Neural Networks (CNNs) processing [5]–[7]. However, this voxelization process can lose fine-grained spatial information as points within each voxel are summarized into single feature representations. Point-based detection networks directly process raw points and preserve more spatial information, offering higher detection accuracy at the cost of longer inference time [8], [9]. Point-voxel based detection networks process both voxel-based and point-based features for detection [10], [11]. While some methods achieve low processing latency, others targeting high accuracy face computational challenges due to intensive point processing.

B. Acceleration of Point Cloud Detection

Numerous acceleration methods have been developed to enhance the performance of point cloud detection [12], [13], [15], [27]–[30], [32]–[39]. Mesorasi [16] achieves faster and more energy-efficient detection on GPU by applying delayed aggregation to the detection network. Point-X [17] proposes a spatial locality-aware accelerator, in which a speculative breadth-first search (BFS) graph traversal method is applied to extract the locality of input. PointAcc [18] presents a versatile design supporting efficient nonzero mapping operations, effectively addressing input sparsity and improving processing speed. Mars [14] is a memory access reduced and scalable accelerator for point-based and voxel-based 3D point cloud neural networks.

III. Motivation

Our preliminary study evaluates the performance limitations of point-based detection. Using the KITTI dataset [19], we compare different detection methods on a desktop platform equipped with an Intel(R) Core(TM) i7-9700K CPU@3.6GHz, 64GB DDR4 RAM and NVIDIA GTX 2080Ti. As shown in Figure 2, point-based and point-voxel based detection networks process fewer than 10 frames per second. The processing speed is insufficient for real-time operation because most LiDAR sensors used in autonomous driving generate point cloud data at 10 Hz [19]–[21]. Voxel-based detection networks achieve faster processing but suffer from lower accuracy due to voxelization's spatial information loss.

Figure 3 identifies point processing as the primary computational bottleneck in both point-based and point-voxel based detection networks. Point processing steps are operations on raw points that include feature extraction through set abstraction and subsequent processing. Point-based and point-voxel based detection networks, sharing the point processing steps, suffer from the same latency challenges. To address this challenge, we propose DAWN, an acceleration framework that speeds up the detection while maintaining accuracy by filtering redundant points based on spatio-temporal frame similarity.

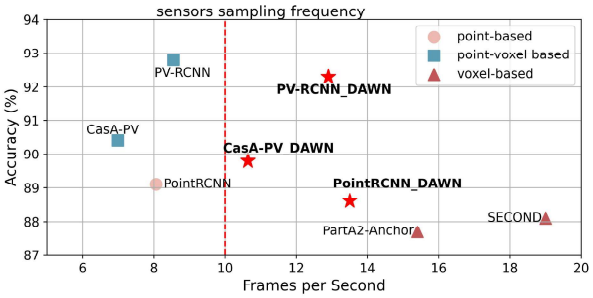


Fig. 2: Detection speed of different methods. The point-based and point-voxel based detection networks process fewer than 10 frames per second. The voxel-based detection networks are faster but have a larger accuracy loss.

IV. Methodology

A. System Overview

DAWN focuses on continuous point cloud frames. As Figure 5 shows, objects can move or remain stationary between frames. Consequently, regions that remain similar across adjacent frames typically produce similar detection results. Based on this observation, DAWN leverages these partial similarities to reduce computation by skipping redundant detection. Figure 4 shows the three key components of DAWN. First,

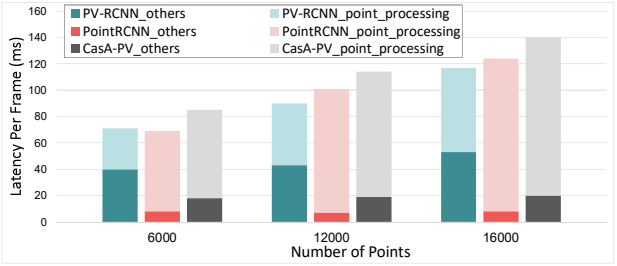


Fig. 3: Latency breakdown. The breakdown reveals that point processing is the main computational bottleneck.

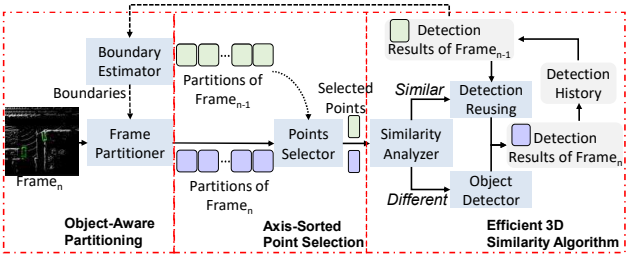


Fig. 4: System Overview: DAWN initially partitions a frame using object-aware partitioning (Section IV-C). It selects points using axis-sorted point selection (Section IV-D) and compares them to the corresponding part in the previous frame. The detector processes points sufficiently different from their previous frame counterparts, filtering redundancies through similarity analysis (Section IV-E).

object-aware partitioning divides each frame using boundaries estimated from previous detection results to avoid object fragmentation. Second, axis-sorted point selection selects points at fine-grained granularity for better similarity analysis. The granularity level defines how many points are processed for the similarity comparison to decide if detection can be skipped. Point selection enables localized comparison between the current (Frame$_n$) and the previous frame (Frame$_{n-1}$) using an efficient 3D similarity algorithm. The detector processes points that differ significantly from their counterparts in previous frames while reusing detection results for sufficiently similar regions. The detection results of the current frame are then stored in the detection history for subsequent frame processing.

B. Design Challenges

DAWN is designed to speed up point cloud detection while maintaining accuracy. However, the design faces several challenges.

Challenge 1: Preserving Object Integrity Maintaining object integrity in frame partitioning is crucial for accurate detection. Naive frame partitioning using fixed boundaries introduces problems. When partition boundaries intersect with objects without considering their positions, the objects' constituent points become fragmented across different partitions. The fragmented point sets cannot provide complete spatial information for accurate detection, resulting in detection errors.

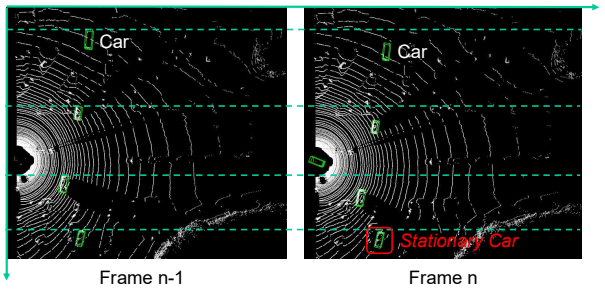


Fig. 5: Partial similarity between adjacent frames. Objects in a frame may have varying degrees of motion.

Challenge 2: Achieving Fine-grained Partitioning Granularity Sufficiently fine-grained partitioning granularity is crucial for utilizing spatial similarity effectively. The initial partitioning may not achieve optimal granularity due to non-uniform point cloud distribution. The partitioning methods may result in skewed point distributions, with some partitions containing no points while others are excessively dense. As a result, the partial similarity cannot be well explored.

Challenge 3: Filtering by Similarity The challenge of similarity-based filtering lies in accurately identifying partial similarity. The method should correctly identify points that are sufficiently different from their counterparts in the previous frame. Otherwise, the speedup would come at the cost of significant accuracy degradation. Additionally, the similarity calculation should be fast to avoid introducing significant overhead.

C. Object-Aware Partitioning

Objects in point cloud frames show varying motion: some move significantly, others remain stationary. Figure 5 illustrates these motion variations between adjacent frames. Stationary regions across frames are highly similar and produce similar detection results, while regions with significant movement require new detection. To identify these partial similarities between frames, developing a partitioning method for localized comparison is essential.

To minimize detection errors, it's essential to maintain object integrity when partitioning frames. A naive approach involves partitioning the frames with fixed boundaries. However, this can lead to the problem of fragmenting the objects across the boundaries. Object detection algorithms struggle to recognize objects from these fragmented sets due to the lack of complete spatial information needed for accurate detection.

To tackle this challenge, we propose object-aware partitioning to split the frame while preserving object integrity. This partitioning defines boundaries based on partial similarities between adjacent frames. This approach is based on the principle that the objects maintain spatial continuity across adjacent frames. The position of objects changes gradually between adjacent frames, rather than exhibiting unexpected movements to distant locations.

Based on this observation, we utilize the predictions from the previous frame, denoted as $Pred = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ to determine the partitioning of the current frame. The predictions of previous frames are denoted as $[x, y, z, w, l, h]$, where $[x, y, z]$ is the position and $[w, l, h]$ is the dimension of objects. We then define a series of partitioning planes $a_i x + b_i y + c_i z + d_i = 0$ that are positioned between these objects without fragmenting the objects. With the partitioning planes established, we determine how to partition the frame and assign each point to its appropriate partition. Each point (x_j, y_j, z_j) in the frame is compared with the partitioning planes to determine its relative position. This comparison is performed by substituting the coordinates of the point (x_j, y_j, z_j) into the equation of each plane. Each comparison will return a result r_i, which equals 1 if the point lies on or above the plane $(a_i x_j + b_i y_j + c_i z_j + d_i \geq 0)$, and 0 if the point lies below the plane $(a_i x_j + b_i y_j + c_i z_j + d_i < 0)$.

Results from comparing all partitioning planes determine the appropriate partition for points. The point with comparison results $r_1, r_2, r_3, r_4, \ldots, r_n$ will be assigned to partition k, where $k = (r_1 r_2 r_3 r_4 \ldots r_n)_2$. Here, $(\ldots)_2$ denotes the binary representation of the partition index. Figure 6 compares fixed boundary partitioning and dynamic object-aware partitioning. Fixed boundaries fragment objects across different partitions, resulting in detection errors. Conversely, object-aware partitioning maintains object integrity within partitions, shown by uniform color in bounding boxes. By maintaining object integrity, object-aware partitioning better preserves detection accuracy.

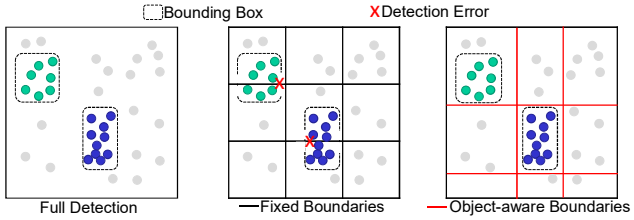


Fig. 6: Object-aware partitioning. The partitioning boundaries are defined based on detection results from previous frames.

D. Axis-Sorted Point Selection

The object-aware partitioning method effectively splits the frame while maintaining object integrity. However, the partitioning may not achieve fine-grained granularity due to non-uniform point cloud distribution. If partitions lack ideal granularity, exploring partial similarity becomes challenging. Directly applying similarity calculations on these partitions does not work well.

To address this challenge, we propose axis-sorted point selection. The point selection step is added before the similarity calculation to select the points at a fine-grained granularity level. Since DAWN leverages partial similarities between adjacent frames within specific regions, our point selection method should ensure that the selected points form cohesive regions rather than scattering across the space. To maintain the spatial continuity of selected points, we implement a two-step sorting process before selection.

The detailed steps of the point selection are illustrated in Algorithm 1. This step selects points from the current frame and the previous frame at an optimal granularity level. First, we sort point sets from both current and previous frames along the x-axis and divide them into groups. To avoid fragmenting objects across different groups, we determine ranges along the x-axis that are likely to contain objects based on previous detection results. The first n_{g_1} points in both point sets are assigned with $id.x = 0$, and the next n_{g_2} points in both point sets are assigned with $id.x = 1$. This assignment continues with each subsequent group of n_{g_i} points receiving an incrementally higher $id.x$ value. The group sizes n_{g_i} are adjusted to avoid objects being fragmented across different groups. The same sorting and assignment steps are then repeated in the y-direction. After the sorting and assignment, each point has a pair of indices $(id.x, id.y)$. The final selection index is computed as $id = id.x + id.y * u_y$, where u_y is the number of unique elements in $id.y$. The points with the same id are selected together, ensuring that the selected points form cohesive regions. Figure 7 illustrates the point selection process with a simplified example, where $id.x$ and $id.y$ range from 0 to 2. We skip selection along the z-axis since objects are primarily distributed on the road plane, having much smaller differences in height than in their x and y positions.

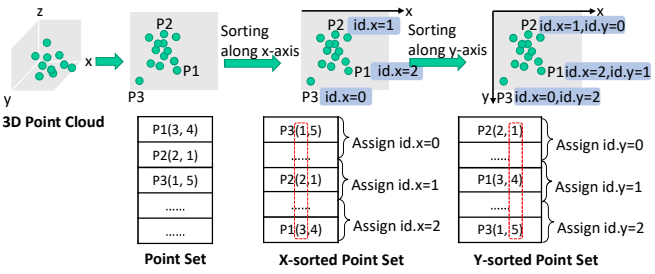


Fig. 7: Axis-sorted point selection selects points at fine-grained granularity to better explore the partial similarity.

E. Efficient 3D Similarity Algorithm

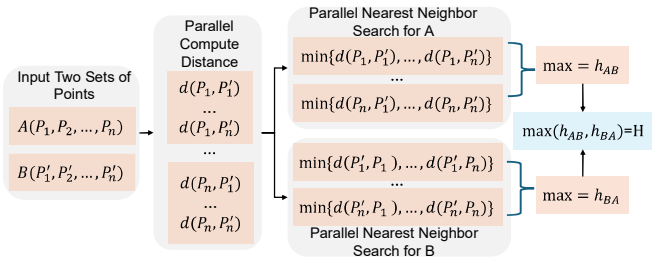


Fig. 8: The 3D similarity calculation provides efficient similarity analysis to identify points to be filtered out.

To accelerate detection while maintaining accuracy, DAWN identifies partial similarities between adjacent frames and

Algorithm 1 Axis-sorted Point Selection.

Input: N Points P_1, P_2 from current and previous frames.
Output: k division of points at optimal granularity levels.

Sort points P_1, P_2 in x direction;
Object ranges $\mathbb{X}_i$=(x^i_{min}, x^i_{max}) from previous frame;
$j = 0$;
while $n_x \leq N$ **do**
  Select $P_2[n_x : n_x + n_{g_i}]$ with range $\mathbb{X}_s$;
  if $\mathbb{X}_i \cap \mathbb{X}_s \neq \emptyset$ **then**
    Adjust n_{g_i} until $\mathbb{X}_i \in \mathbb{X}_s$;
  end if
  points in $P_1[n_x : n_x + n_{g_i}]$ have $id.x = j$;
  points in $P_2[n_x : n_x + n_{g_i}]$ have $id.x = j$;
  $n_x = n_x + n_{g_i}$,$j = j + 1$;
end while
Repeat the division in y direction and assign $id.y$;
Points are indexed with division $id = id.x + id.y * u_y$, u_y is count of unique $id.y$ values;

accurately filters redundant points. We propose an efficient 3D similarity algorithm using the Hausdorff distance to identify points requiring detection. The Hausdorff distance is widely used to measure the degree of similarity between two sets of points. The original Hausdorff algorithm is implemented as follows. Let A and B be two non-empty subsets in 3D space. $h(A, B)$ is the directed Hausdorff distance from A to B. First, for each element in A, it finds its nearest neighbour in B and the corresponding distance. Next, it finds the maximum value among these minimum distances to obtain $h(A, B)$. Then, repeat the same process by swapping sets A and B to get $h(B, A)$. Finally, the Hausdorff distance $H(A, B)$ is determined by taking the larger value between $h(A, B)$ and $h(B, A)$.

Our method proposes an efficient implementation of the Hausdorff algorithm on GPU. In our design, the distance calculation of the Hausdorff algorithm is implemented in parallel on GPU. The implementation is illustrated in Figure 8. The process begins with pairwise distance computation, where the distance between any pair of points P_i and P'_j is calculated in parallel. Then, the nearest neighbour search can be implemented in parallel for each point, obtaining distance from each point in one set to its closest point in the other set. The maximum distance of P_i in A to its nearest neighbour in B is the directed Hausdorff distance from A to B $h(A, B)$. Similarly, the maximum distance of P'_i in B to its nearest neighbour in A is the directed Hausdorff distance from B to A $h(B, A)$. The Hausdorff distance $H(A, B)$ is the maximum of $h(A, B)$ and $h(B, A)$.

DAWN uses distance thresholds in localized similarity analysis to identify redundant points. Partitions with Hausdorff distances below the threshold are considered similar, and the corresponding detection is skipped. Otherwise, a new detection is necessary. A smaller threshold imposes stricter similarity criteria, resulting in fewer partitions being considered similar and more frequent new detections. The similarity analysis starts with an initial threshold value t_0. This threshold is adaptively adjusted as $t'_0 = t_0 + \Delta t$, where Δt is the scene adjustment factor based on scene complexity. The similarity criteria become more stringent for partitions containing multiple objects in the previous frame and less stringent for those with fewer objects. The filtering ratio can be adjusted by tuning t_0 and Δt, enabling flexible latency-accuracy trade-offs for different application requirements.

V. Evaluation

A. Experiment Setup

The experiments are conducted on a high-performance desktop computer equipped with an Intel(R) Core(TM) i7-9700K CPU operating at 3.6 GHz, 64 GB of DDR4 RAM, and an NVIDIA GeForce RTX 2080Ti graphics card. We implement our method on OpenPCDet, a toolbox for point cloud object detection [25]. Our experiments are conducted on the dataset of KITTI [19]. We evaluate DAWN's performance by applying it to detection networks PointRCNN [8], PV-RCNN [10] and CasA-PV [26] using mean average precision (mAP) as the metric. The accuracy is calculated at the moderate difficulty level for the car class with 40 recall positions.

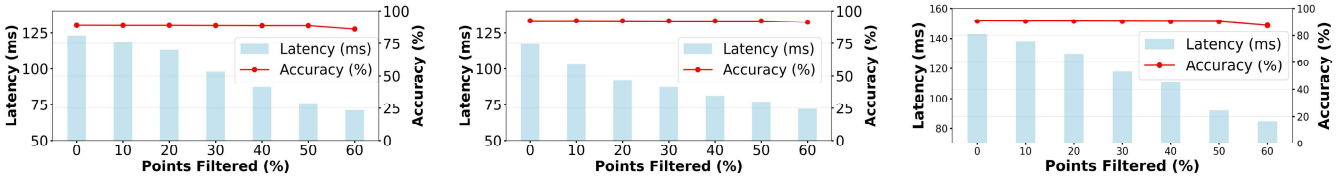
B. Performance Evaluation

Speedup and Accuracy Figure 9 shows the improvement of DAWN on networks PointRCNN [8],PV-RCNN [10] and CasA-PV [26]. A 50% filtering ratio means on average 50% of the input points require processing in each frame. The filtering ratio is adjusted by tuning the threshold in the similarity analysis. With 50% points filtered, PointRCNN [8] achieves $1.70\times$ speedup with 0.3 accuracy loss. PV-RCNN [10] demonstrates a $1.51\times$ speedup with 0.2 accuracy degradation when filtering 50% of points. CasA-PV [26] is accelerated by $1.56\times$ with 0.4 accuracy loss when filtering 50% of points. The detection accuracy shows significant degradation when more than 60% of points are filtered.

System Performance Analysis We evaluate the system efficiency improvements of networks optimized by DAWN. As demonstrated in Figure 10, PV-RCNN [10], CasA-PV [26], and PointRCNN [8] optimized by DAWN show lower GPU memory usage and GPU utilization. This improvement can be attributed to the filtering of redundant points and reduced computational workload.

C. Comparison with Related Work

We compare DAWN with QuickFPS [31] and Mesorasi [16] in Table I. The performance of DAWN is evaluated with 50% of points filtered. Prior works show limited applicability: QuickFPS [31] supports only point-based and point-voxel based detection networks, while Mesorasi [16] is confined to point-based detection networks. In contrast, DAWN is compatible with voxel-based, point-based, and point-voxel based networks. The dash ('-') in Table I indicates that the corresponding method cannot be applied to that network. On PointRCNN [8], DAWN achieves 13.6 FPS with 0.3



(a) PointRCNN [8] (b) PV-RCNN [10] (c) CasA-PV [26]

Fig. 9: Latency-accuracy trade-off under different point filtering ratios for PointRCNN [8], PV-RCNN [10] and CasA-PV [26].

accuracy loss, outperforming Mesorasi [16] (10.3 FPS with 1.0 accuracy loss) and QuickFPS [31] (9.9 FPS with 0.7 accuracy loss). For PV-RCNN [10], DAWN reaches 13.0 FPS with 0.2 accuracy loss, better than QuickFPS [31] (11.0 FPS with 0.7 loss). On CasA-PV [26], DAWN achieves 10.6 FPS with 0.4 accuracy loss, outperforming QuickFPS [31] which achieves 8.8 FPS with 0.8 accuracy loss. Additionally, DAWN enhances SECOND's performance [5] to 21.7 FPS with 0.5 accuracy loss, demonstrating its effectiveness across different network architectures.

Method	Voxel-based			Point-based			Point-voxel based					
	SECOND [5]			PointRCNN [8]			PV-RCNN [10]			CasA-PV [26]		
	Acc.	Lat.	FPS	Acc.	Lat.	FPS	Acc.	Lat.	FPS	Acc.	Lat.	FPS
Original	88.1	52ms	19.2	89.1	124ms	8.1	92.8	117ms	8.5	90.4	142ms	7.0
Mesorasi [16]	-	-	-	88.1	97ms	10.3	-	-	-	-	-	-
QuickFPS [31]	-	-	-	88.4	102ms	9.9	92.1	91ms	11.0	89.6	113ms	8.8
DAWN	87.6	46ms	**21.7**	88.8	74ms	**13.6**	92.6	77ms	**13.0**	90.0	94ms	**10.6**

TABLE I: Performance comparison with related work. Acc. stands for Accuracy, Lat. stands for Latency and FPS denotes frames processed per second. DAWN is evaluated with 50% of points filtered. Dash (-) indicates that the corresponding method cannot be applied to that network.

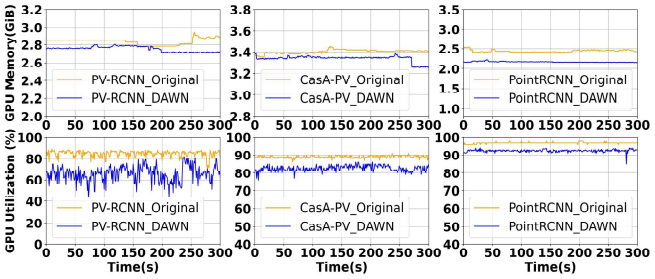


Fig. 10: Memory usage and utilization comparison between original and DAWN-optimized networks.

D. Partitioning Method Analysis

Distribution Analysis Figure 11 shows the granularity levels and accuracy performance of different partitioning methods. The granularity level defines how many points are processed for the similarity analysis. The histograms display the normalized point distribution on the log scale. In Figure 11 (a), fixed partitioning results in many empty partitions and other partitions containing the majority of points. Figure 11 (b) demonstrates that object-aware partitioning reduces large granularity (3000 to 6000). Figure 11 (c) shows that object-aware partitioning with axis-sorted selection ($n_g = 600$) minimizes mid-range granularity (2000 to 4000 points) while concentrating points around the target of $n_g = 600$.

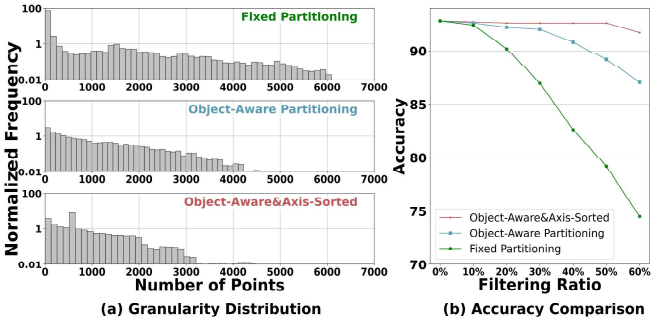


Fig. 11: The granularity distribution and accuracy comparison of different partitioning methods. The granularity is the number of points after partitioning and point selection.

Fragmentation Reduction Experimental results show that fixed partitioning results in 34.8% of objects being fragmented. Object-aware partitioning, defining boundaries based on previous detection results, reduces this fragmentation rate to 2.6%.

Accuracy Analysis Figure 11 (b) compares the accuracy of different partitioning methods to evaluate their ability to capture partial similarities. Filtering using object-aware partitioning demonstrates a much slower accuracy drop than filtering using fixed partitioning. The filtering with object-aware partitioning and axis-sorted point selection achieves the best performance, maintaining stable accuracy.

VI. Conclusion

This paper addresses the latency challenge of point cloud detection. We propose DAWN to leverage partial similarity between adjacent point cloud frames to accelerate detection. The object-aware partitioning and axis-sorted point selection optimize partitioning granularity to enable localized similarity comparison. DAWN filters redundant points by an efficient 3D similarity algorithm. It provides a flexible latency-accuracy trade-off to meet different application requirements. The experiment results show that DAWN improves state-of-the-art detection networks by 1.59× on average and up to 1.70×, with minimal accuracy loss.

REFERENCES

[1] Guo, Y., Wang, H., Hu, Q., Liu, H., Liu, L. and Bennamoun, M. (2020). Deep learning for 3d point clouds: A survey. IEEE transactions on pattern analysis and machine intelligence, 43(12), 4338-4364.

[2] Chen, X., Ma, H., Wan, J., Li, B., & Xia, T. (2017). Multi-view 3d object detection network for autonomous driving. In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 1907-1915).

[3] Mao, J., Shi, S., Wang, X., & Li, H. (2023). 3D object detection for autonomous driving: A comprehensive survey. International Journal of Computer Vision, 131(8), 1909-1963.

[4] Qian, R., Lai, X., & Li, X. (2022). 3D object detection for autonomous driving: A survey. Pattern Recognition, 130, 108796.

[5] Yan, Y., Mao, Y., & Li, B. (2018). Second: Sparsely embedded convolutional detection. Sensors, 18(10), 3337.

[6] Zhou, Y., & Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4490-4499).

[7] Deng, J., Shi, S., Li, P., Zhou, W., Zhang, Y., & Li, H. (2021, May). Voxel r-cnn: Towards high performance voxel-based 3d object detection. In Proceedings of the AAAI conference on artificial intelligence (Vol. 35, No. 2, pp. 1201-1209).

[8] Shi, S., Wang, X., & Li, H. (2019). Pointrcnn: 3d object proposal generation and detection from point cloud. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 770-779).

[9] Yang, Z., Sun, Y., Liu, S., & Jia, J. (2020). 3dssd: Point-based 3d single stage object detector. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 11040-11048).

[10] Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., & Li, H. (2020). Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10529-10538).

[11] Sheng, H., Cai, S., Liu, Y., Deng, B., Huang, J., Hua, X. S., & Zhao, M. J. (2021). Improving 3d object detection with channel-wise transformer. In Proceedings of the IEEE/CVF international conference on computer vision (pp. 2743-2752).

[12] Lyu, D., Li, Z., Chen, Y., Xu, N., & He, G. (2023, July). FLNA: An energy-efficient point cloud feature learning accelerator with dataflow decoupling. In 2023 60th ACM/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.

[13] Xu, T., Tian, B., & Zhu, Y. (2019, October). Tigris: Architecture and algorithms for 3d perception in point clouds. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 629-642).

[14] Yang, X., Fu, T., Dai, G., Zeng, S., Zhong, K., Hong, K., & Wang, Y. (2023, July). An efficient accelerator for point-based and voxel-based point cloud neural networks. In 2023 60th ACM/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.

[15] Ying, Z., Bhuyan, S., Kang, Y., Zhang, Y., Kandemir, M. T., & Das, C. R. (2023, June). EdgePC: Efficient deep learning analytics for point clouds on edge devices. In Proceedings of the 50th Annual International Symposium on Computer Architecture (pp. 1-14).

[16] Feng, Y., Tian, B., Xu, T., Whatmough, P., & Zhu, Y. (2020, October). Mesorasi: Architecture support for point cloud analytics via delayed-aggregation. In 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 1037-1050). IEEE.

[17] Zhang, J. F., & Zhang, Z. (2021, October). Point-x: A spatial-locality-aware architecture for energy-efficient graph-based point-cloud deep learning. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (pp. 1078-1090).

[18] Lin, Y., Zhang, Z., Tang, H., Wang, H., & Han, S. (2021, October). Pointacc: Efficient point cloud accelerator. In MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture (pp. 449-461).

[19] Geiger, A., Lenz, P. & Urtasun, R. (2012, June). Are we ready for autonomous driving? the kitti vision benchmark suite. In 2012 IEEE conference on computer vision and pattern recognition (pp. 3354-3361). IEEE.

[20] Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., ... & Beijbom, O. (2020). nuscenes: A multimodal dataset for autonomous driving. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 11621-11631).

[21] Wilson, B., Qi, W., Agarwal, T., Lambert, J., Singh, J., Khandelwal, S., ... & Hays, J. (2023). Argoverse 2: Next generation datasets for self-driving perception and forecasting. arXiv preprint arXiv:2301.00493.

[22] Yang, Z., Sun, Y., Liu, S., Shen, X., & Jia, J. (2018). Ipod: Intensive point-based object detector for point cloud. arXiv preprint arXiv:1812.05276.

[23] Shi, W., & Rajkumar, R. (2020). Point-gnn: Graph neural network for 3d object detection in a point cloud. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 1711-1719).

[24] Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 652-660).

[25] OpenPCDet Development Team. OpenPCDet: An Open-source Toolbox for 3D Object Detection from Point Clouds, 2020.

[26] Wu, H., Deng, J., Wen, C., Li, X., Wang, C., & Li, J. (2022). CasA: A cascade attention network for 3-D object detection from LiDAR point clouds. IEEE Transactions on Geoscience and Remote Sensing, 60, 1-11.

[27] Feng, Y., Hammonds, G., Gan, Y., & Zhu, Y. (2022, June). Crescent: taming memory irregularities for accelerating deep point cloud analytics. In Proceedings of the 49th Annual International Symposium on Computer Architecture (pp. 962-977).

[28] Feng, X., Tang, C., Zhang, Z., Sun, W., & Liu, Y. (2023, January). Semantic Guided Fine-grained Point Cloud Quantization Framework for 3D Object Detection. In Proceedings of the 28th Asia and South Pacific Design Automation Conference (pp. 390-395).

[29] Li, W., Qu, Y., Chen, G., Ma, Y., & Yu, B. (2021, January). TreeNet: Deep point cloud embedding for routing tree construction. In Proceedings of the 26th Asia and South Pacific Design Automation Conference (pp. 164-169).

[30] Lee, M., Park, S., Kim, H., Yoon, M., Lee, J., Choi, J. W., ... & Choi, J. (2024, March). SPADE: Sparse Pillar-based 3D Object Detection Accelerator for Autonomous Driving. In 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA) (pp. 454-467). IEEE.

[31] Han, M., Wang, L., Xiao, L., Zhang, H., Zhang, C., Xu, X. & Zhu, J. (2023). QuickFPS: Architecture and algorithm co-design for farthest point sampling in large-scale point clouds. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 42(11), 4011-4024.

[32] Zhao, P., Yuan, G., Cai, Y., Niu, W., Liu, Q., Wen, W., ... & Lin, X. (2021, December). Neural pruning search for real-time object detection of autonomous vehicles. In 2021 58th ACM/IEEE Design Automation Conference (DAC) (pp. 835-840). IEEE.

[33] Zheng, J., Jiang, H., Nie, X., Huang, Z., Chen, C., & Liu, Q. (2023, July). TiPU: A Spatial-Locality-Aware Near-Memory Tile Processing Unit for 3D Point Cloud Neural Network. In 2023 60th ACM/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.

[34] Tang, D., Sun, X., Guan, N., Kuo, T. W., & Xue, C. J. (2022, December). *p*LPAQ: Accelerating LPAQ Compression on FPGA. In 2022 International Conference on Field-Programmable Technology (ICFPT) (pp. 1-6). IEEE.

[35] Mao, Y., Cui, Y., Kuo, T. W., & Xue, C. J. (2022, April). Trace: A fast transformer-based general-purpose lossless compressor. In Proceedings of the ACM Web Conference 2022 (pp. 1829-1838).

[36] Liu, X., Song, Z., Chen, H., Li, X., & Liang, X. (2024, June). MoC: A Morton-Code-Based Fine-Grained Quantization for Accelerating Point Cloud Neural Networks. In Proceedings of the 61st ACM/IEEE Design Automation Conference (pp. 1-6).

[37] Yoon, H., & Kim, J. J. (2024, June). Fused Sampling and Grouping with Search Space Reduction for Efficient Point Cloud Acceleration. In Proceedings of the 61st ACM/IEEE Design Automation Conference (pp. 1-6).

[38] Mao, Y., Li, J., Cui, Y., & Xue, J. C. (2023, July). Faster and stronger lossless compression with optimized autoregressive framework. In 2023 60th ACM/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.

[39] Li, Jingzong, et al. "Moby: Empowering 2d models for efficient point cloud analytics on the edge." Proceedings of the 31st ACM International Conference on Multimedia. 2023.

Scaling Laws of Graph Neural Networks for Atomistic Materials Modeling*

Chaojian Li[4], Zhifan Ye[4], Massimiliano Lupo Pasini[1], Jong Youl Choi[2], Cheng Wan[4], and Prasanna Balaprakash[3]
[1]*Computational Sciences and Engineering Division, Oak Ridge National Laboratory*
[2]*Computer Science and Mathematics Division, Oak Ridge National Laboratory*
[3]*Computing and Computational Sciences Directorate, Oak Ridge National Laboratory*
[4] *Georgia Institute of Technology*
{cli851, zye327, cwan39}@gatech.edu, {lupopasinim, choij, pbalapra}@ornl.gov

***Abstract*—Atomistic materials modeling is a critical task with wide-ranging applications, from drug discovery to materials science, where accurate predictions of the target material property can lead to significant advancements in scientific discovery. Graph Neural Networks (GNNs) represent the state-of-the-art approach for modeling atomistic material data thanks to their capacity to capture complex relational structures. While machine learning performance has historically improved with larger models and datasets, GNNs for atomistic materials modeling remain relatively small compared to large language models (LLMs), which leverage billions of parameters and terabyte-scale datasets to achieve remarkable performance in their respective domains. To address this gap, we explore the scaling limits of GNNs for atomistic materials modeling by developing a foundational model with billions of parameters, trained on extensive datasets in terabyte-scale. Our approach incorporates techniques from LLM libraries to efficiently manage large-scale data and models, enabling both effective training and deployment of these large-scale GNN models. This work addresses three fundamental questions in scaling GNNs: the potential for scaling GNN model architectures, the effect of dataset size on model accuracy, and the applicability of LLM-inspired techniques to GNN architectures. Specifically, the outcomes of this study include (1) insights into the scaling laws for GNNs, highlighting the relationship between model size, dataset volume, and accuracy, (2) a foundational GNN model optimized for atomistic materials modeling, and (3) a GNN codebase enhanced with advanced LLM-based training techniques. Our findings lay the groundwork for large-scale GNNs with billions of parameters and terabyte-scale datasets, establishing a scalable pathway for future advancements in atomistic materials modeling.**

***Index Terms*—AI/ML Application and Infrastructure; AI/ML System and Platform Design.**

I. Introduction

Atomistic material modeling is essential for the discovery of new materials with target properties because it accelerates the process by predicting material properties with sufficiently accurate results using atomistic information, without requiring costly full first-principles calculations [22], [24]. Neural networks play an important role in atomistic material modeling by training on data collected from chemistry experiments and first-principles calculations, formalizing material modeling as a neural network inference process [1]. Among these neural networks, Graph Neural Networks (GNNs) [16], [30], [36] have emerged as the state-of-the-art (SOTA) approach because atomistic material structures can be naturally mapped onto a graph, where atoms and interatomic bonds are treated as the nodes and edges of a graph, respectively [4], [32].

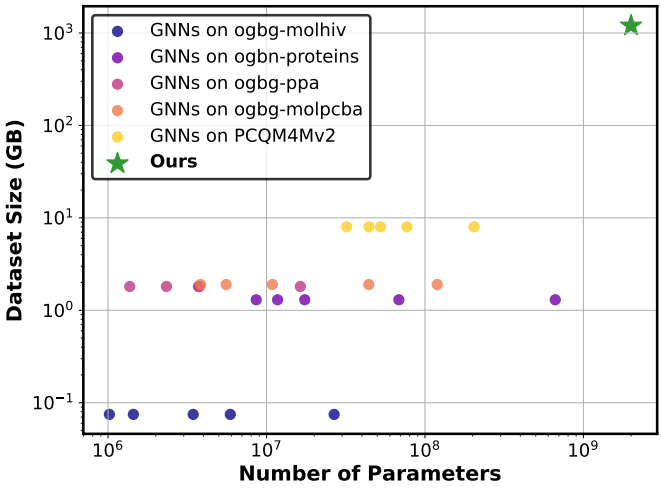


Fig. 1. Comparison of large-scale GNNs on multiple commonly-used biology/chemistry materials modeling datasets [12] with the foundational GNN developed in this work (indicated by a green star), after scaling both the model size and dataset size.

However, the great promise of GNNs for atomistic material modeling has not yet been fully realized because the potential of larger GNN models and more extensive graph datasets has not been fully exploited [19], [20], [32]. Specifically, as highlighted in recent studies on language and image-based neural networks [14], [42], larger neural networks trained on more comprehensive datasets tend to better capture the complex relationships in input data, leading to improved accuracy and reduced prediction errors. This scalability is often described by scaling laws, which indicate that increasing model size and

*This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

dataset size generally improves prediction quality. In contrast, GNNs for atomistic material modeling currently lag behind in terms of the scale of models and datasets used compared to language and image-based neural networks. For example, as shown in Fig. 1, prior scaled-up GNNs are typically trained on datasets in the range of megabytes or gigabytes and have millions of parameters [20], [24], whereas commonly used language models today are trained on terabyte-scale datasets and contain billions of parameters [33].

The aforementioned gap raises a series of fundamental questions about scaling up GNNs: (1) What is the potential of scaling GNN model architectures when the number of parameters reaches the billion level? (2) What is the effect of dataset size on achieved accuracy or prediction errors when scaled to the terabyte level? (3) Can prior techniques for scaling up distributed training of large-scale language and image-based neural networks also be applied to GNNs?

To address the questions raised above, we progressively scale up GNN model sizes from millions of parameters to billions, alongside a corresponding increase in the atomistic materials modeling dataset size to the terabyte level, as depicted in Fig. 1. In summary, our contributions are threefold:

- **Insight**: Extracted scaling laws for GNNs in atomistic materials modeling, offering guidance on the relationship between GNN model sizes, the amount of available training data, and the prediction error of material properties.
- **Model**: A foundational GNN model with billion-level parameters and terabyte-scale data, which, to the best of our knowledge, represents the largest GNN developed for atomistic materials modeling to date.
- **Infrastructure**: A scalable GNN training codebase integrated with SOTA distributed training techniques from language and image-based neural networks, enabling efficient training of billion-level GNNs with terabyte-scale atomistic materials modeling data.

II. Related Works

A. GNN in Atomistic Materials Modeling

GNNs have emerged as a powerful tool for atomistic materials modeling, offering state-of-the-art (SOTA) performance across various tasks in chemistry and materials science [2], [29]. These models leverage the inherent graph-like atomistic structures, where atoms are represented as nodes and chemical bonds as edges, allowing for a natural and intuitive representation of atomistic materials [40]. GNNs excel in capturing complex relationships between atoms and their local chemical environments, enabling them to learn meaningful representations that can be used to predict a wide range of atomistic materials' properties [23]. Recent advancements in GNN architectures have led to significant improvements in prediction accuracy, with models like EGNN [30] achieving chemical accuracy on benchmark datasets such as QM9 [28]. These models have demonstrated their ability to predict quantum mechanical properties, solubility, toxicity, and other crucial molecular characteristics with high precision. The success of GNNs in property prediction has opened up new possibilities for accelerating drug discovery, materials design, and other fields where understanding structure-property relationships is crucial. However, the challenge remains in scaling these models to handle even larger datasets and more complex atomistic structures, a gap this paper aims to address.

B. GNN Foundation Model

Graph Foundation Models (GFMs) represent a significant advancement in graph representation learning, designed to learn transferable representations that can generalize across diverse graph structures and tasks [19]. These models aim to overcome the traditional limitations of task-specific GNNs by learning universal graph representations that can be applied to previously unseen graphs and tasks. Notable examples include GraphAny [44], which successfully demonstrates the ability to perform node classification across multiple graphs with varying feature dimensions and class numbers. In the realm of atomistic materials modeling, HydraGNN-GFM [24] has emerged as a pioneering GFM architecture, specifically designed for large-scale scientific applications. HydraGNN-GFM's unique features include multi-task learning capabilities, flexible message passing neural network layers, and scalable distributed training support, allowing it to process hundreds of millions of graphs efficiently across thousands of GPUs. This scalability has been demonstrated through near-linear strong scaling performance on major supercomputing systems, processing over 154 million atomistic structures while maintaining high prediction accuracy. Motivated by the scalability demonstrated by HydraGNN-GFM, we adopt the same GNN model architecture for our scaling experiments in this work.

C. Scalable Training Techniques

Scalable training techniques are crucial for enabling GNNs to handle large-scale graphs that exceed the memory capacity of a single machine. Traditional distributed training approaches, widely adopted in deep learning [37], primarily rely on data parallelism, which distributes data batches across multiple devices while maintaining full model replicas on each device, synchronizing gradients during updates. Optimizer-level parallelism methods such as ZeRO [25], further reduce memory requirements by distributing optimizer states across devices. Conversely, model parallelism partitions the model architecture itself across devices, necessitating careful management of intermediate activations. Another memory-saving strategy, activation checkpointing, lowers memory usage by rematerializing intermediate activations during backward propagation [6]. For GNNs, distributed training introduces unique challenges due to the interconnected nature of graph data. Partition-based methods, which divide the graph into subgraphs distributed across devices, mitigate these challenges by distributing computational workloads but often incur significant communication overhead from exchanging neighbor features [43].

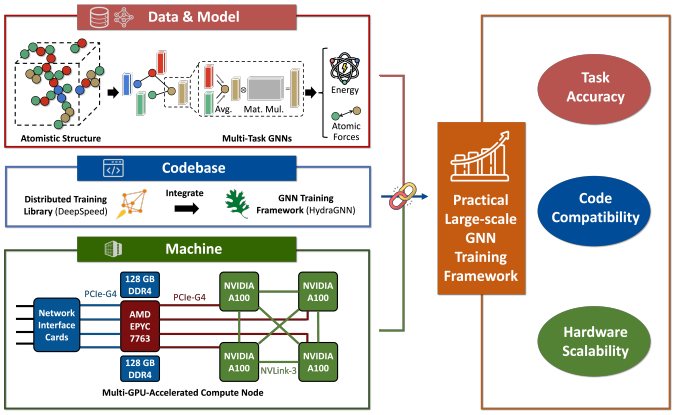


Fig. 2. An overview of the developed multi-stack infrastructure for scalable GNN training, integrating data and model configuration, refactored codebase, and multi-GPU machine setup. This unified framework aims to simultaneously improve GNN task accuracy through data-driven model design, ensure code modularity and reuse via codebase restructuring, and optimize training efficiency and scalability with a multi-GPU hardware architecture.

III. Infrastructure Setup

As summarized in Fig. 2, the entire infrastructure used for the scaling experiments includes not only the data and model but also the codebase supporting scalable GNN training and the high-performance computing (HPC) cluster.

A. Data

The dataset used in our scaling experiments is aggregated from multiple publicly available atomistic materials modeling datasets. As summarized in Tab. I, each dataset contains a different number of samples(graphs). Specifically, ANI1x [31] consists of up to 57 thousand distinct molecular configurations featuring the chemical elements C, H, N, and O. QM7-X [11] includes 42 physicochemical properties for approximately 4.2 million equilibrium and non-equilibrium structures of small organic molecules. OC2020-20M [4] is a subsampled set from the original OC2020 [4] dataset, spanning a range of oxide materials, coverages, and adsorbates. OC2022 [34] is an updated version of OC2020 [4], containing additional samples. MPTrj [13] focuses on atomistic structures of inorganic materials. Combining the five aforementioned data sources resulted in an aggregated dataset of 1.2 TB. We further sampled multiple datasets with sizes ranging from 0.1 TB to 1.2 TB from the original 1.2 TB dataset and used them in the scaling experiments. Following [20], [24], the corresponding tasks in the aggregated dataset are defined as predicting energy and atomic forces from atomistic structures represented in graph format. Specifically, the energy to be predicted is a property of the entire atomistic structure (graph), whereas the atomic forces are properties of individual atoms.

B. Model

To ensure equivalence with rotations, translations, reflections, and permutations in atomistic materials modeling, we select EGNN [30] as the GNN model type, which is specifically designed for predicting molecular properties. To align with the task definitions in the aggregated dataset used for our scaling experiments, we add two types of output heads on top of the EGNN models: one for graph-level property prediction and the other for node-level property prediction. To study the effectiveness of the backbone model independently of task types when scaling data and model sizes, we vary only the depth and width of the EGNN backbone during the scaling experiments. The hyperparameter settings used for training follow those in [24], e.g., all models are trained for 10 epochs.

C. Codebase

We conduct the scaling experiments using HydraGNN [21], a commonly used GNN training framework for scientific discovery. However, as discussed in Sec. V, while HydraGNN supports data parallelism, directly scaling the model and data to sizes beyond those previously encountered in the codebase can exceed memory limits, even on high-performance HPC machines equipped with A100 GPUs. To address this limitation, we integrate DeepSpeed, a well-known distributed learning library, into HydraGNN to alleviate memory constraints, enabling us to scale our models to billions of parameters and handle terabyte-level datasets. The corresponding modifications have been committed to the official HydraGNN repository at: https://github.com/ORNL/HydraGNN.

D. Machine

We utilize Perlmutter [3], an HPC cluster consisting of A100-accelerated compute nodes. Each node is equipped with an AMD EPYC 7763 CPU, 256 GB of DDR4 memory, and four NVIDIA A100 GPUs interconnected via NVLink-3. Additionally, to accelerate the data loading process on the HPC machines, we employ the ADIOS [9] scientific data management library and DDStore [7], a distributed data store that facilitates in-memory data transfer between processes. In particular, we use 32 compute nodes for training each model.

IV. Scaling Laws in GNN

To examine the scaling laws for GNNs in atomistic material modeling, we conducted experiments on scaling up GNN model sizes (Sec. IV-A), increasing dataset size (Sec. IV-B), and analyzing how model depth and width affect the final test loss (Sec. IV-C). Specifically, model scaling is achieved by increasing the number of neurons in each layer, while dataset scaling is accomplished by adding samples to the training set

TABLE I
Summary of the data sources of the aggregated dataset used in the scaling experiments.

Data Source	# of Nodes	# of Edges	# of Graphs	Size
ANI1x [31]	75,700,481	1,050,357,960	4,956,005	25 GB
QM7-X [11]	70,675,659	1,020,408,506	4,195,237	25 GB
OC2020-20M [4]	1,538,055,547	33,734,466,610	20,994,999	726 GB
OC2022 [34]	705,379,388	18,937,505,384	8,834,760	395 GB
MPTrj [13]	49,286,440	729,940,098	1,580,227	17 GB

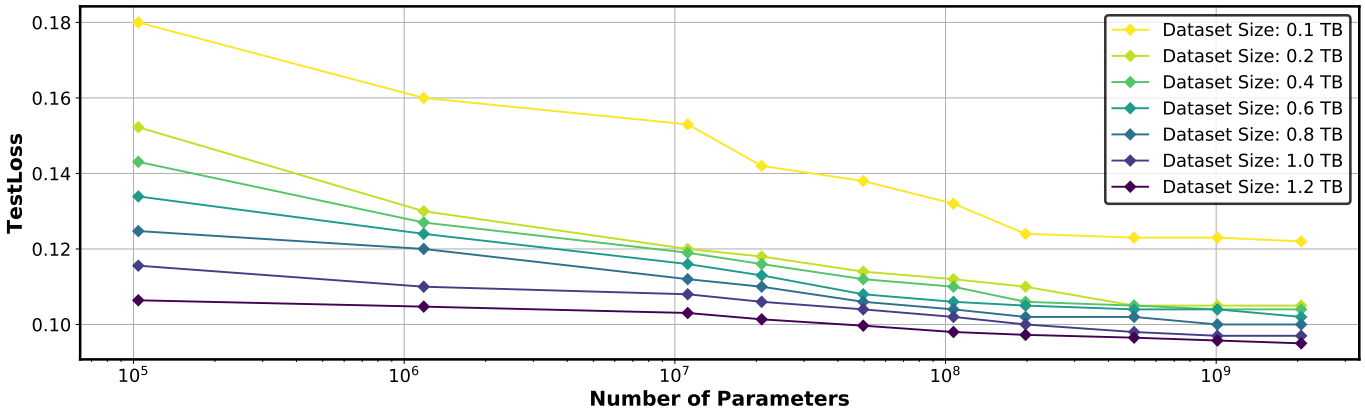


Fig. 3. The effect of scaling GNN **model** sizes across various dataset sizes on the final test loss.

from the aggregated 1.2 TB dataset. Additionally, the test loss for different model and dataset combinations is evaluated using the same held-out test set from the 1.2 TB dataset.

A. Model Scaling

As shown in Fig. 3, when the model size is scaled up by increasing the number of neurons in each layer, the test loss consistently decreases. This trend is observed across datasets of different sizes, ranging from 0.1 TB to 1.2 TB. However, unlike the observations in large-scale language and image-based models [14], [42], where the loss is nearly linear with the log-scale of the number of parameters, the decrease in test loss for GNNs in atomistic materials modeling shows diminishing returns as model size increases. We conjecture that this is due to architectural differences between Transformer models [35], commonly used in language and image-based tasks, and GNNs used in atomistic materials modeling. Specifically, Transformer models [35] rely on attention mechanisms, which can adaptively learn connections between different input samples (tokens). In contrast, GNN architectures, even advanced ones like EGNN [30] that account for rotational, translational, reflective, and permutational equivalences, are inherently limited by their locality constraints. These constraints restrict the ability to freely learn connections between any pair of nodes, reducing their overall learning capacity compared to Transformers.

Although there has been some exploration of applying Transformers to graphs or encoding graphs directly into Transformer tokens [35], [41], their effectiveness in atomistic materials modeling remains underexplored. Furthermore, many of these approaches are limited to solving simple graph statistics-related problems [10], [39].

In conclusion, **further scaling of GNN model sizes is a promising direction for improving the quality of atomistic materials modeling**. However, when scaling beyond 2 billion parameters, the limitations of current GNN architectures may become a bottleneck.

B. Data Scaling

As summarized in Fig. 4, the test loss of GNNs with varying parameter counts, ranging from 0.1 million to 2 billion, consistently decreases as more data becomes available for training. In particular, when the dataset size increases from 0.1 TB to 0.2 TB, there is a noticeable drop in test loss compared to the more gradual decreasing trend observed beyond 0.2 TB. We conjecture that this is due to the significant differences between the 0.1 TB subset and the full 1.2 TB dataset. The test set is sampled from the complete 1.2 TB dataset and remains fixed across all experiments. As a result,

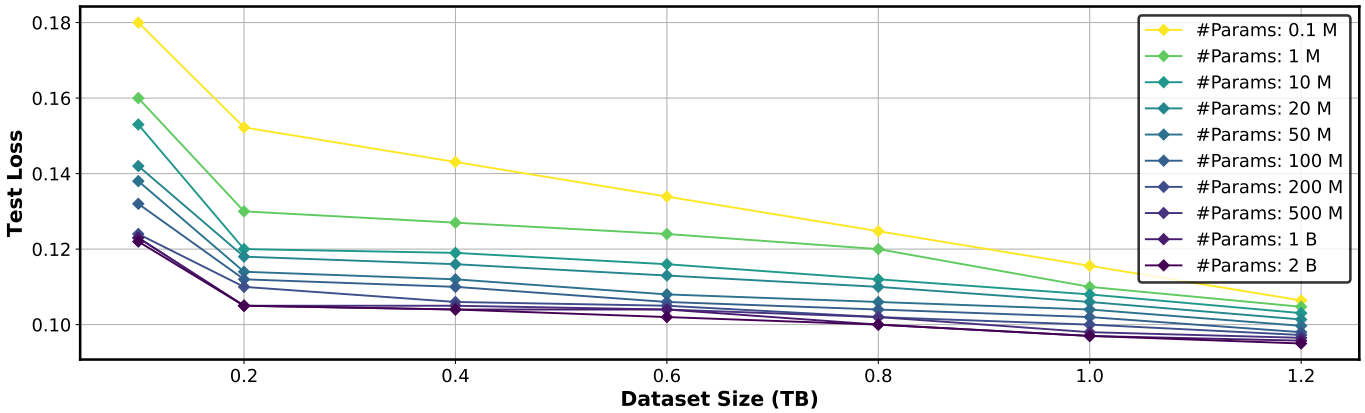


Fig. 4. The effect of scaling atomistic materials modeling **dataset** sizes across various GNN model sizes on the final test loss.

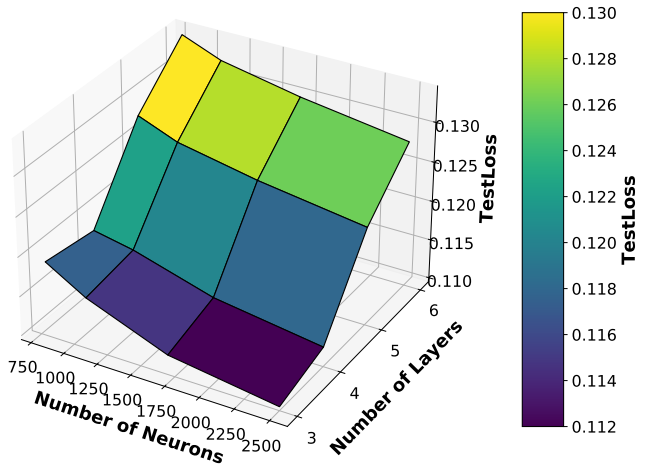


Fig. 5. Comparison of how scaling GNN model **depth** (i.e., number of layers) and **width** (i.e., number of neurons in each layer) affects the test loss when training on a dataset of 0.4 TB in size.

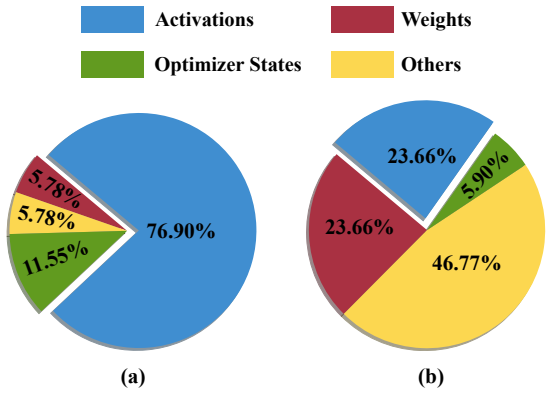


Fig. 6. Summary of the peak memory usage breakdown when training GNNs using (a) vanilla PyTorch-based HydraGNN [21] and (b) HydraGNN integrated with activation checkpointing and the ZeRO optimizer [25].

when only 0.1 TB is sampled for training, there is likely a mismatch between the distribution of the training dataset and the test set, leading to a relatively higher test loss for the 0.1 TB setting.

In contrast, as the dataset size gradually increases from 0.2 TB to 1.2 TB, the test loss decreases steadily and predictably, consistent with observations in existing large-scale language and image-based models [14], [42]. The more pronounced loss reduction, even at the 1.2 TB scale, suggests that **scaling data is more effective than scaling model size when both reach relatively large scales**, i.e., millions of parameters and terabytes of data.

However, it is important to note that scaling data is more challenging than scaling models. Simply increasing model depth and width produces a larger model, whereas scaling data involves tedious processes such as data collection, cleaning, and verification, as demonstrated in [24]. Therefore, the bottleneck in exploring the limits of scaling GNN models for atomistic materials modeling is similar to the bottleneck faced in scaling language and image-based models: the need for large-scale, high-quality data.

C. Model Depth vs. Width

As noted in previous works on scaling neural networks, a model's depth and width are two critical factors that influence the achieved test loss [14], [42]. Examining the effect of depth and width in GNN models is particularly important, as it is widely recognized that building deep GNNs is more challenging compared to convolutional neural networks or Transformer models [17], [18]. Motivated by prior observations on scaling depth and width, we conducted experiments to explore how these two factors affect relatively large-scale GNN models with 10 to 100 million parameters on a substantial 0.4 TB dataset.

As summarized in Fig. 5, our findings indicate that increasing model width—i.e., the number of neurons in each layer—consistently results in lower test loss. In contrast, increasing the number of layers beyond three leads to higher test loss, even when the total model size increases. We hypothesize that the over-smoothing [5] issue inherent to GNN architectures persists even at such large-scale dataset sizes and model capacities. This suggests that scaling GNN models by **increasing the number of neurons per layer is a more effective strategy than increasing the number of layers**.

V. Effectiveness of Training Techniques

To support training models with millions of parameters and datasets at terabyte scales, using the pure PyTorch-based data-distributed training in HydraGNN leads to out-of-memory issues, even on mainstream A100 GPUs. To address this, we investigated (1) the peak memory bottlenecks in large-scale GNN training (Sec. V-A) and (2) whether commonly used methods for scalability in large language models can be effectively applied to GNNs in atomistic material modeling (Sec. V-B and Sec. V-C).

A. Bottleneck Profiling

Limited memory capacity has long been a critical bottleneck in training large deep learning models [26]. To overcome this limitation, numerous memory-efficient training techniques have been proposed [6], [8], primarily targeting applications in image and text domains. However, the memory constraints in training large GNNs pose similar challenges, limiting their scalability [38]. To better understand and address these challenges, we conducted a memory bottleneck profiling analysis to evaluate the memory consumption in training foundational GNNs and to identify effective techniques for scalable GNN training. Since memory usage varies throughout the training process, our analysis focuses on profiling *peak memory usage*, defined as the highest memory consumption observed during training. This peak typically occurs across three stages: (1) the forward pass, (2) the backward pass, and (3) the weight updates performed by the optimizer.

TABLE II
REDUCTION IN PEAK MEMORY USAGE AND THE OVERHEAD OF TRAINING TIME AFTER ADOPTING ACTIVATION CHECKPOINTING AND ZERO OPTIMIZER.

Setting	Relative Peak Memory	Relative Training Time
Vanilla Pytorch	100%	100%
+ Activation Checkpointing	42%	110%
+ ZeRO Optimizer	27%	133%

Fig. 6 (a) illustrates the peak memory usage breakdown during GNN training without any memory-efficient optimizations. The results indicate that the peak memory usage arises at the start of the backward pass. The breakdown highlights two key insights: (1) **Activations** dominate peak memory usage, accounting for 76.90% of the total. These tensors store intermediate values computed during the forward pass and are essential for gradient computation during the backward pass; (2) **Optimizer states** represent the second largest contributor to peak memory usage. These states, maintained by the Adam optimizer [15], include momentum vectors, which require storage equivalent to twice the size of the model weights.

B. Activation Checkpointing

To mitigate the substantial memory overhead caused by activation tensors, we implemented the activation checkpointing technique [6] in HydraGNN framework [21]. This method reduces memory consumption by selectively recomputing partial activations during the backward pass, rather than storing all intermediate activations generated during the forward pass. By applying this technique, activation tensors are no longer the dominant contributor to memory usage, resulting in a significant 58% reduction in peak memory usage. Following this optimization, the new peak memory usage shifts to the weight update phase. However, as shown in Tab. II, activation checkpointing comes at the cost of a 10% increase in training latency due to the overhead introduced by recomputation.

C. ZeRO Optimizer

After adopting activation checkpointing, optimizer states became the largest contributor to peak memory usage. To address this, we incorporated the ZeRO optimizer [25] by adding DeepSpeed library [27] in HydraGNN framework [21], which reduces memory demands by partitioning optimizer states across multiple GPUs instead of duplicating them on each device. With four GPUs within one compute node, this approach achieved a 36% reduction in peak memory usage compared to using activation checkpointing alone, as shown in Fig. 6. However, as elaborated in Tab. II, this optimization introduces additional cross-GPU communication overhead, leading to a 133% increased training runtime.

VI. IMPACT OF THE SCALED MODELS AND THE INFRASTRUCTURE

As demonstrated in Sec. IV-A, the scaled-up GNN models demonstrate improved accuracy over smaller versions. This improvement highlights the potential of scaling in addressing the complexities of molecular property prediction tasks. The enhanced performance of these models could advance a range of scientific applications. For example, in material discovery, these models can enable rapid exploration of vast compositional and structural spaces, predicting key properties such as mechanical strength, thermal conductivity, and electronic behavior. By predicting critical properties, this capability accelerates the identification of novel materials for energy storage, catalysis, and sustainable manufacturing. Similarly, in drug design, the enhanced predictive power of these models can facilitate the identification of drug candidates with desired pharmacological properties, streamlining the traditionally time-consuming and resource-intensive process of lead optimization. Beyond these applications, the techniques and practices of the delivered infrastructure lay a foundation for scaling GNN models further and adapting them to broader applications in molecular science and other domains including biology, sociology, and more.

VII. CONCLUSION

This work bridges the gap between current GNNs for atomistic materials modeling and advances in scalable training techniques and scaling laws observed in large language and image-based models. By scaling GNNs to billions of parameters and terabyte-level datasets, we achieve significant improvements in predicting material properties. Our findings uncover scaling laws for GNNs, emphasizing the relationship between model size, dataset volume, and prediction accuracy, and establish a new benchmark in this field. The extracted insights, large-scale foundation models, and infrastructure developed in this work enable efficient handling of large-scale models and data, advancing atomistic materials modeling and expanding scientific applications.

ACKNOWLEDGEMENTS

Massimiliano Lupo Pasini would like to thank Dr. Vladimir Protopopescu for his valuable feedback in the preparation of the manuscript. This research is sponsored by the Artificial Intelligence Initiative as part of the Laboratory Directed Research and Development (LDRD) Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the US Department of Energy under contract DE-AC05-00OR22725. This work used resources of the Oak Ridge Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy, under INCITE award CPH161. This work also used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231, under awards ERCAP0027259 and ERCAP0030519. This work was supported in part by the National Science Foundation (NSF) (Award ID: 2400511 and 2016727), the Department of Health and Human Services Advanced Research Projects Agency for Health (ARPA-H) under Award Number AY1AX000003, and CoCoSys, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA.

REFERENCES

[1] R. M. Balabin and E. I. Lomakina, "Neural network approach to quantum-chemistry data: Accurate prediction of density functional theory energies," *The journal of chemical physics*, vol. 131, no. 7, 2009.

[2] D. Buterez, J. P. Janet, S. J. Kiddle, D. Oglic, and P. Lió, "Transfer learning with graph neural networks for improved molecular property prediction in the multi-fidelity setting," *Nature Communications*, vol. 15, no. 1, p. 1517, 2024.

[3] N. E. R. S. C. Center, "Perlmutter supercomputer," 2022. [Online]. Available: https://www.nersc.gov/systems/perlmutter/

[4] L. Chanussot, A. Das, S. Goyal, T. Lavril, M. Shuaibi, M. Riviere, K. Tran, J. Heras-Domingo, C. Ho, W. Hu *et al.*, "Open catalyst 2020 (oc20) dataset and community challenges," *Acs Catalysis*, vol. 11, no. 10, pp. 6059–6072, 2021.

[5] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 3438–3445.

[6] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," *arXiv:1604.06174*, 2016.

[7] J. Y. Choi, M. Lupo Pasini, P. Zhang, K. Mehta, F. Liu, J. Bae, and K. Ibrahim, "Ddstore: Distributed data store for scalable training of graph neural networks on large atomistic modeling datasets," in *Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 941–950.

[8] J. Duan, S. Zhang, Z. Wang, L. Jiang, W. Qu, Q. Hu, G. Wang, Q. Weng, H. Yan, X. Zhang, X. Qiu, D. Lin, Y. Wen, X. Jin, T. Zhang, and P. Sun, "Efficient training of large language models on distributed infrastructures: A survey," 2024.

[9] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck *et al.*, "Adios 2: The adaptable input output system. a framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.

[10] J. Guo, L. Du, H. Liu, M. Zhou, X. He, and S. Han, "Gpt4graph: Can large language models understand graph structured data? an empirical evaluation and benchmarking," *arXiv:2305.15066*, 2023.

[11] J. Hoja, L. Medrano Sandonas, B. G. Ernst, A. Vazquez-Mayagoitia, R. A. DiStasio Jr, and A. Tkatchenko, "Qm7-x, a comprehensive dataset of quantum-mechanical properties spanning the chemical space of small organic molecules," *Scientific data*, vol. 8, no. 1, p. 43, 2021.

[12] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *Advances in neural information processing systems*, vol. 33, pp. 22 118–22 133, 2020.

[13] A. Jain, S. P. Ong, G. Hautier, W. Chen, W. D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder *et al.*, "Commentary: The materials project: A materials genome approach to accelerating materials innovation," *APL materials*, vol. 1, no. 1, 2013.

[14] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv:2001.08361*, 2020.

[15] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv:1412.6980*, 2014.

[16] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv:1609.02907*, 2016.

[17] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9267–9276.

[18] G. Li, C. Xiong, A. Thabet, and B. Ghanem, "Deepergcn: All you need to train deeper gcns," *arXiv:2006.07739*, 2020.

[19] J. Liu, C. Yang, Z. Lu, J. Chen, Y. Li, M. Zhang, T. Bai, Y. Fang, L. Sun, P. S. Yu *et al.*, "Towards graph foundation models: A survey and beyond," *arXiv:2310.11829*, 2023.

[20] J. Liu, H. Mao, Z. Chen, T. Zhao, N. Shah, and J. Tang, "Neural scaling laws on graphs," *arXiv:2402.02054*, 2024.

[21] M. Lupo Pasini, S. T. Reeve, P. Zhang, J. Y. Choi, M. Lupo Pasini, S. T. Reeve, P. Zhang, and J. Y. Choi, "Hydragnn," oct 2021.

[22] M. Lupo Pasini, G. Samolyuk, M. Eisenbach, J. Y. Choi, J. Yin, and Y. Yang, "First-principles data for solid solution niobium-tantalum-vanadium alloys with body-centered-cubic structures," *Scientific Data*, vol. 11, no. 1, p. 907, 2024.

[23] H. Ma, Y. Bian, Y. Rong, W. Huang, T. Xu, W. Xie, G. Ye, and J. Huang, "Cross-dependent graph neural networks for molecular property prediction," *Bioinformatics*, vol. 38, no. 7, pp. 2003–2009, 2022.

[24] M. L. Pasini, J. Y. Choi, K. Mehta, P. Zhang, D. Rogers, J. Bae, K. Z. Ibrahim, A. M. Aji, K. W. Schulz, J. Polo *et al.*, "Scalable training of graph foundation models for atomistic materials modeling: A case study with hydragnn," *arXiv:2406.12909*, 2024.

[25] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20*. IEEE.

[26] S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," 2021.

[27] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3505–3506.

[28] P. Reiser, M. Neubert, A. Eberhard, L. Torresi, C. Zhou, C. Shao, H. Metni, C. van Hoesel, H. Schopmans, T. Sommer *et al.*, "Graph neural networks for materials science and chemistry," *Communications Materials*, vol. 3, no. 1, p. 93, 2022.

[29] J. G. Rittig, Q. Gao, M. Dahmen, A. Mitsos, and A. M. Schweidtmann, "Graph neural networks for the prediction of molecular structure–property relationships," 2023.

[30] V. G. Satorras, E. Hoogeboom, and M. Welling, "E (n) equivariant graph neural networks," in *International conference on machine learning*. PMLR, 2021, pp. 9323–9332.

[31] J. S. Smith, R. Zubatyuk, B. Nebgen, N. Lubbers, K. Barros, A. E. Roitberg, O. Isayev, and S. Tretiak, "The ani-1ccx and ani-1x data sets, coupled-cluster and density functional theory properties for molecules," *Scientific data*, vol. 7, no. 1, p. 134, 2020.

[32] M. Sypetkowski, F. Wenkel, F. Poursafaei, N. Dickson, K. Suri, P. Fradkin, and D. Beaini, "On the scalability of gnns for molecular graphs," *arXiv:2404.11568*, 2024.

[33] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv:2302.13971*, 2023.

[34] R. Tran, J. Lan, M. Shuaibi, B. M. Wood, S. Goyal, A. Das, J. Heras-Domingo, A. Kolluru, A. Rizvi, N. Shoghi *et al.*, "The open catalyst 2022 (oc22) dataset and challenges for oxide electrocatalysts," *ACS Catalysis*, vol. 13, no. 5, pp. 3066–3084, 2023.

[35] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[36] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv:1710.10903*, 2017.

[37] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A survey on distributed machine learning," *Acm computing surveys (csur)*, vol. 53, no. 2, pp. 1–33, 2020.

[38] C. Wan, Y. Li, A. Li, N. S. Kim, and Y. Lin, "Bns-gcn: Efficient full-graph training of graph convolutional networks with partition-parallelism and random boundary node sampling," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 673–693, 2022.

[39] L. Wei, Z. He, H. Zhao, and Q. Yao, "Unleashing the power of graph learning through llm-based autonomous agents," *arXiv:2309.04565*, 2023.

[40] Z. Wu, J. Wang, H. Du, D. Jiang, Y. Kang, D. Li, P. Pan, Y. Deng, D. Cao, C.-Y. Hsieh *et al.*, "Chemistry-intuitive explanation of graph neural networks for molecular property prediction with substructure masking," *Nature Communications*, vol. 14, no. 1, p. 2585, 2023.

[41] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*.

[42] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 12 104–12 113.

[43] S. Zhang, A. Sohrabizadeh, C. Wan, Z. Huang, Z. Hu, Y. Wang, J. Cong, Y. Sun *et al.*, "A survey on graph neural network acceleration: Algorithms, systems, and customized hardware," *arXiv:2306.14052*, 2023.

[44] J. Zhao, H. Mostafa, M. Galkin, M. Bronstein, Z. Zhu, and J. Tang, "Graphany: A foundation model for node classification on any graph," *arXiv:2405.20445*, 2024.

Espresso: Exploiting the Sparsity Property in Event Sensors with Spatiotemporal Ordering

Leshan Li†, Hongyi Li†, Qingyuan Yang, Mingtao Ou, Rong Zhao*, and Xinglong Ji*
Tsinghua University, Beijing 100084, China †Contribute equally
*Corresponding Authors (xinglongji@tsinghua.edu.cn, r_zhao@tsinghua.edu.cn)

***Abstract*—Event-based vision sensors are novel cameras inspired by human eyes, capable of capturing the rapid motion of objects with the high-speed sparse event stream. However, it is challenging to efficiently stream process event data without demolishing its sparsity. In this paper, we design Espresso, an architecture for event-based vision processing that can efficiently stream spatiotemporal events while preserving sparsity. We implement Espresso on the FPGA platform and design experiments to compare the performance with embedded GPU and line-buffer-based accelerator. In real-world scenarios, Espresso achieves throughput up to 5000 fps, which is 5.1× higher than the embedded GPU.**

***Index Terms*—Event-based Vision Sensor, Accelerator, Sparsity, Dynamic Vision**

I. Introduction

Event cameras are bio-inspired imaging sensors that capture the changes in light intensity in the scene with high temporal resolution and low latency. Unlike traditional frame-based cameras, event cameras output a stream of asynchronous events, each of which contains the spatial location and the timestamp of the change in the scene [18]. State-of-the-art bio-inspired cameras can integrate events to generate images with a frame rate of up to 10,000 fps [10], [22]. The high temporal resolution and low latency make event data suitable for vision processing in high-speed motion scenes such as autonomous driving and robotics [13]. In addition, the sparsity of event data can significantly reduce the amount of data to be processed and stored, which is believed to benefit edge devices with limited resources and power [14].

However, the events generated by typical event cameras like DVS [18] are ordered temporally without the spatial correlation between events. Such temporal order hinders the efficient processing of event data because events without spatial correlation can hardly be processed in the stream. For traditional frame-based images with spatially continuous pixels (Fig. 1A), stream processing is a widely applied method to deploy vision algorithms, a typical design being the line-buffer [5], [20], [21], which enables spatially adjacent pixels to be streamed continuously into the computation blocks and achieves very high level of data reusing and very low latency as a result (processing each pixel takes only one clock cycle). Stream processing has also been proved promising for efficient processing on sparse data [16]. However, such a design struggles to work for events in temporal order. As for traditional vision processing algorithms which typically work on groups of neighboring pixels (such as 3×3 Sobel or 5×5 Gaussian), spatially out-of-order events can hardly be directly streamed into the computation blocks since the arrival of a single event offers no information about its vicinity (Fig. 1B).

Current works on event-based vision processing mainly follow two methods: 1. integrate the events to traditional frame-based dense images and pose frame-based vision algorithms on them [1], [2], [15]; 2. design novel vision algorithms customized to temporal-order event stream for real-time stream processing on events [4], [7]. The former can be aided with architecture design like line-buffer to achieve relatively low latency, but the act of framing inevitably demolishes the sparsity of event data (Fig. 1C), which hinders the further reduction of latency and memory. The latter maintains sparsity of event data, by which the latency can be cut down to 130ns/event, but the fundamental different mechanisms of algorithms may cause incorrect results under certain conditions (overall false rate over 10% on event-camera dataset [4]). Recently, some works on sensors have noticed the shortcomings of temporal-ordering events and have been trying to integrate spatial features directly into the hardware to output events in spatiotemporal order (events within every frame are in spatial order) [22] (Fig. 1.D). However, **to the best of our knowledge, there has been no dedicated design to fully utilize such a spatiotemporal order feature.**

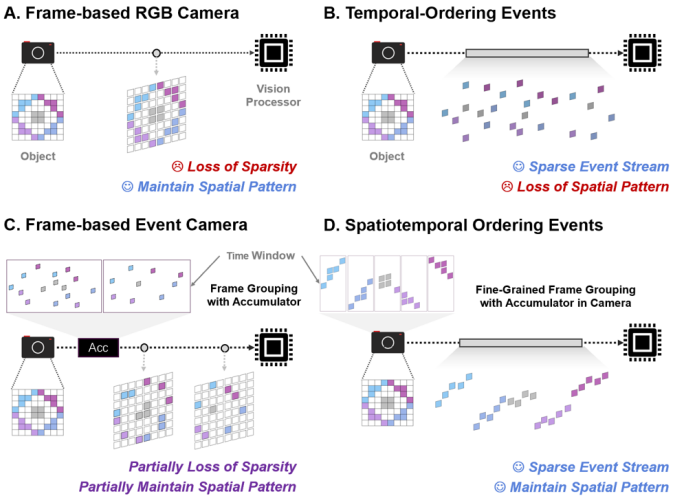


Fig. 1. The motivation of Espresso: the variation of event stream coding.

To address this issue, we propose Espresso, a vision processing architecture for spatiotemporal ordering events, which al-

lows the flexible implementation of traditional computer vision algorithms. Espresso employs an order-maintaining method and is able to efficiently stream process events. Besides, Espresso completely preserves and leverages the sparsity of event data by skipping computation on zeros and saving the memory of zeros. As a result, Espresso achieves high-level data reusing and very low latency. Specifically, our main contributions are as follows:

- We propose the N-Pending-FIFOs design to solve the challenge of order mismatching between input and output streams, enabling the efficient stream process of spatiotemporal ordering events.
- We propose the Shift-Hash Table design for the storing of events, which greatly reduces the memory access latency and preserves the sparsity of event data for low memory consumption.
- Based on N-Pending-FIFOs and Shift-Hash Table, we build Event-Scheduler which takes in events and outputs windows. Applying the Event-Scheduler to each layer of a vision algorithm, we establish the Espresso architecture to achieve efficient event stream processing.

We implement Espresso on an embedded FPGA platform and evaluate its performance. As a result, Espresso achieves a speedup of 5.1× than Jetson Orin NX (with Cuda-based OpenCV) and 3.3× than the line-buffer-based accelerator on FPGA. Such design sheds light on efficient dynamic vision processing with both low overhead and high speed.

II. Background

A. Event Camera

Event cameras are bio-inspired imaging sensors that can respond to the change in light intensity. Instead of generating frame-based images at a fixed rate, event cameras asynchronously detect pixel brightness change and generate an event stream, each event encoded with its position, timestamp, and value [13]. Compared to traditional cameras, event cameras offer several key advantages: 1. High temporal resolution (less than $1\mu s$) and high dynamic range (up to 140dB), enabling event cameras to perform effectively in high-speed or high-dynamic-range scenarios such as high-speed drones and autonomous driving [8], [13]. 2. Low power consumption and the inherent sparsity of event data, making event cameras well-suited for edge computing devices with limited power and resources [14] [8]. State-of-the-art event cameras can generate 4.6G events per second [12] or up to 10K event frames per second [10]. Some works have noticed the shortcomings of temporal-ordering event streams and integrated spatial features to generate spatiotemporal-ordering event streams [22]. Spatiotemporal order potentially enables the efficient stream processing of events, which is regarded as the trend of event-based sensor development. However, this feature has not been well utilized in computing devices.

B. Event-based Vision Processing

Since the invention of event cameras, plenty of studies on event-based vision processing have been carried out. [15] track the 6-DoF fast camera motions based on event frame using a geometric method. [2] fuses measurement from the event camera with IMU, realizing Visual Inertial Odometry on the event camera dataset. [1] further combines standard frame with events and inertial measurement and achieves robust visual SLAM with high accuracy and high speed. Recently, there have also been works applying DNN to event-based vision algorithm [17]. However, most of these works pre-process the events by accumulating them into frame-like representation, which demolishes the asynchronous and sparse feature of event data. There have also been works to avoid accumulating events to frames, trying to further exploit the feature of event data by software or hardware design. [7] proposes a novel feature detection algorithm that can process events asynchronously using a Surface of Active Events. [4] further develops a novel tracking algorithm, and achieves real-time process on event (130ns/event for detection and 1970ns/event for tracking) on CPU. Unfortunately, the novel processing mechanism for temporal-ordering events sees a drop in accuracy. Some works also try to deploy spiking neural networks (SNN) on neuromorphic hardware such as SpiNNaker [9], TrueNorth [3], and Loihi [11], in order to accelerate the event processing. However, the inference latency of the heavy network still proves unsatisfactory (105ms on DVS Gesture Recognition dataset [14]).

III. Method

Spatiotemporal ordering events maintain spatial order, which brings possibilities for efficient streamed processing. Fig. 2 illustrates the ideal architecture for spatiotemporal ordering events processing. For stream processing of dense images, previous works use the line-buffer shown in Fig. 2A to realize pixel-level streaming and pipelining. For the spatiotemporal ordering events, in order to maintain the sparsity in a frame-based image, all zero points are skipped and each event contains the value (Val) and the position (Pos) (Fig. 2B).

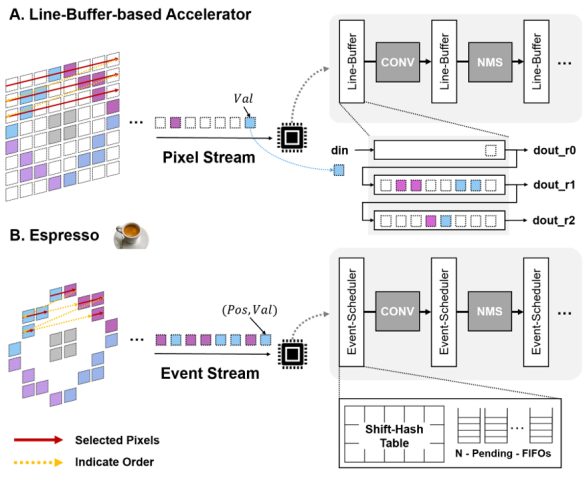


Fig. 2. Design goal of Espresso.

However, compared with the processing of dense pixel streams, two key challenges prevent us from achieving the

efficiency of stream processing and fully utilizing the sparsity of event cameras at the same time. Fig. 3 concludes these two issues:

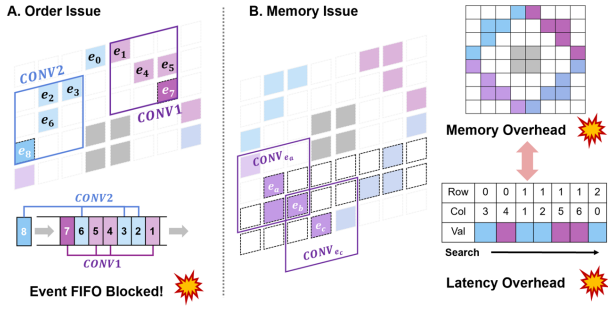


Fig. 3. Challenges of sparse event stream processing.

1) **Order Issue**: The ideal stream processing hardware implements communication between computational operators based on FIFOs [6]. However, for the convolution (CONV) example in Fig. 3A, when e_7 enters the queue[1], all the events for CONV1 are ready. But at this time, e_2 is still waiting for e_8 to complete the subsequent CONV2, and therefore it cannot exit the queue immediately, the FIFO is blocked. In a nutshell, *order mismatching exists between input and output stream*.
2) **Memory Issue**: As shown in Fig. 3B, e_b is needed for both e_a centered CONV and e_c centered CONV, indicating searching for previous events is inevitable. *The sparse memory of event data with discontinuous spatial addresses implies an address-matching process is possibly required during memory access*, which may bring great latency overhead.

For the order issue, we bridge the gap between input order and output order by maintaining the "Window Address" (WinAddr) order with N-Pending-FIFOs in section III-A. For the memory issue, we develop a hash-based data structure, dubbed as Shift-Hash Table in section III-B to balance the trade-off between memory and latency overhead. Finally, section III-C gives the whole picture of the collaboration between N-Pending-FIFOS and Shift-Hash Tables.

A. N-Pending-FIFOs for Order Issue

Fortunately, we note that window sliding is the most widely used computational scheme in vision processing with the following two advantages:

1) Fixed window size for a certain algorithm.
2) If window sliding is ordered, then for a particular pixel on an image, the windows that sweep over that pixel remain in the same order spatially.

Fig. 4A illustrates the second advantage. We use ★ to represent the event, ♦ to mark the center of the window sliding over the ★. WinAddr is defined as the address of ♦. For each ★, the number of its ♦ is the same as the window size.

[1]The subscript of e_i indicates the order in which the events arrive.

To solve the order issue, we leverage the feature of window sliding and design hardware of N rows of Pending FIFOs. A key design philosophy is: **For each event ★, focus only on the order and calculation of its associated WinAddr ♦.**

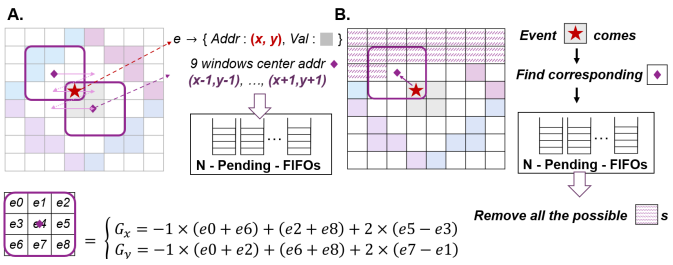


Fig. 4. The relationship between event order at an image view and N-Pending-FIFOs operation.

For the case of 3×3 sobel kernel (a computer vision operator widely used in edge detection [19]) in Fig. 4A, when a new event e (marked as ★) with spatial address (x, y) arrives, the nine associated WinAddrs (marked as ♦) will be triggered, pushed into the N-Pending-FIFOs. This operation implements the conversion of all subsequent operations from the event perspective to the WinAddr perspective, essentially **updating all event-related calculations rather than the event itself**.

The pop mechanism of N-Pending-FIFOs is also event-driven, as shown in Fig. 4B. For the coming events ★, all the WinAddr within the N-Pending-FIFOs with a smaller address than the smallest address of the ★'s ♦s will be popped. Because the arrival of the event ★ is spatially ordered, so is its smallest ♦. The arrival of the event ★ reveals that locations smaller than the address of ♦ will no longer be calculated in subsequent computations, and can be removed from the FIFOs.

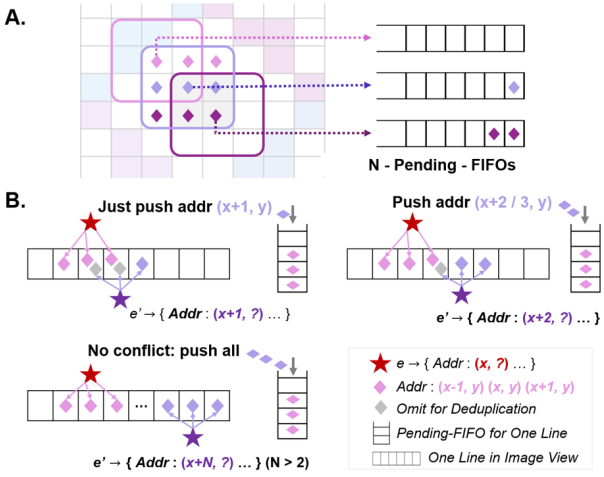


Fig. 5. Mechanism of N-Pending-FIFOs in details.

The operation above successfully bridges the gap between input order and output order. However, a single FIFO still faces the problems mentioned in Fig. 3A, and a more dedicated design of the FIFO is also needed. Our design idea is to

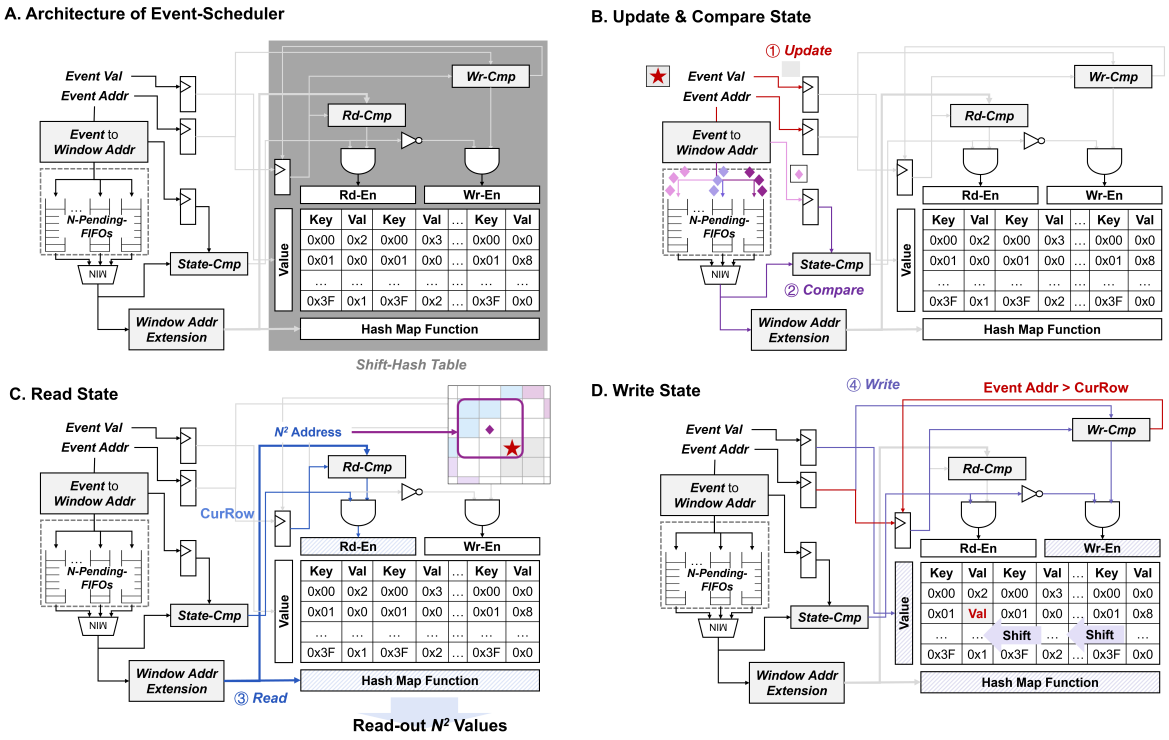


Fig. 6. The design of Event-Scheduler.

simplify the discussion of the problem by decomposing the order in two-dimensional space into one dimension.

Fig. 5 shows the design details of the N-pending-FIFOs. For the Sobel kernel in Fig. 5A, there should be 3 Pending FIFOs. Each of the N FIFOs is responsible for storing the addresses of one row of the WinAddrs ♦ triggered by an event. The advantages of this design are two-fold:

1) By comparing the current event's spatial address with the last event's spatial address, we can design logic to avoid repeated WinAddr in each FIFO (Fig. 5B).
2) Besides, the window addresses in each FIFO will always be in spatial order for both push and pop.

Therefore, we can simply complete the pop operation by comparing the addresses at the head of the FIFO, avoiding complex address lookups and searches.

In summary, we use WinAddr as the medium, supplemented by N-Pending-FIFOs, to realize a solution to the problem of inconsistent order of input and output.

B. Shift-Hash Table for Memory Issue

To solve the memory challenge, the Shift-Hash Table, an N-row hash table based on the shift register, is designed. Considering the sparsity of events, we set the length of each row less than the image width. The privilege of only N rows instead of the whole image being necessary to store also comes from the stream processing mechanism.

Specifically, upon the arrival of event ★ with spatial address (x, y), all the computation of pixels with address row less than $y - N$ has been completed, so that the memory of pixels over N rows ahead of ★ can be released. The releasing process is implemented by the shift register design in practice. The above design avoids excessive memory overhead.

TABLE I
OVERALL AND MEMORY READ LATENCY PERFORMANCE OF HASH AND NON-HASH DESIGN

	Sobel3		Sobel3 + Gaussian5	
	Non-Hash	Hash	Non-Hash	Hash
Overall Latency	$209\mu s$	$66\mu s$	$524\mu s$	$112\mu s$
Mem-Rd Latency	$137\mu s$	$11\mu s$	$422\mu s$	$38\mu s$
Ratio	**65.6%**	**16.7%**	**80.5%**	**33.9%**

Another important benefit brought by the Shift-Hash Table is the low latency of memory reading, which allows the reading of $N \times N$ values within one clock cycle. Compared with other non-hash discontinuous memory design which requires an address-matching process for memory read, the latency improvement can be substantial. As Tab. I shows, for example only containing Sobel3, memory based on non-hash design causes much greater latency overhead than Hash design. The case is even worse when the number of layers grows, where the memory read process accounts for over 80% of the overall latency. On the contrary, the shift-hash design achieves an acceptable latency overhead, an overall speedup being $3.2\times$ and $4.7\times$ faster than the non-hash design.

From the perspective of the system, the Mem-Rd latency, which directly influences the average time for processing one event, can be of great influence on the overall latency, and the Shift-Hash Table which limits Mem-Rd latency to 1 clock cycle is the optimal choice in this design.

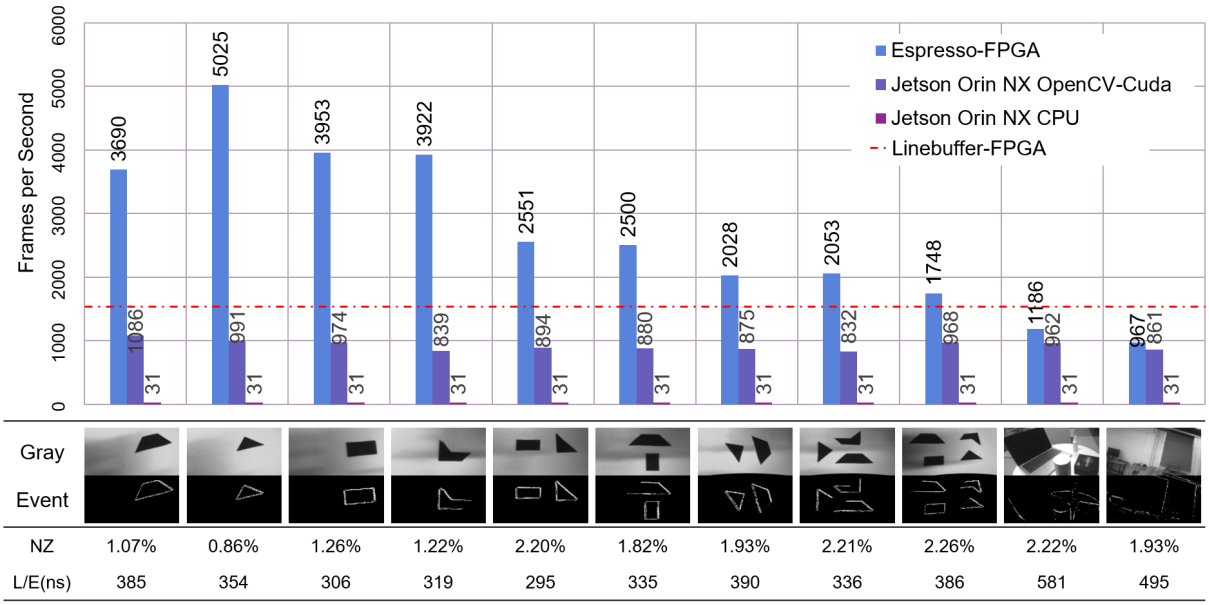


Fig. 7. Evaluation on different scenarios. (NZ: percentage of non-zero values, L/E: latency/event, resolution: 256×256)

C. Event-Scheduler Architecture

Event-Scheduler enables the cooperation of N-Pending-FIFOs and Shift-Hash Table with a finite state machine (FSM) with 4 states: update, compare, read, and write. Fig. 6A illustrates the overall architecture; Fig. 6B, C, and D show the execution behavior within Event-Scheduler in different states.

The update state (Fig. 6B) is the default one. Upon receiving the event ★, Event-Scheduler first converts it to the corresponding WinAddrs and pushes them into the N-Pending-FIFOs. The smallest WinAddr ♦ will be copied into separate registers to implement the pop mechanism of Fig. III-AB. Then, the FSM transfers to the compare state.

In the compare state (Fig. 6B), the minimum address of the heads of N-Pending-FIFOs is selected and compared with the copied WinAddr ♦ to judge whether to read the Shift-Hash Table or not. If the minimum address is smaller, it indicates that all events for the convolutional computation centered on this address are ready. Therefore, the FSM will go into the read state. Otherwise, it goes into the write state and writes the value of the event value to the shift-hash table.

In the read state (Fig. 6C), the WinAddr is first extended to its $N \times N$ surrounding addresses (illustrated in the upper right of the subfigure) which are then sent simultaneously to the Shift-Hash Table to read $N \times N$ values within one clock cycle. After these, revert back to the compare state.

In the write state (Fig. 6D), the current event's value and address are sent to the Shift-Hash Table, and after the writing is done in one or multiple clock cycles (which happens if the shifting is needed; shifting p times requires p extra cycles), the state turns back to the update state.

The Event-Scheduler design allows the number of layers in Espresso to be added. In other words, any vision algorithms with windows sliding operators can be realized on Espresso.

IV. Evaluation

We implement Espresso architecture on the FPGA platform and test the system performance of Espresso in real-world scenarios IV-B. Based on the results of real-world performance, we design and conduct three parameter sensitivity experiments IV-C to further analyze the behavior of Espresso.

A. Experiment Setup

To test the performance of Espresso architecture, we established Espresso-Harris, a Harris Corner Detection system based on Espresso architecture, which has a three-layer structure. The first layer is a 3×3 Sobel; the second layer is a 5×5 Gauss + Harris Response, the third layer is a 5×5 NMS (Non-Maximum Suppression). We synthesize and implement the Espresso-Harris on the Zynq UltraScale+ MPSoC ZCU104 device, with a global clock frequency of 100MHz. The baseline device is a Jetson Orin NX with an 8-core Arm Cortex-A78AE 64-bit CPU (2MB L2 + 4MB L3 and 16GB 128-bit LPDDR5 RAM) and a 1024-core NVIDIA Ampere architecture GPU with 32 Tensor Cores. Besides, we also build a line-buffer-based harris-corner-detection accelerator on the same FPGA device and compare its throughput and latency with Espresso-Harris. Real-world data have been collected for the throughput test, including both simple-shape scenarios and complex-environment scenarios. Besides, a series of parameter sensitivity experiments have been conducted on randomly generated data to further analyze the performance of Espresso-Harris under different conditions.

B. Systematic Performance of Espresso

Fig. 7 shows the latency performance of Espresso-Harris in simple-shape scenarios and complex-environment scenarios compared with CPU-Harris, GPU-Harris (OpenCV Cuda Lib)

and line-buffer-Harris. In simple-shape scenarios, Espresso-Harris's throughput is 2.4× higher than line-buffer-Harris, 3.8× higher than GPU-Harris and 116× higher than CPU-Harris. As the scenarios become more complex, the throughput of Espresso-Harris drops while others' remain the same level. In complex-environment scenarios, Espresso-Harris's throughput is 1.2× higher than GPU-Harris, but 1.3× lower than line-buffer-Harris. Additionally, the latency/event of Espresso-Harris increases as the scenarios become more complex. The results show that Espresso-Harris can achieve a higher throughput than line-buffer-Harris and GPU-Harris in simple-shape scenarios, but the advantage fades away as the scenarios become more complex.

We hypothesize there are two main attributes of complex-environment scenarios that challenge the Espresso design. First, a complex environment with more curves and edges will definitely cause higher NZ within a certain time period, which causes more computing times and longer latency/frame. Second, a complex environment with heavier noise will cause more separated single-dot events. Such a separated pattern of events increases the computing times at an incredible rate, strongly magnifying the latency/event and also causing longer latency/frame as a result. To confirm our hypothesis, a series of parameter sensitivity experiments have been conducted and the results are shown in Fig. 8 and 9.

C. Parameter Sensitivity Analysis

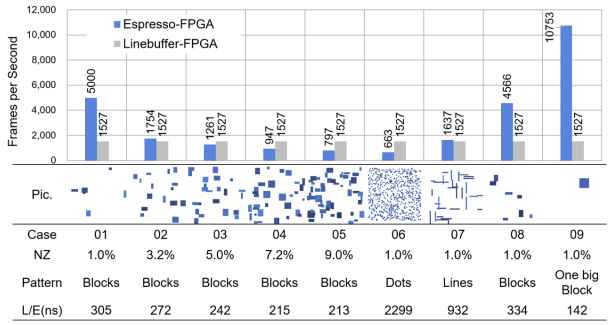


Fig. 8. Evaluation on input with different sparsity and different patterns.

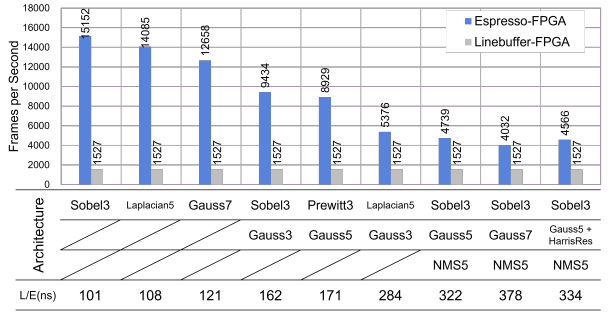


Fig. 9. Evaluation on different architectures. (Pattern: Blocks, NZ: 1%)

To further analyze the performance and behavior of Espresso architecture, three sensitivity experiments have been conducted. The input data are randomly generated by scripts, with different NZ levels and different aggregation levels. As case 01 to case 05 in Fig. 8 show, with the same level of aggregation (pattern=block), higher NZ actually results in lower throughput (from 9% NZ to 1% NZ, the throughput increases by 6.3×). However, inputs with lower NZ do not see a drop in latency/event; on the contrary, latency/event tends to increase as NZ drops. That, actually, is because at high NZ such as 9%, blocks overlap to form bigger blocks with higher aggregation levels, which pull down the latency/event dramatically. The conclusion that latency/event descends as aggregation level rises is also illustrated by case 06 to case 09 in Fig. 8, which demonstrate that the time Espresso-Harris takes to process inputs with the same NZ(1%) but different aggregation level can vary significantly. As the pattern changes from separated dots to one single block, the latency/event drops by more than 16.1×, and the throughput increases also by 16.2×. This result accounts for why the latency advantage of Espresso-Harris may weaken in complex real-world scenes with strong noise, which typically appears as separated dots.

In particular, we test the performance of different architectures of Espresso with different vision processing kernels, and the results are shown in Fig. 9. With the same input of 1%NZ and block pattern, as the layers of the architecture increase, the throughput drops accordingly with an almost constant proportion. That is because the NZ of input rises as the data are processed through layers. Since every layer has to wait for the completion of its following layer's calculation before sending out a new event, the bottleneck of the latency lies in the last layer with the input of the highest NZ. Also, kernels with larger size raises NZ faster than smaller ones (3×3 kernel up to 9× while 5×5 kernel up to 25×), thus resulting in longer latency unless they are at the ending layer in the architecture. The results above inspire us to minimize the number of layers as much as possible and avoid using large kernels at non-ending layers when designing Espresso accelerators so that Espresso architecture can show the greatest power in terms of throughput.

V. Conclusion

In this work, we design Espresso, a novel architecture for event-based vision processing. Espresso enables efficient stream processing of spatiotemporal ordering events. We design N-Pending-FIFOs and Shift-Hash Table to address the challenge of order mismatching and the latency overhead of discontinuous memory reading, respectively. Evaluations show that in real-world scenarios Espresso can achieve throughput up to 5000 fps, which is 5.1× higher than the embedded GPU.

Acknowledgment

This work was supported by the STI 2030—Major Projects (Grant No. 2021ZD0200300) and was financially supported by State Grid Corporation Headquarters Science and Technology Research Program (5700-202340289A-1-1-ZN). Taoyi Wang from Tsinghua University provided valuable suggestions on this work.

REFERENCES

[1] T. H. A. R. Vidal, H. Rebecq and D. Scaramuzza, "Ultimate slam? combining events, images, and imu for robust visual slam in hdr and high-speed scenarios," *IEEE Robotics and Automation Letters*, 2018.

[2] N. A. A. Zhu and K. Daniilidis, "Event-based visual inertial odometry," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, p. 1537–1557, Oct. 2015.

[4] I. Alzugaray and M. Chli, "Asynchronous corner detection and tracking for event cameras in real time," *IEEE Robotics and Automation Letters*, 2018.

[5] K. Z. J. T. Y. D. C. Wang, Y. Liu and P. Ren, "ac2slam: Fpga accelerated high-accuracy slam with heapsort and parallel keypoint extractor," *International Conference on Field-Programmable Technology (ICFPT)*, 2021.

[6] V. Dadu, J. Weng, S. Liu, and T. Nowatzki, "Towards general purpose acceleration by exploiting common data-dependence forms," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. ACM, Oct. 2019, p. 924–939. [Online]. Available: http://dx.doi.org/10.1145/3352460.3358276

[7] C. B. E. Mueggler and D. Scaramuzza, "Fast event-based corner detection," *Proceedings of the British Machine Vision Conference(BMVC)*, 2017.

[8] A. A. et al., "A low power, fully event-based gesture recognition system," *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[9] E. P. et al., "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, 2013.

[10] K. K. et al., "1.22m 35.6mpixel rgb hybrid event-based vision sensor with 4.88m-pitch event pixels and up to 10k event frame rate by adaptive control on event sparsity," *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023.

[11] M. D. et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, 2018.

[12] M. G. et al., "A 3-wafer-stacked hybrid 15mpixel cis + 1 mpixel evs with 4.6gevent/s readout, in-pixel tdc and on-chip isp and esp function," *IEEE International Solid-State Circuits Conference (ISSCC)*, 2023.

[13] G. Gallego, T. Delbruck, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis, and D. Scaramuzza, "Event-based vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 1, p. 154–180, Jan. 2022.

[14] Y. Gao, B. Zhang, Y. Ding, and H. K.-H. So, "A composable dynamic sparse dataflow architecture for efficient event-based vision processing on fpga," in *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '24. ACM, Apr. 2024, p. 246–257. [Online]. Available: http://dx.doi.org/10.1145/3626202.3637558

[15] G. G. H. Rebecq, T. Horstschaefer and D. Scaramuzza, "Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time," *IEEE Robotics and Automation Letters*, 2017.

[16] O. Hsu, M. Strange, R. Sharma, J. Won, K. Olukotun, J. S. Emer, M. A. Horowitz, and F. Kjølstad, "The sparse abstract machine," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, p. 710–726. [Online]. Available: https://doi.org/10.1145/3582016.3582051

[17] S. Klenk, M. Motzet, L. Koestler, and D. Cremers, "Deep event visual odometry," in *2024 International Conference on 3D Vision (3DV)*. IEEE, Mar. 2024, p. 739–749.

[18] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128×128 120 db 15 μs latency asynchronous temporal contrast vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, p. 566–576, 2008. [Online]. Available: http://dx.doi.org/10.1109/JSSC.2007.914337

[19] M. K. Vairalkar and S. Nimbhorkar, "Edge detection of images using sobel operator," *International Journal of Emerging Technology and Advanced Engineering*, vol. 2, no. 1, pp. 291–293, 2012.

[20] V. Vemulapati and D. Chen, "Fslam: an efficient and accurate slam accelerator on soc fpgas," *International Conference on Field-Programmable Technology (ICFPT)*, 2022.

[21] B. Y. W. Fang, Y. Zhang and S. Liu, "Fpga-based orb feature extraction for real-time visual slam," *International Conference on Field Programmable Technology (ICFPT)*, 2017.

[22] Z. Yang, T. Wang, Y. Lin, Y. Chen, H. Zeng, J. Pei, J. Wang, X. Liu, Y. Zhou, J. Zhang, X. Wang, X. Lv, R. Zhao, and L. Shi, "A vision chip with complementary pathways for open-world sensing," *Nature*, vol. 629, no. 8014, p. 1027–1033, May 2024.

CIM-BLAS: Computing-in-Memory Accelerator for BLAS

Rui Liu*†, Zerun Li*‡, Xiaoyu Zhang*‡, Xiaoming Chen*‡, Yinhe Han*‡, Minghua Tang†§
*Institute of Computing Technology, Chinese Academy of Science
†School of Materials Science and Engineering, Xiangtan University
‡University of Chinese Academy of Sciences
§National Center for Applied Mathematics in Hunan
Corresponding authors: chenxiaoming@ict.ac.cn, tangminghua@xtu.edu.cn

***Abstract*—Basic Linear Algebra Subprograms (BLAS) is a foundational software library for linear algebra kernels, which is widely used in scientific and engineering computing. Existing BLAS accelerations mainly rely on CPUs and GPUs. Many operations in BLAS are data intensive, so they are constrained by the limited memory bandwidth of CPUs and GPUs. The computing-in-memory (CIM) technology can effectively alleviate the memory wall bottleneck and is particularly suitable for accelerating BLAS. In this paper, we propose the first CIM accelerator for BLAS, CIM-BLAS, based on non-volatile memory. CIM-BLAS includes a unified floating-point pipeline to support high-precision arithmetics. High efficiency of the accelerator is achieved by developing configurable data flows to support various BLAS functions. Compared with GPU implementations, CIM-BLAS demonstrates several orders of magnitude performance and energy efficiency improvements for executing level-1 and level-2 BLAS functions, and can achieve an energy efficiency improvement of 2.6-24.1× for executing level-3 BLAS functions. The improvement increases with the size of the matrix, indicating excellent scalability of CIM-BLAS. Application-level evaluations also demonstrate the potential of CIM for accelerating BLAS.**

***Index Terms*—Computing-in-memory, Non-volatile memory, Basic Linear Algebra Subprograms**

I. Introduction

Linear algebra is the foundation of numerous scientific computing and engineering applications, covering core topics such as matrix operations, vector operations, solving systems of linear equations, eigenvalue problems, etc. Basic Linear Algebra Subprograms (BLAS) is a standardized library for linear algebra computations, which provides a series of common and fundamental linear algebra kernels. Over the past few decades, researchers have made significant efforts to accelerate BLAS on CPU and GPU platforms, continuously optimizing workloads and data communication at the software level [1]–[5]. However, in traditional computing architectures, frequent cross-layer data transfers cause severe transmission energy consumption issues, and the limited memory bandwidth significantly restricts the performance. Especially when handling large-scale matrix operations, the performance bottlenecks of traditional computing architectures become significant.

Computing-in-memory (CIM) breaks through the limitations of traditional architectures by allowing data to be processed in place within memory. This effectively mitigates the memory wall bottleneck by reducing the latency and energy overhead caused by data transfers. Recently, lots of CIM-based accelerators have been proposed, demonstrating better performance and energy efficiency compared with traditional ASIC accelerators. The target applications include neural network training (e.g., [6]–[8]), neural network inference (e.g., [9]–[11]), scientific computing (e.g., [12]–[14]), cryptography (e.g., [15]–[17]), etc. Since most BLAS functions are data intensive, CIM also has great potential, but currently we have not seen studies on CIM-based BLAS accelerators. Different from existing accelerators which typically accelerate a specific class of application, since BLAS provides an interface for common and fundamental linear algebra kernels, accelerating BLAS can benefit a wide range of applications. It faces two major challenges when accelerating BLAS by CIM.

Difficulty in supporting floating-point arithmetics: To enable linear algebra computations, BLAS requires the deployed hardware to support 32- and 64-bit floating-point arithmetics. However, the physical location of data in memory is fixed, and the analog data paths in memory cannot meet the exponent alignment requirements for floating-point arithmetics. Therefore, most existing CIM architectures are limited to performing fixed-point operations [18], [19].

Difficulty in supporting complex functions: BLAS requires more complex computations, but current CIM architectures only support a limited set of fixed operations [20]–[22]. Complex calculations such as matrix transposition, subtraction, and division often require other co-processors or additional peripheral modules, which inevitably cause significant communication or area overhead.

In this work, we for the first time attempt deploying the standard library BLAS for fundamental vector and matrix kernels onto CIM hardware, and propose a CIM-based BLAS accelerator, CIM-BLAS. The main contributions of this paper

This work was supported by National Key R&D Program of China (No. 2023YFF0719600), by National Natural Science Foundation of China (Nos. 62488101, 62495104, 62122076 and U23A20322), by Youth Innovation Promotion Association CAS, by Hunan Provincial Natural Science Foundation (Grant Nos. 2023JJ50009 and 2023JJ30599), by the Foundation of Innovation Center of Radiation Application (No. KFZC2023020701), and by Postgraduate Scientific Research Innovation Project of Xiangtan University (No. XDCX2023Y184).

are as follows.

- We study the design challenges of deploying BLAS on CIM-based hardware, and propose the first computation-in-memory BLAS accelerator based on non-volatile memory, which includes a unified floating-point pipeline to support high-precision arithmetics.
- We map complex arithmetic operations onto CIM arrays with conventional peripheral modules, and design a configurable data flow for different BLAS functions, enabling efficient BLAS acceleration without the need for additional hardware modules.
- A comprehensive evaluation of the CIM-BLAS architecture is conducted. Performance benchmarking indicates significant advantages over GPU implementations for executing level-1 and level-2 BLAS functions. For executing level-3 BLAS functions, the energy efficiency improves by 2.6-24.1× and increases with the size of the matrix. Application-level evaluations also demonstrate the potential of CIM for accelerating BLAS.

II. Motivation

BLAS is a typical data-intensive application, involving computations on matrices and vectors, such as matrix multiplication and vector addition. BLAS functions include three levels: level-1 for vector-vector operations [23], level-2 for matrix-vector operations [24], and level-3 for matrix-matrix operations [25], [26]. These operations require extensive memory access and data transfer, especially when handling large-scale matrices, making them well-suited for acceleration with CIM architectures. However, existing CIM architectures have certain limitations in supporting floating-point operations and complex linear algebra operations, such as matrix transposition, subtraction, and division.

This work aims to explore the potential of CIM architectures in supporting complex linear algebra operations. We observe that the primary difficulty in supporting floating-point operations with CIM is the exponent alignment problem, and it can be achieved using a delayed mantissa input method, which equivalently transforms the original floating-point operations into fixed-point operations within CIM arrays. We further design a new data flow and execution mechanisms to extend the potential of CIM arrays with peripheral modules, to support various BLAS functions. We hope to overcome the limitations of existing CIM architectures while enhancing the efficiency of BLAS computations.

III. CIM-BLAS Architecture Design

A. Architecture Overview and Data Layout

Fig. 1 shows the proposed CIM-BLAS accelerator based on the IEEE-754 floating-point format [27]. Processing engines (PEs)/computation units (CUs) are interconnected with an H-tree topology. In the H-tree topology, the nodes serve as computing units that can aggregate intermediate results from CUs, thereby allowing flexible configuration of the CUs to efficiently handle various BLAS functions.

A CU integrates multiple crossbar arrays which are dedicated to performing the basic operations of BLAS, which

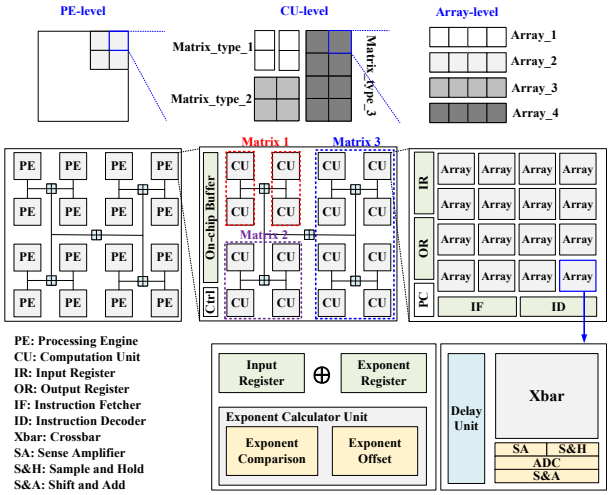


Fig. 1: CIM-BLAS accelerator overview.

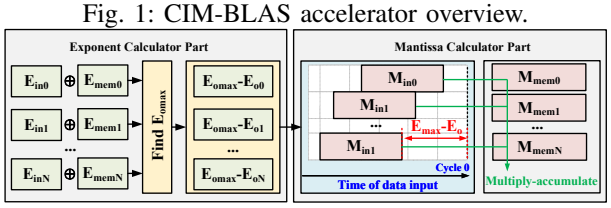


Fig. 2: In-memory floating-point operation pipeline.

can be implemented with popular non-volatile devices (e.g., RRAM, PCM, FeFET, MRAM). Each array contains a set of input registers and exponent registers, an exponent computation unit, a delay unit, and a crossbar equipped with a set of peripheral components. The floating-point data stored in the array is split into exponent and mantissa, which are stored in the exponent registers and the crossbar, respectively. It is worth noting that by adding a bias constant to the mantissas to represent negative numbers, the sign bit can be ignored [9]. The input registers are used to receive data to be processed and split it into exponent and mantissa. The exponent calculation unit, delay unit, and crossbar are used to implement the multiply-accumulate operations for floating-point numbers.

Fig. 1 also illustrates the data layout in the accelerator. A matrix is directly mapped onto different PEs on the chip. The sub-matrix in each PE is further partitioned into blocks and stored in different CUs. During computation, these blocks are combined through the nodes of the H-tree. In a CU, the data is split horizontally, with each array storing a row of the matrix. Different matrices can also be mapped within the same PE/CU to meet various matrix computation requirements.

B. In-Memory Floating-Point Pipeline

Non-volatile memory-based crossbars are widely used for accelerating neural networks. However, they are mostly designed for low-precision fixed-point computations which cannot meet the high-precision requirement of BLAS. Based on the floating-point storage in Section III-A, we propose a unified 5-stage pipeline mechanism for in-memory floating-point operations, which is shown in Fig. 2.

In the 1st stage, the exponent stored in the input register is added to the corresponding data in the exponent register. The exponent output results (E_o) of all rows are sent to the exponent comparison unit in the 2nd stage to find their

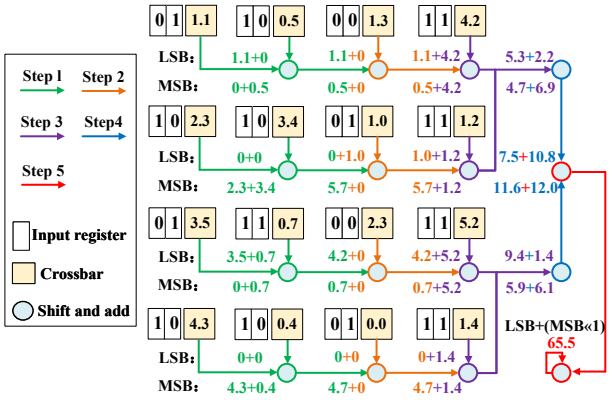


Fig. 3: Data flow of the Ripple pipeline.

maximum value (E_{omax}). In the 3rd stage, the exponent offset for each row is calculated by $E_{omax} - E_o$ with the exponent comparison units. In the 4th stage, according to the obtained exponent offsets, the corresponding input mantissas are shifted and some bits '0' are padded in the lower bits, thereby achieving exponent alignment. In other words, the delay units construct a new input, transforming the original floating-point operation into an equivalent fixed-point operation. In the 5th stage, the crossbar array receives the shifted mantissa data from the delay unit and performs in-memory integer multiply-accumulate operations. Finally, the floating-point results can be obtained by combining the exponent and mantissas parts. Notably, when only one multiplication operation is performed in the array, no alignment operation is needed, and only the 1st and 5th stages are executed. In this case, exponents and mantissas can be processed at the same time.

In the remaining part of this section, we will show in detail the data flow of each function of BLAS based on the CIM-BLAS architecture with the unified floating-point pipeline. Various BLAS functions can be efficiently accelerated on the unified hardware architecture by configuring the data path.

C. SCAL, DOT, ASUM, and NRM2 Functions

The SCAL, ASUM, and DOT functions are part of level-1 BLAS, which are used for vector-scalar multiplication, sum, and dot product operations, respectively.

For the SCAL function, as described in Section III-A, the matrix is mapped to the crossbar arrays in a row-wise manner. Note that a vector can be regarded as a matrix with N rows and 1 column, which means that each element of the vector is distributed across different arrays. Therefore, each array will perform a floating-point multiplication operation in parallel with the data sent to their input registers. These computation results will be collected by the CUs' output registers or the PEs' on-chip buffer for subsequent operations.

For the DOT function, the products of corresponding elements from two vectors need to be accumulated. However, traditional architectures rely on an addition tree to sum the output results from the array, which incurs additional area and latency overhead. To maximize the potential of the peripheral components and to avoid introducing additional hardware, we propose a pipeline structure to efficiently support dot-product operations, named Ripple. Fig. 3 illustrates the bit-serial execution flow of the Ripple pipeline. Here, we assume that each CU is configured with 4×4 arrays, with each array's input register storing a 4-bit mantissa. The green, orange, purple, blue, and red nodes are shift-and-add units of the arrays. By using the shifters for alignment operations, they can be extended to perform floating-point addition. We construct a systolic-like structure with those existing components. Each node will receive data from the array or the previous node and perform an accumulation operation. The data flows along the row direction, and the partial sum of the dot product operation can be obtained at the blue node. By accumulating these partial sums through the red node, the final result can be obtained.

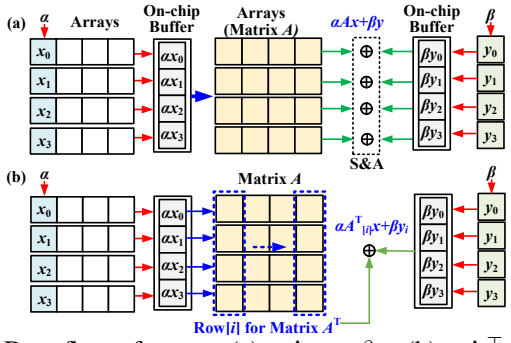


Fig. 4: Data flow of GEMV. (a) $\alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}$. (b) $\alpha\mathbf{A}^\top\mathbf{x} + \beta\mathbf{y}$.

ASUM and NRM2 can be considered as sub-operations of the DOT function. The ASUM function computes the sum of the elements in a vector. When performing an ASUM function, we only need to read out the corresponding data from each array and use the Ripple pipeline to execute the sum operation.

The basic formula of NRM2 function is $nrm2(\mathbf{x}) = \sqrt{\sum_{i=1}^{n} x_i^2}$, where $\mathbf{x}$ is a vector with elements x_i's, and n is the number of elements in the vector. To compute NRM2, each array first performs a read operation and feeds the read data back into its input register. Finally, the vector performs a DOT function with itself to obtain an intermediate result ($\sum_{i=1}^{n} x_i^2$), which is then sent to a co-processor or CPU for square root calculation. It is worth noting that the amount of data required to be transmitted to the co-processor or CPU for the NRM2 function is very small, which is only 64-bit data for a double-precision floating-point. Therefore, the communication overhead is negligible compared with the computation involved.

D. GEMV and GEMM Functions

GEMV is an important function in level-2 BLAS, which is used for matrix-vector multiplication. Specifically, GEMV performs the following operations

$$\mathbf{y} \leftarrow \alpha\mathbf{A}\mathbf{x} + \beta\mathbf{y}, \text{ or } \mathbf{y} \leftarrow \alpha\mathbf{A}^\top\mathbf{x} + \beta\mathbf{y} \quad (1)$$

where $\mathbf{A}$ is an $m \times n$ matrix, x is an n-dimensional vector, y is an m-dimensional vector, and α and β are scalars.

Fig. 4 illustrates the data flow of the GEMV function. In order to simplify the computation process, vectors $\mathbf{x}$ and $\mathbf{y}$ first perform SCAL functions to obtain $\alpha\mathbf{x}$ and $\beta\mathbf{y}$ at the same time. Then the vector $\alpha\mathbf{x}$ is loaded into the input registers of the CUs where matrix $\mathbf{A}$ resides and subsequently sent to the corresponding arrays to perform multiply-accumulate

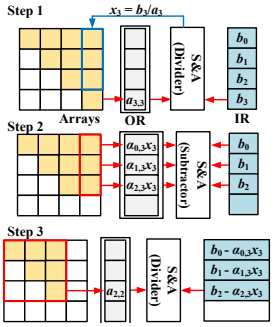


Fig. 5: Data flow of solving an upper triangular matrix.

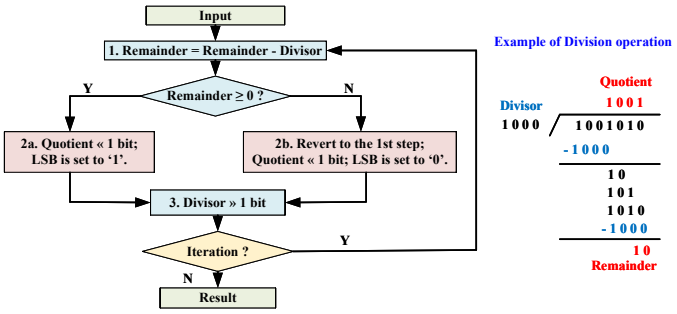


Fig. 6: Division operation implementation with shift-and-add module.

operations in parallel. The elements of the output column vector $\alpha\mathbf{Ax}$ are the output results of the arrays. Finally, $\beta\mathbf{y}$ is added to the output column vector $\alpha\mathbf{Ax}$ with shift-and-add modules of each array, obtaining the result of the GEMV function.

When matrix $\mathbf{A}$ needs an additional transpose operation, each array can be considered as storing a column vector of matrix $\mathbf{A}^\top$. Due to the physical storage location of the data, the elements of the same row vector of $\mathbf{A}^\top$ are dispersed across various arrays. This prevents us from relying on the arrays to perform matrix-vector multiplication operations in parallel to compute the output column vector. In this case, we can utilize the Ripple pipeline to perform the computation of $\alpha\mathbf{A}^\top\mathbf{x}$. We divide the matrix $\mathbf{A}^\top$ into N row vectors and perform DOT functions with vector $\alpha\mathbf{x}$ sequentially, which allows us to obtain the elements of the output column vector one by one. Finally, the elements of the output column vector will be added to the corresponding elements of vector $\beta\mathbf{y}$ to obtain the result of $\alpha\mathbf{A}^\top\mathbf{x} + \beta\mathbf{y}$.

In our accelerator, GEMM is implemented based on GEMV. We split the second input matrix into multiple column vectors and sequentially execute the GEMV function with the target matrix until all column vectors have been processed.

E. TRSV *Function*

The TRSV function is one of the most widely used kernel in BLAS, which solvers triangular matrix problems. Its operation can be defined by the following formula

$$\mathbf{Ax} = \mathbf{b} \tag{2}$$

where $\mathbf{A}$ is an $n \times n$ triangular matrix, $\mathbf{x}$ represents the unknown vector to be solved, and $\mathbf{b}$ is a constant vector.

We use forward/backward substitution to solve upper/lower triangular matrices. Fig. 5 shows an example of solving an upper triangular matrix. In the first step, the element $a_{3,3}$ is read out and a division operation is performed with by b_3. At this point, we obtain the solution x_3 for the first equation. In the second step, x_3 is fed back into the matrix $\mathbf{A}$, and SCAL functions are performed. The constant term vector b_i is subtracted by the result of $a_{i,3}x_3$ for the corresponding row, and the result is stored in the registers. The remaining elements that are not involved in the computation will form a new upper triangular matrix. We repeat the above steps until all unknown variables x_i are solved.

To avoid hardware overhead from additional peripheral components, we also use the shift-and-add module to support

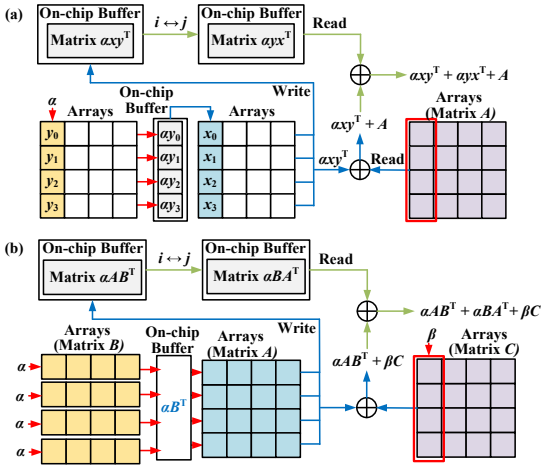


Fig. 7: (a) Data flow of GER, SYR, and SYR2 functions. (b) Data flow of SYR2k function.

matrix subtraction and division operations. To obtain $\mathbf{A} - \mathbf{B}$, we keep the minuend $\mathbf{A}$ unchanged and take the two's complement of the subtrahend $\mathbf{B}$. In this way, $\mathbf{A} - \mathbf{B}$ will be converted into an add operation of $\mathbf{A}$ and the two's complement of $\mathbf{B}$. The division operation implementation primarily rely on the subtraction and shift operations, as shown in Fig. 6. First, we calculate the difference between the remainder and the divisor, and check if the result is non-negative. If the value is negative, it reverts to the 1st step, left-shifts the quotient by 1 bit, and sets the new rightmost bit to '0'; otherwise, we left-shift the quotient by 1 bit, and set the new rightmost bit to '1'. The divisor register is right-shifted by 1 bit. The divisor is right-shifted by 1 bit and checked whether all the computational operations have been completed. If not, we return to the 1st step and continue executing the above process.

F. GER, SYR, SYR2, SYRK, *and* SYR2K *Functions*

GER, SYR, and SYR2 are functions widely executed in data analysis, optimization problems, and other scientific computing applications, which can effectively handle incremental updates to a matrix. Their specific operations are as follows

$$\mathbf{A} \leftarrow \alpha\mathbf{xy}^\top + \mathbf{A}, \quad \mathbf{A} \leftarrow \alpha\mathbf{xy}^\top + \alpha\mathbf{yx}^\top + \mathbf{A}. \tag{3}$$

Fig. 7a provides a detailed description of the data flow of GER, SYR, and SYR2 functions. Multiplying a column vector $\mathbf{x}$ by a row vector $\mathbf{y}^\top$ results in a product matrix $\mathbf{xy}^\top$. To enhance computational parallelism, this product matrix can be transformed into a combination of scalar multiplications

of the column vector elements. To perform a SYR function, vector $\mathbf{y}$ first performs a SCAL function with scalar α, and the result is then sequentially sent to input registers of the CUs where vector $\mathbf{x}$ resides. In this way, we can obtain the result of $\alpha\mathbf{x}\mathbf{y}^\top$ column by column. Each calculation result is sent to the corresponding column of matrix $\mathbf{A}$ for addition with shift-and-add modules. When all elements of vector $\alpha\mathbf{y}$ have been processed, we obtain the SYR function result.

For SYR2, as shown in Eq. (3), the SYR function is a sub-operation of the SYR2 function. Furthermore, $\alpha\mathbf{x}\mathbf{y}^\top$ is the transposes of $\alpha\mathbf{y}\mathbf{x}^\top$. To eliminate redundant computations, we store the results of $\alpha\mathbf{x}\mathbf{y}^\top$ in the buffer and swap their row and column indices. Finally, adding $\alpha\mathbf{y}\mathbf{x}^\top$ from the buffer to the corresponding elements of the SYR function matrix will yield the final result of the SYR2 function.

SYRK and SYR2K are described by the following formulas:

$$\mathbf{A} \leftarrow \alpha\mathbf{A}\mathbf{A}^\top + \beta\mathbf{C},\ \mathbf{A} \leftarrow \alpha\mathbf{A}\mathbf{B}^\top + \alpha\mathbf{B}\mathbf{A}^\top + \beta\mathbf{C}. \quad (4)$$

For the SYRK function, we first apply the SCAL function to all column vectors of matrices $\mathbf{A}$ and $\mathbf{C}$, obtaining $\alpha\mathbf{A}$ and $\alpha\mathbf{C}$, respectively. Afterward, the result of SYRK can be obtained by adding the result of performing GEMM with matrix $\mathbf{A}$ and $\alpha\mathbf{A}^\top$ to the corresponding elements of $\beta\mathbf{C}$. Fig. 7b shows the execution flow of SYR2K function. To obtain the result of the SYR2K function, we first compute the results for $\alpha\mathbf{B}^\top$ and $\beta\mathbf{C}$. Similar to the SYR2 function, we obtain the column vectors of $\alpha\mathbf{A}\mathbf{B}^\top$ sequentially using the GEMV function, and while adding them to the corresponding elements of $\beta\mathbf{C}$, $\alpha\mathbf{A}\mathbf{B}^\top$ is also written into the buffer. By swapping the indices in the buffer, we can easily obtain the result for matrix $\alpha\mathbf{B}\mathbf{A}^\top$. Finally, adding the elements of $\alpha\mathbf{B}\mathbf{A}^\top$ in the buffer to the corresponding elements of matrix storing the intermediate results yields the final result of the SYR2K function.

G. Summary

Based on the unified hardware architecture and floating-point execution pipeline, we have constructed a comprehensive data flow for each BLAS function, covering all BLAS functions except ROTG and AMAX, which require more specialized modules and designs. A Ripple pipeline structure is proposed to efficiently support the SCAL, DOT, ASUM, NRM2 functions, and matrix transpose operation. An in-memory floating-point computation mechanism is used to support the GEMV and GEMM functions. We further map the subtraction and division functions to the shift-and-add module, enabling in-memory TRSV and TRSM functions. Finally, the GER, SYR, SYR2, SYRK, and SYR2K functions are deployed on our accelerator with parallel computation flows.

IV. Evaluation

The CIM arrays and all analog components are simulated with HSPICE. We use the 45nm Predictive Technology Model (PTM) [28] as the base MOSFET model and select RRAMs with a resistance range of 5kΩ-5MΩ [29], [30] to construct the memory arrays. Synopsys Design Compiler is used for synthesizing the digital components. The size of the CIM arrays is 256×256, with 16 shared ADCs used in each array. For the ADCs and DACs used in the accelerator, we obtain their parameters from existing designs/products [31], [32]. Based on these fundamental parameters, we evaluate the performance and energy efficiency of various functions on CIM-BLAS and compare CIM-BLAS with the cuBLAS library that runs on an NVIDIA Tesla V100 GPU (fabricated with 12nm technology). It is worth emphasizing that currently there is no work on CIM- or ASIC-based accelerators for BLAS, so no such baseline can be compared.

A. Comparison with Level-1 and Level-2 cuBLAS

Fig. 8 shows a comparison of performance and energy efficiency between CIM-BLAS and GPU for the DDOT, DGEMV, DTRSV, DSYR, and DSYR2 functions.

For the DDOT and DGEMV functions, the GPU's execution time varies with the size of the matrix/vector. However, for CIM-BLAS, each array can perform matrix multiplication operations simultaneously, and the Ripple pipeline execution cycle of DOT only depends on the array size, so the time for aggregating data is very short. As a result, the execution time of our accelerator remains almost unchanged when performing DDOT and DGEMV functions. Compared with the GPU, CIM-BLAS performs the DDOT and DGEMV functions with speedups of 4290–26108× and 2503–247774×, respectively, while reducing energy consumption by 4101–5391× and 1829–2828×, respectively.

For the DTRSV function, CIM-BLAS demonstrates speedups of 219-3103× and energy savings of 5248-50423× over the GPU baseline. The main reason for the substantial energy savings is that the array relies solely on extremely low-power precharge amplifiers for row-wise calculations. Additionally, the significant communication energy consumption of the GPU further amplifies the disparity.

For the DSYR and DSYR2 functions, CIM-BLAS achieves energy savings of 5323-6471× and a maximum speedup of 12.4× . The main reason is the use of a column-wise computation mechanism, where the execution time is proportional to the number of elements in vector $\mathbf{y}$. This serial computation approach results in longer computation times.

B. Comparison with Level-3 cuBLAS

We compare the performance and energy consumption of executing matrix-matrix multiplication operations, as shown in Fig. 9. GEMM_T denotes the execution of a transposition operation on matrix $\mathbf{A}$, achieved through the Ripple pipeline.

In terms of energy consumption, as the array size increases, CIM-BLAS's energy for the DGEMM, DGEMM_T, DSYRK, and DSYR2K functions continually rises, eventually exceeding the energy consumption of the GPU. This is because CIM-BLAS drives the entire array to perform computations, and the energy consumption of in-memory floating-point multiply operations is inherently higher than that of GPU cores. Therefore, as the matrix size increases, the computation workload increases, the proportion of memory access energy consumption decreases, and the energy consumption gap between CIM-BLAS and GPU cores accumulates and gradually widens, eventually exceeding the GPU's memory communication energy consumption. Benefiting from the low-power precharge circuitry,

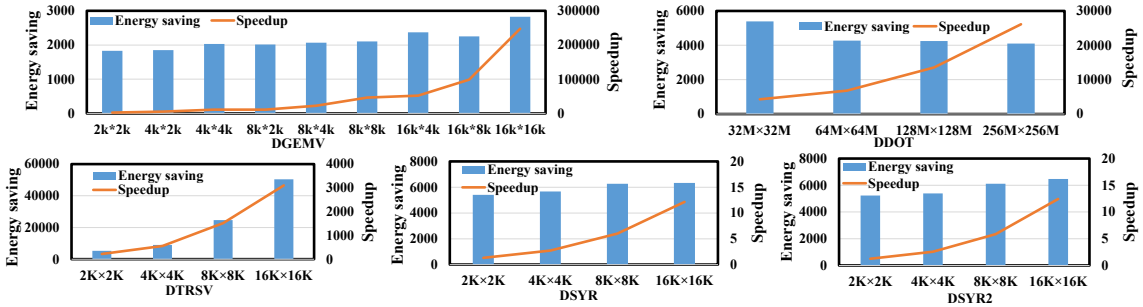


Fig. 8: Speedups and energy savings of CIM-BLAS against GPU for level-1 and level-2 cuBLAS.

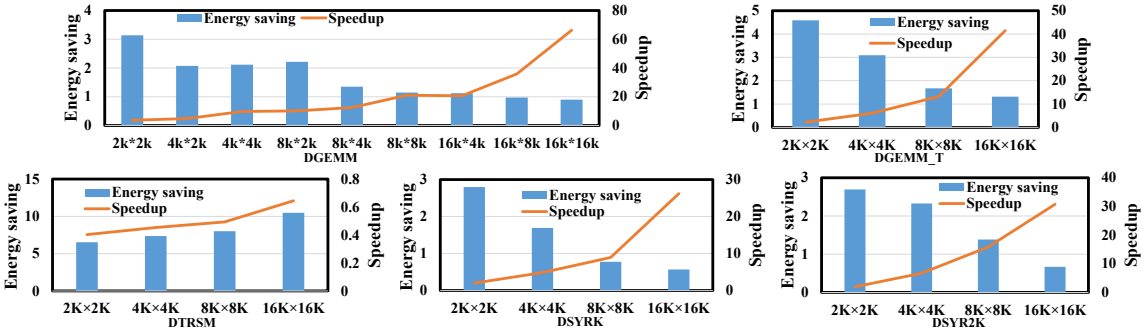


Fig. 9: Speedups and energy savings of CIM-BLAS against GPU for level-3 cuBLAS.

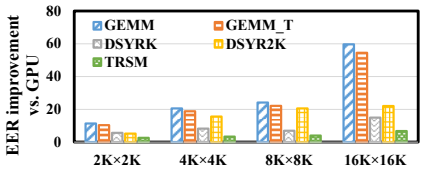


Fig. 10: Energy efficiency ratio (EER) improvements of CIM-BLAS against GPU.

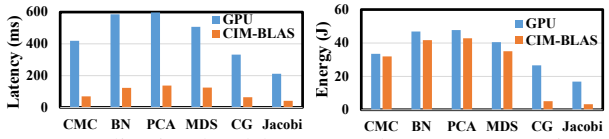


Fig. 11: Application-level latency and energy comparison for different applications.

CIM-BLAS's energy consumption in the DTRSM function is significantly lower than that of the GPU cores, demonstrating energy savings of 6.5-10.5× compared with the GPU baseline.

In terms of performance, for the DGEMM, DGEMM_T, DSYRK, and DSYR2K functions, CIM-BLAS exhibits 1.9-66.3× speedup compared with the GPU. The speedup is even higher for larger matrix sizes. The performance improvement primarily comes from the reduced inter-layer communication and high parallelism of CIM-BLAS. At the current scale, there is a performance gap between DTRSM and the GPU. The main reason is that we use forward/backward substitution to solve upper/lower triangular matrices, which results in longer solving times due to this serial computation approach. However, as the size scale increases, CIM-BLAS shows an upward trend in acceleration. It is expected that CIM-BLAS will surpass the GPU in efficiency for executing the DTRSM function with larger matrix sizes.

Fig. 10 shows the energy efficiency ratio (EER) improvements of CIM-BLAS against the GPU baseline. For various functions, CIM-BLAS's EER is 2.6–24.1× higher than that of the GPU. The EER advantage of CIM-BLAS increases with the matrix size, showing an upward trend as the size grows. This also confirms that the proposed CIM-BLAS can efficiently support BLAS functions.

C. Application-Level Evaluation

We use 6 applications related to scientific computing for the application-level evaluation: covariance matrix calculation (CMC), Bayesian network (BN), principal component analysis (PCA), multidimensional scaling (MDS), conjugate gradient (CG), and Jacobi iterative method. As shown in Fig. 11, CIM-BLAS reduces the execution latency by 4.88× on average. Due to the dominant matrix-matrix operations, CIM-BLAS have limited energy savings in CMC, BN, PCA, and MDS. CIM-BLAS's efficient matrix-vector operations reduce the energy consumption of CG and Jacobi by 5.15× on average.

V. Conclusion

BLAS is a fundamental software library for linear algebra kernels. It is typically implemented on CPUs or GPUs, whose performance is heavily restricted by the memory bandwidth on conventional von Neumann computers. Due to the in-situ computation capability, CIM is evidently a promising solution for enhancing BLAS performance. This paper presents the first CIM-based BLAS accelerator architecture, which includes a unified floating-point pipeline to support high-precision arithmetics. By developing the potential of CIM arrays and peripheral components, we further propose a configurable data flow for various BLAS functions, achieving high efficiency for BLAS executions. Evaluation results show that, compared with GPU implementations, our CIM accelerator has significant potential for accelerating BLAS.

REFERENCES

[1] L. Wang, W. Wu, Z. Xu, J. Xiao, and Y. Yang, "Blasx: A high performance level-3 blas library for heterogeneous multi-gpu computing," in *Proceedings of the 2016 International Conference on Supercomputing*, 2016, pp. 1–11.

[2] A. Abdelfattah, D. Keyes, and H. Ltaief, "Kblas: An optimized library for dense matrix-vector multiplication on gpu accelerators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 3, pp. 1–31, 2016.

[3] cublas: Basic linear algebra on nvidia gpus. [Online]. Available: https://developer.nvidia.com/cublas (Access Date: January 2023)

[4] Amd, amd core math library (acml). [Online]. Available: http://developer.amd.com/acml

[5] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, and I. Yamazaki, "Accelerating numerical dense linear algebra calculations with gpus," *Numerical computations with GPUs*, pp. 3–28, 2014.

[6] H. Jiang, X. Peng, S. Huang, and S. Yu, "Cimat: A transpose sram-based compute-in-memory architecture for deep neural network on-chip training," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 490–496.

[7] Z. Wang, C. Li, P. Lin, M. Rao, Y. Nie, W. Song, Q. Qiu, Y. Li, P. Yan, J. P. Strachan, N. Ge, N. McDonald, Q. Wu, M. Hu, H. Wu, R. S. Williams, Q. Xia, and J. J. Yang, "In situ training of feed-forward and recurrent convolutional memristor networks," *Nature Machine Intelligence*, vol. 1, no. 9, pp. 434–442, 2019.

[8] Y. Luo and S. Yu, "Accelerating deep neural network in-situ training with non-volatile and volatile memory based hybrid precision synapses," *IEEE Transactions on Computers*, vol. 69, no. 8, pp. 1113–1127, 2020.

[9] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016, pp. 14–26.

[10] T. Song, X. Chen, X. Zhang, and Y. Han, "Brahms: Beyond conventional rram-based neural network accelerators using hybrid analog memory system," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1033–1038.

[11] X. Zhang, R. Liu, T. Song, Y. Yang, Y. Han, and X. Chen, "Re-femat: A reconfigurable multifunctional fefet-based memory architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 5071–5084, 2022.

[12] X. Chen and Y. Han, "Solving least-squares fitting in $o(1)$ using rram-based computing-in-memory technique," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 339–344.

[13] X. Yin, Y. Qian, A. Vardar, M. Günther, F. Müller, N. Laleni, Z. Zhao, Z. Jiang, Z. Shi, Y. Shi, X. Gong, C. Zhuo, T. Kämpfe, and K. Ni, "Ferroelectric compute-in-memory annealer for combinatorial optimization problems," *Nature Communications*, vol. 15, no. 1, p. 2419, 2024.

[14] S. Wang, Z. Sun, Y. Liu, S. Bao, Y. Cai, D. Ielmini, and R. Huang, "Optimization schemes for in-memory linear regression circuit with memristor arrays," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 12, pp. 4900–4909, 2021.

[15] M. Xie, S. Li, A. O. Glova, J. Hu, and Y. Xie, "Securing emerging nonvolatile main memory with fast and energy-efficient aes in-memory implementation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 11, pp. 2443–2455, 2018.

[16] M. Ma, Y. Zhu, Z. Zhu, R. Yuan, J. Liu, L. Xu, Y. Yang, and Y. Wang, "Efficient in-memory aes encryption implementation using a general memristive logic: Surmounting the data movement bottleneck," *IEEE Nanotechnology Magazine*, vol. 16, no. 2, pp. 24–C3, 2022.

[17] R. Liu, X. Zhang, Z. Xie, X. Wang, Z. Li, X. Chen, Y. Han, and M. Tang, "Fecrypto: instruction set architecture for cryptographic algorithms based on fefet-based in-memory computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 9, pp. 2889–2902, 2023.

[18] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[19] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H.-S. P. Wong, and G. Cauwenberghs, "A compute-in-memory chip based on resistive random-access memory," *Nature*, vol. 608, no. 7923, pp. 504–512, 2022.

[20] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, "X-sram: Enabling in-memory boolean computations in cmos static random access memories," *TCAS-I*, vol. 65, no. 12, pp. 4219–4232, 2018.

[21] X. Chen, Y. Wu, and Y. Han, "Fepim: Contention-free in-memory computing based on ferroelectric field-effect transistors," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 114–119.

[22] R. Liu, X. Zhang, X. Chen, Y. Han, and M. Tang, "Femic: Multi-operands in-memory computing based on fefets," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 678–683.

[23] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh, "Basic linear algebra subprograms for fortran usage," *ACM Transactions on Mathematical Software (TOMS)*, vol. 5, no. 3, pp. 308–323, 1979.

[24] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, "An extended set of fortran basic linear algebra subprograms," *ACM Trans. Math. Softw.*, vol. 14, no. 1, p. 1–17, 1988.

[25] J. J. Dongarra, J. D. Cruz, S. Hammarling, and I. S. Duff, "Algorithm 679: A set of level 3 basic linear algebra subprograms: model implementation and test programs," vol. 16, no. 1, p. 18–28, 1990.

[26] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, "A set of level 3 basic linear algebra subprograms," vol. 16, no. 1, p. 1–17, 1990.

[27] W. Kahan, "Ieee standard 754 for binary floating-point arithmetic," *Lecture Notes on the Status of IEEE*, vol. 754, no. 94720-1776, p. 11, 1996.

[28] W. Zhao and Y. Cao, "New generation of predictive technology model for sub-45 nm early design exploration," *IEEE Transactions on electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[29] H.-L. Chang, H.-C. Li, C. Liu, F. Chen, and M.-J. Tsai, "Physical mechanism of hfo 2-based bipolar resistive random access memory," in *Proceedings of 2011 International Symposium on VLSI Technology, Systems and Applications*. IEEE, 2011, pp. 1–2.

[30] S.-S. Sheu, P.-C. Chiang, W.-P. Lin, H.-Y. Lee, P.-S. Chen, Y.-S. Chen, T.-Y. Wu, F. T. Chen, K.-L. Su, M.-J. Kao, K.-H. Cheng, and M.-J. Tsai, "A 5ns fast write multi-level non-volatile 1 k bits rram memory with advance write scheme," in *2009 Symposium on VLSI Circuits*. IEEE, 2009, pp. 82–83.

[31] L. Kull, T. Toifl, M. Schmatz, P. A. Francese, C. Menolfi, M. Braendli, M. Kossel, T. Morf, T. M. Andersen, and Y. Leblebici, "A 3.1 mw 8b 1.2 gs/s single-channel asynchronous sar adc with alternate comparators for enhanced speed in 32 nm digital soi cmos," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 12, pp. 3049–3058, 2013.

[32] T.-H. Yang, H.-Y. Cheng, C.-L. Yang, I.-C. Tseng, H.-W. Hu, H.-S. Chang, and H.-P. Li, "Sparse reram engine: Joint exploration of activation and weight sparsity in compressed neural networks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 236–249.

H3Match: A Hybrid Heterogeneous Hypergraph Matching Method for Subcircuit Identification

Bohao Li[1], Qingsong Peng[1], Changhong Wang[2], Tianming Ni[3], Tinghuan Chen[4], Qi Sun[1,*], Cheng Zhuo[1,*]
[1]Zhejiang University,
[2]Shandong Yunhai Guochuang Cloud Computing Equipment Industry Innovation Co. Ltd.,
[3]Anhui Polytechnic University, [4]The Chinese University of Hong Kong, Shenzhen
[*]Corresponding authors, email: {qisunchn, czhuo}@zju.edu.cn

***Abstract*—Subcircuit matching is widely applied in logic synthesis, design verification, hardware security, *etc*. Previous works employ redundant circuit representations, coupled with time-consuming enumerative search methods. Subsequent works use a hybrid "approximate filtering – exact verification" framework, but the numerous false negatives predicted by the graph neural network (GNN) based filtering lead to severe matching failure. In this paper, an improved hybrid method named H3Match is proposed to achieve a better tradeoff between runtime, accuracy, and false negative rate. First, we model the circuits as hypergraphs to fully capture the topology and construct diverse heterogeneous hyperedge features to facilitate the learning of circuit topologies. Second, to reduce the false negatives, we reformulate the subgraph matching problem as matching directed acyclic graphs (DAGs) with embedded circular structure information and develop a directed GNN-based *approximate* matching approach to identify potential matching subcircuits. Finally, we propose a general mixed integer nonlinear programming (MINLP) formulation for *exact* verification, with convergency speed accelerated by extracting the initial solution from the results in *approximate* matching. Experimental results show that our *approximate* method outperforms state-of-the-art (SOTA) methods by 4.31% in accuracy while achieving virtually zero false negatives. Our exact verification is on average 4.16× faster than SOTA *exact* methods. Overall, the end-to-end flow achieves a 7.08× speedup compared to existing approaches.**

I. Introduction

Subcircuit matching involves identifying subcircuits in a target circuit that have the same device attributes and connectivity as the query circuit. It is a fundamental step in the EDA flow, with wide applications in layout generation, design verification, reverse engineering, *etc*, [1]–[3]. The current matching methodology primarily comprises two steps: representing the circuit in a graph-based form and performing subgraph isomorphism on the resulting graphs. Due to the NP-complete nature of subgraph isomorphism, the matching time increases significantly as integrated circuits continue to scale up. Therefore, developing efficient matching algorithms is imperative to accelerate the circuit design flow.

Previous studies mainly focus on developing subgraph isomorphism algorithms, with insufficient emphasis on circuit modeling methods. Typically, these studies represent circuits as graphs [4], [5], bipartite graphs [6]–[8], or heterogeneous multigraphs [2], [9]. Since edges in these graph structures can only represent pairwise relationships, the one-to-many connections between nets and devices are decomposed into multiple one-to-one connections. Consequently, additional edges (as in graphs or multigraphs) or nodes (as in bipartite graphs) are required to maintain the original structure, as shown in Fig. 1(a). Therefore, these modeling methods face two major issues. First, these approaches introduce redundancy in representation, which increases the runtime of subcircuit matching. Second, due to the decomposition process, the rich semantic features inherent in the connections between nets and devices are lost, limiting subsequent matching steps from effectively leveraging this information to enhance efficiency.

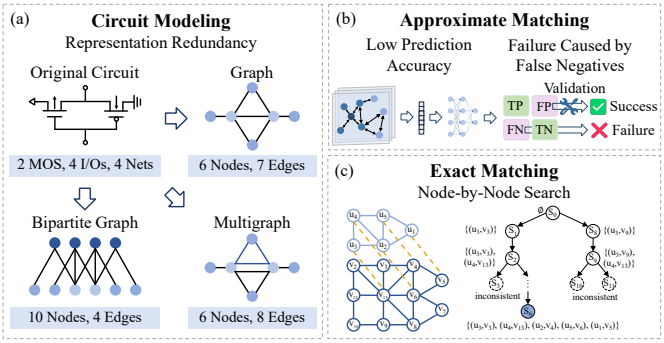


Fig. 1 Major challenges faced by existing hybrid matching methods. (a) More nodes or edges are required for circuit modeling. (b) Approximate matching suffers from low accuracy and numerous false negatives that can lead to matching failure. (c) Node-by-node search is time-consuming.

For the matching algorithms, prior works primarily employ search-based methods for exact subcircuit matching. Some arts assign discriminative labels to nodes based on device attributes or neighborhood structures, and establish mappings between nodes via breath first search or depth first search [10], [11]. Other works adopt tree-search methods originated from subgraph isomorphism research [12]–[14]. However, when applied to large-scale circuits, developing discriminative labeling algorithms is highly challenging, and the exhaustive tree-search become time-consuming.

With advancements in deep learning, GNNs have emerged as a promising solution [15]–[18]. By extracting graph topology into embedding vectors and using multilayer perceptron

(MLP) for prediction, GNN-based methods eliminate the need for node-wise matching and achieve a speedup of over 10 $\times$ compared to conventional methods. However, these methods are unable to provide the one-to-one correspondences between the node set of two graphs and suffer from low accuracy due to information loss in message-passing. Therefore, recent works adopt a hybrid method, where GNN is first used to filter out candidate matches, followed by verification using search-based methods [19], [20]. This approach achieves a tradeoff between runtime and accuracy, making it a promising solution. Nonetheless, two critical challenges persist: first, the search-based verification dominates the overall runtime, posing a bottleneck for further acceleration. Second, in most applications, the number of positive examples is significantly smaller than that of negative examples. Therefore, the search-based method only verifies the positive results from approximate matching. As a result, only false positives (FPs) can be corrected, leaving false negatives (FNs) unaddressed, which leads to the failure of the entire matching flow. Therefore, minimizing the number of false negative results in approximate matching is essential, while previous studies overlook this issue.

In this paper, to address the aforementioned challenges, we propose a heterogeneous hypergraph representation for circuit modeling and a hybrid subgraph isomorphism method for efficient subcircuit matching. Our contributions are as follows:

- We introduce an effective heterogeneous hypergraph representation for circuit modeling to reduce representation redundancy and improve accuracy.
- To minimize the false negatives, we reformulate the subgraph matching problem as sub-DAG matching and develop a directed HGNN-based approximate matching method to filter out potential matching subcircuits.
- To enhance the matching speed, we formulate the exact verification as a mixed integer nonlinear programming problem and accelerate the optimization by providing an initial solution extracted from the filtered results.

Experimental results demonstrate that our approximate filtering achieves 98.96% accuracy and nearly 100% recall, while our exact verification is 4.16$\times$ faster than existing approaches. Consequently, our end-to-end flow attains 100% accuracy and a 7.08$\times$ speedup in the subcircuit matching task.

II. Preliminaries

A. Problem Formulation

Definition 1 (Heterogeneous Hypergraph). A heterogeneous hypergraph can be represented as $\mathcal{HHG} = (\mathcal{V}, \mathcal{E}, \mathcal{T}_v, \mathcal{T}_e, m_v, m_e)$, where $\mathcal{V}$, $\mathcal{E}$, $\mathcal{T}_v$, $\mathcal{T}_e$ is the set of nodes, hyperedges, node types and hyperedge types, respectively. m_v and m_e are mapping functions that associate each node in $\mathcal{V}$ or hyperedge in $\mathcal{E}$ with one type in $\mathcal{T}_v$ or $\mathcal{T}_e$. Each hyperedge $e \in \mathcal{E}$ stands for the connection between a group of nodes $\{v_1, v_2, ..., v_n\}$ where $v_1, v_2, ..., v_n \in \mathcal{V}$.

Definition 2 (Subgraph Isomorphism). Given a target graph $\mathcal{HHG}_1 = (\mathcal{V}, \mathcal{E}, \mathcal{T}_v, \mathcal{T}_e, m_v, m_e)$ and a query graph $\mathcal{HHG}_2 = \left(\mathcal{V}^{'}, \mathcal{E}^{'}, \mathcal{T}_v^{'}, \mathcal{T}_e^{'}, m_v^{'}, m_e^{'}\right)$, subgraph isomorphism is to find all injective functions $f : \mathcal{V}^{'} \rightarrow \mathcal{V}$ such that $\forall u \in \mathcal{V}^{'} : m_v^{'}(u) = m_v(f(u))$; $\forall \{u_1, u_2, ..., u_n\} \in \mathcal{V}^{'} : (u_1, u_2, ..., u_n) \in \mathcal{E}^{'} \implies (f(u_1), f(u_2), ..., f(u_n)) \in \mathcal{E}, m_e^{'}((u_1, u_2, ..., u_n)) = m_e(f(u_1), f(u_2), ..., f(u_n))$.

Problem 1 (Subcircuit Identification). Given a target circuit and a query circuit, subcircuit identification is to find all subcircuits in the target circuit that have the same topology and device attributes as the query circuit.

B. Hypergraph Neural Network

Graphs serve as an effective model for capturing pairwise relationships between two entities. However, in real-world scenarios, it is often necessary to represent more complex relationships among multiple entities. Therefore, hypergraphs and hypergraph neural networks have been introduced to model and extract features from such relationships. When extending GNNs to hypergraphs, the message-passing framework maintains, while the change from the adjacency matrix to the incidence matrix requires normalization using the node degree matrix $\mathbf{D}_v$ and the hyperedge degree matrix $\mathbf{D}_e$ simultaneously [21] , as illustrated in the Equation (1):

$$\mathbf{X}^{(l+1)} = \sigma(\mathbf{D}_v^{-1}\mathbf{H}\mathbf{W}\mathbf{D}_e^{-1}\mathbf{H}^{\top}\mathbf{X}^{(l)}), \tag{1}$$

where $\mathbf{X}^{(l)}$, $\mathbf{H}$, $\mathbf{W}^{(l)}$ represent the node embedding matrix, the incidence matrix, and the weight matrix at the l_{th} layer, respectively. This process can be regarded as first aggregating the features of nodes connected to each hyperedge onto the hyperedge itself, and then aggregating the features of multiple hyperedges onto the central node.

C. Subgraph Isomorphism

Subgraph isomorphism, also known as subgraph matching, is a fundamental problem in graph data analysis. The mainstream research is centered on search-based algorithms [22]–[25], which guarantee to find all matches. VF2 [23] is one of the most commonly used exact algorithms. It determines node matching using subgraph constraints and heuristic rules, then explores other nodes by depth-first search and backtracking. Subsequent studies optimize heuristic rules to enhance the efficiency of search space pruning [24], [26]. However, due to the enormous search space and the exhaustive search strategy, matching on large-scale graphs remains time-consuming.

Another approach represents the node mapping between two graphs as an assignment matrix $\mathbf{S}$, where $\mathbf{S}_{i,j}$ represents a match between the i_{th} node of the target graph and the j_{th} node of the query graph, thus reformulating subgraph isomorphism as an integer programming problem, as shown in Equation (2):

$$\begin{aligned} \min \quad & \|\mathbf{S}\mathbf{A}'\mathbf{S} - \mathbf{A}\|_F^2, \\ \text{s.t.} \quad & \sum_i \mathbf{S}_{i,j} = 1, \sum_j \mathbf{S}_{i,j} \leq 1, \mathbf{S}_{i,j} \in \{0,1\}, \end{aligned} \tag{2}$$

where $\mathbf{A}'$ and $\mathbf{A}$ denote the adjacency matrices of the target graph and query graph, respectively. Being a nonconvex

MINLP, this optimization problem poses challenges for various approximation techniques, which may fail to yield an exact solution or get stuck in local minima.

The advancements of GNNs for effectively learning low-dimensional representations of graphs facilitate a variety of downstream tasks, including subgraph isomorphism [27], [28]. By leveraging GNNs to extract global graph features and the efficiency of GPU computation, GNN-based methods achieve significantly higher speeds than other approaches, but face serious challenges in accuracy and generalization.

III. H3MATCH

A. Overall Flow

To achieve better overall performance in terms of runtime, FNs, and accuracy, we develop a novel subcircuit matching method. Our method takes two circuits as input: a target circuit and a query circuit. The objective is to identify all subcircuits in the target circuit that match the query circuit.

The overall flow of our framework is outlined as follows: we first employ a netlist parser to extract the topology and device attributes from the input netlists, followed by constructing the heterogeneous hypergraph representation. Subsequently, the heterogeneous hypergraph is converted into a DAG with circular structure information embedded. We use a directed HGNN to embed the DAG and identify potentially isomorphic subcircuits by partial order comparison. Thereafter, we employ an exact subgraph isomorphism approach on the identified small-scale subcircuits to verify the matching results. The overall flow is shown in Fig. 2. Henceforth, subgraphs of the target graph will be referred to as sub-targets.

B. Circuit Modeling

In circuit modeling, we aim to represent the circuit with fewer nodes and edges while fully preserving its topology. Many existing works in the literature model the transistor as four separate nodes, *i.e.*, source, drain, gate, body, and each assigned a unique node feature. Moreover, prior works either decompose nets into multiple pairwise connections between devices or introduce additional nodes to represent the one-to-many relationship between nets and devices.

In our representation, NMOS, PMOS, V_{ss}, V_{dd}, and other devices are each modeled as one node and assigned a 5-dimensional one-hot vector to encode their types. Nets are modeled as heterogeneous hyperedges, with each hyperedge assigned a 9-dimensional feature vector to represent the connections to different device pins. Each element of the vector corresponds to the count of connections to the gate, drain, and source of NMOS and PMOS, as well as the connections to V_{ss}, V_{dd}, and other devices.

Our heterogeneous hypergraph representation takes two advantages. First, fewer nodes and edges are required to represent the circuit, which reduces the structural complexity (compared to graphs) and the number of GNN layers required for learning (compared to bipartite graphs), thereby enhancing the accuracy for approximate matching while shortening the runtime for exact matching. Second, the diverse hyperedge features derived from heterogeneous hyperedges are fully preserved, providing significant benefits in differentiating subcircuits with similar topologies.

C. Approximate Matching

In approximate matching, we focus on two key aspects. On the one hand, the false negatives should be as low as possible to reduce the overall matching errors within the framework. On the other hand, false positives should be kept at a low level to minimize the computational overhead of subsequent exact matching. Additionally, the matching process needs to be optimized to achieve a matching speed far exceeding that of exact matching. To achieve these objectives simultaneously, we propose the following matching method.

1) Subgraph Matching Simplification: We first simplify the subgraph matching problem to a sub-DAG matching for two main reasons. First, the presence of numerous circular structures in transistor-level circuits may cause information confusion during message passing. Second, with circular structures removed, sub-DAG matching becomes a necessary condition for subgraph matching, *i.e.*, sub-DAG matching relaxes the criteria of subgraph isomorphism, which in turn helps reduce false negatives. Therefore, we simplify the heterogeneous hypergraph into a DAG with heterogeneous hyperedges, as shown in Fig. 2.

To minimize the depth of the DAG and thereby reduce the GNN layers required for embedding, we initiate the DAG transformation by determining the root node of the query graph. The eccentricity of each node is calculated as the maximum of the shortest path distances to all other nodes. The node with the smallest eccentricity is selected as the root node. We then add the n-th hop neighbor nodes of the root node to the n-th layer of the DAG. Hyperedges connect nodes between two adjacent layers, with the node in the n-th layer acting as the head node and the nodes in the (n+1)-th layer as the tail nodes. We ensure that each hyperedge connects only one head node. If multiple head nodes are present, the hyperedge is decomposed into several hyperedges, each sharing identical tail nodes but distinct head nodes. Furthermore, removing circular structures may result in a decrease in accuracy. To mitigate this issue, we introduce a new child node to the node with the minimum hop count on the circle, assigning it the average feature value of all nodes within the circle.

2) Graph Learning Model: Previous heterogeneous GNNs leverage attention mechanisms to capture various relationships among entities. However, our topology extraction task focuses on distinguishing different hyperedge types. To this end, we assign a learnable parameter to each type of hyperedge for computational simplicity. Consequently, the incidence matrix can be formulated as Equation (3):

$$\mathbf{H} = \sum_{r \in \mathcal{T}_e} w_r \mathbf{H}_r. \tag{3}$$

To learn the low-dimensional representation of the heterogeneous DAG, we extend the HGNN model from Equation (1)

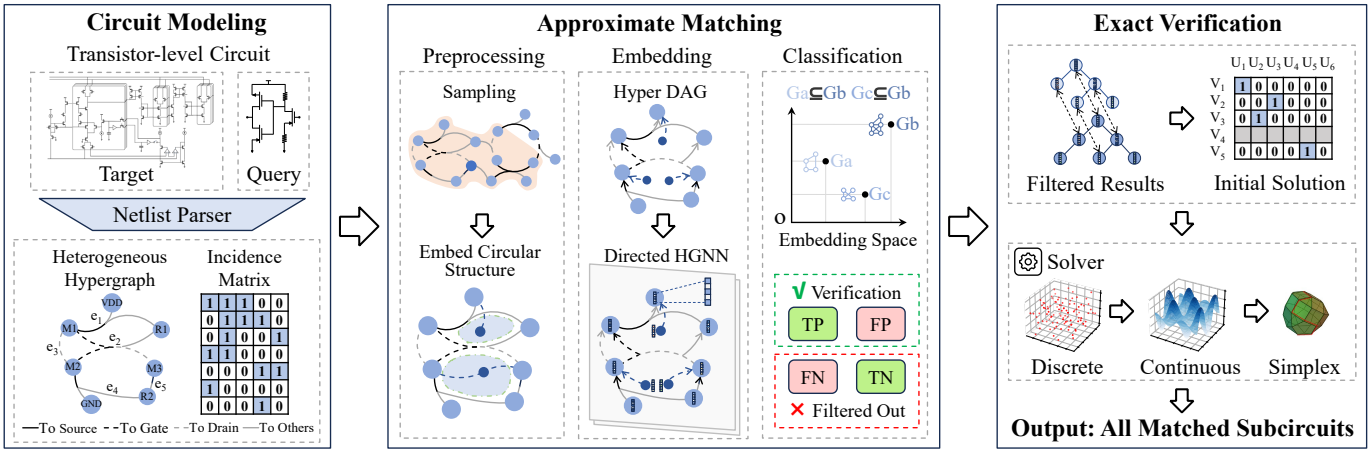


Fig. 2 Overview of the proposed method.

to directed hypergraphs. The directed HGNN maintains a two-stage message passing process. In the first stage, messages are passed from tail nodes to hyperedges, while in the second stage, messages are passed from hyperedges to head nodes. By employing the directed HGNN model, the heterogeneous topological information is aggregated at the root node of the DAG. The embedding of the root node is then used to predict subgraph relationships.

Inspired by NeuroMatch [27], we introduce partial order comparison to predict subgraph relationships based on the embeddings. Given that both subgraph relationships and dominance relationships between graph embeddings exhibit partial order properties, we aim for the GNN model to preserve this partial order relationship during graph embedding. This enables subgraph isomorphism to be simplified to comparing the magnitudes of embedding vectors, resulting in a tenfold speed improvement over traditional MLP-based predictions. To achieve the above objective, a max-margin loss is utilized, as shown in Equation (4):

$$L(\mathbf{z}_q, \mathbf{z}_t) = \alpha \sum_{(\mathbf{z}_q, \mathbf{z}_t) \in \mathcal{P}} ||\max\{0, \mathbf{z}_q - \mathbf{z}_t\}||_2^2 + \sum_{(\mathbf{z}_q, \mathbf{z}_t) \in \mathcal{N}} \max\{0, \beta - ||\max\{0, \mathbf{z}_q - \mathbf{z}_t\}||_2^2\}. \tag{4}$$

For positive examples, we expect the sub-target embedding $\mathbf{z}_t$ to dominate the query embedding $\mathbf{z}_q$, *i.e.*, $\mathbf{z}_q$ should be positioned in the lower-left of $\mathbf{z}_t$ in the embedding space. For negative examples, we require $\mathbf{z}_q$ and $\mathbf{z}_t$ to violate the dominance relationship and maintain a distance of at least β in the embedding space. Given that positive examples are scarce in our subcircuit identification task and the cost of misclassifying positive examples is significant, we assign a higher priority to the loss of positive examples by introducing a weight α in the loss function.

3) Sub-DAG Matching: Benefiting from our heterogeneous DAG representation and graph learning model, we utilize the following three techniques to further improve performance.

First, following the root node selection method described before, we determine the root node of the query graph. We then sample neighborhoods from the target graph, centered at each node, with the same radius as the query graph. In comparison, prior research uses a sampling radius twice that of the query graph to ensure the sub-target covers the query. As a result, our method only uses a $2k$-layer GNN to embed a k-hop query graph, while other approaches twice the number of layers, which not only increases matching time but also risks over-smoothing.

Second, to reduce the number of matching executions, we propose two filtering criteria aimed at excluding non-isomorphic sub-targets. First, only two nodes with identical features are allowed to check the subgraph relationship. Second, after converting the neighborhoods of the two nodes into DAGs, the number of nodes at each layer of the DAG is counted. Subsequent matching is performed only if each layer of the sub-target DAG contains at least as many nodes as the corresponding layer in the query DAG.

Third, to fully exploit the diverse hyperedge features, we aim to conduct message-passing based on these features. Therefore, we apply a duality transformation to the hypergraph, *i.e.*, swapping the roles of nodes and hyperedges. In the transformation, the topology is fully preserved, and the hyperedge features become the node features in the dual hypergraph.

To summarize, in the sub-DAG matching flow, we first identify the root node of the query graph and sample the neighborhood of each node in the target graph with the same radius as the query graph. Next, we filter out non-isomorphic sub-targets based on the two criteria. Subsequently, we perform dual transformation and DAG conversion to the query and sub-target using the aforementioned method. Finally, we extract topological features using our directed HGNN model and make predictions through partial order comparison.

D. Exact Matching

To improve the matching accuracy, we employ an exact verification method to validate the examples classified as

matched in approximate matching. The conventional subgraph matching formulation is extended to accommodate the matching of heterogeneous hypergraphs, as shown in Equation (5):

$$\begin{aligned} \min \quad & \left\|\mathbf{P}^{(v)}\mathbf{H}'\mathbf{P}^{(e)} - \mathbf{H}\right\|_F^2 + \left\|\mathbf{P}^{(v)}\mathbf{F}' - \mathbf{F}\right\|_F^2 \\ & + \left\|(\mathbf{P}^{(e)})^\top\mathbf{K}' - \mathbf{K}\right\|_F^2 \\ \text{s.t.} \quad & \mathbf{P}^{(e)} = \{0,1\}^{|E'|\times|E|}, \mathbf{P}^{(v)} = \{0,1\}^{|V|\times|V'|}, \\ & \sum_j p_{i,j}^{(e)} \le 1, \sum_i p_{i,j}^{(e)} = 1, \sum_j p_{i,j}^{(v)} = 1, \sum_i p_{i,j}^{(v)} \le 1, \end{aligned} \tag{5}$$

where $\mathbf{H}'$ and $\mathbf{H}$ represent the incidence matrix of the target graph and the query graph. $\mathbf{F}$, $\mathbf{K}$ represent the node feature matrix and hyperedge feature matrix. The assignment matrices $\mathbf{P}^{(v)}$ and $\mathbf{P}^{(e)}$ are used to permute and select rows and columns of $\mathbf{H}'$, which corresponds to reordering the nodes and hyperedges of the sub-target graph, followed by selecting a subset to match with the query graph. The objective function consists of three terms that represent the matching of the input graphs' topology, node features, and hyperedge features, respectively. If $\mathbf{P}^{(v)}$ and $\mathbf{P}^{(e)}$ exist such that the objective function equals zero, it indicates that the two input graphs are subgraph isomorphic. Given that the formulation is a nonconvex MINLP problem with numerous variables and constraints, we adopt the following heuristics and settings to accelerate the solution.

First, to reduce the number of variables, we prune all hyperedges in the sub-target whose features do not exist in the query graph and further remove the isolated nodes resulting from pruning. Notably, while this approach still holds for graphs and bipartite graphs, the rich hyperedge features enable our method to prune significantly more hyperedges.

Second, to accelerate the convergency, we extract prior knowledge from approximate matching to provide an initial solution for the solver. The approximate matching outputs a pair of potential matching DAGs, where the two root nodes are assumed to be matched. Additionally, we select nodes with unique features from each layer in each DAG and match nodes with identical features within the same layer between both DAGs. Therefore, the corresponding rows in $\mathbf{P}^{(v)}$ for these matched nodes are set to 0 or 1. Notably, the approximate matching may provide incorrect node mappings for positive examples (*e.g.*, the root nodes do not match) or produce node mappings for negative examples. Our method still guarantees an exact solution without an obvious change in matching time.

Third, in the sampling stage of approximate matching, the radius of the sub-target is set equal to that of the query, which ensures that there is at most one match between them. Owing to this, we execute a subgraph isomorphism check, *i.e.*, terminating once a minimum value is found, rather than determining all minimum values of the nonconvex problem. Based on the above heuristics, the formulation can be effectively solved using relaxation and the simplex method.

TABLE I Statistics of the Testset.

	# Transistor	# Net		# Transistor	# Net
Case1	26520	29178	Case5	60775	51655
Case2	45187	41553	Case6	104892	98716
Case3	47561	47787	Case7	119927	112271
Case4	60110	64670	Case8	190562	180316

TABLE II Performance of our approximate matching method against **SOTA** methods on the validation set.

Query	NeuroMatch		MLCAD'24		D2Match		Ours	
Size	Acc.	# FN	Acc.	# FN	Acc.	# FN	Acc.	# FN
25	88.50%	155	93.66%	17	94.90%	365	**99.31%**	**0**
50	87.37%	129	91.90%	49	94.43%	424	**99.14%**	**0**
75	81.93%	276	91.08%	55	94.85%	435	**98.86%**	**0**
100	80.46%	328	89.80%	53	94.40%	610	**98.51%**	**0**

TABLE III Runtime comparison between our methods and **SOTA** methods. Runtime: seconds.

Query	Approximate Methods				Exact Methods	
Size	NeuroMatch	MLCAD'24	D2Match	Ours	VF3	Ours
25	0.009	0.017	0.047	**0.009**	0.086	**0.012**
50	0.013	0.022	0.069	**0.012**	0.794	**0.145**
75	0.015	0.025	0.076	**0.015**	1.941	**0.784**
100	**0.018**	0.034	0.160	0.020	3.661	**2.418**

* The VF3 algorithm refers to the exact matching step in MLCAD'24, rather than the VF3 baseline.

IV. Experiment

A. Experimental Setup and Dataset

We conduct subcircuit identification on eight in-house transistor-level circuits, as summarized in TABLE I. We first extract the topology and device types using the netlist parser, followed by constructing the graph and hypergraph representations with the NetworkX [29] and HyperNetX [30] library. We implement our method on a Linux server with Intel Xeon 8475B CPUs and NVIDIA RTX 4090 GPUs.

In approximate matching, we train the models on a randomly generated dataset consisting of 60,000 examples. Validation is performed on 20,000 examples obtained by extracting the neighborhoods of randomly selected nodes from the 8 test cases. To improve the accuracy of positive examples, the ratio of positive to negative samples is set to 1:1 for both the training and validation sets. The weight parameter α and margin β in the loss function are 2 and 0.1, respectively. For all graph learning methods, the batch size is set to 64.

In the exact matching stage, our method and MLCAD'24 execute a subgraph isomorphism check on each pair of the query graph and the sub-target graph. All exact matching experiments are repeated 100 times, and the final results are determined after discarding the top 5% and bottom 5% of values and then calculating the average.

In the comparison of end-to-end methods, the VF3 baseline performs a complete subgraph matching to identify all isomorphic subgraphs within the entire target graph. The runtime of hybrid methods comprises four parts: circuit modeling, neighborhood extraction, GNN inference, and exact matching, whereas the runtime of the search-based method is composed of circuit modeling and exact matching.

TABLE IV The performance comparisons of our method against other end-to-end approaches. Runtime: seconds.

Case	Query Size	VF3		MLCAD'24		Ours		Case	Query Size	VF3		MLCAD'24		Ours	
		Acc.	Runtime	Acc.	Runtime	Acc.	Runtime			Acc.	Runtime	Acc.	Runtime	Acc.	Runtime
1	25	100.00%	37.25	99.71%	27.82	100.00%	**3.04**	5	25	100.00%	61.45	99.87%	63.67	100.00%	**6.90**
	50	100.00%	66.69	99.49%	147.79	100.00%	**6.57**		50	100.00%	155.47	99.54%	338.61	100.00%	**14.89**
	75	100.00%	101.57	99.43%	363.87	100.00%	**23.75**		75	100.00%	236.36	99.55%	833.78	100.00%	**54.20**
	100	100.00%	154.34	99.57%	718.56	100.00%	**75.55**		100	100.00%	327.23	99.61%	1646.62	100.00%	**172.84**
2	25	100.00%	55.54	99.74%	47.26	100.00%	**5.05**	6	25	100.00%	125.26	99.72%	109.98	100.00%	**12.01**
	50	100.00%	111.32	99.56%	251.68	100.00%	**10.92**		50	100.00%	278.20	99.69%	584.50	100.00%	**25.90**
	75	100.00%	179.95	99.46%	619.85	100.00%	**40.06**		75	100.00%	499.85	99.41%	1439.13	100.00%	**93.83**
	100	100.00%	269.45	99.59%	1224.20	100.00%	**128.19**		100	100.00%	582.50	99.48%	2842.01	100.00%	**298.68**
3	25	100.00%	59.13	99.80%	49.72	100.00%	**5.29**	7	25	100.00%	150.22	99.85%	125.55	100.00%	**13.53**
	50	100.00%	111.38	99.50%	264.88	100.00%	**11.44**		50	100.00%	344.27	99.58%	668.08	100.00%	**29.21**
	75	100.00%	196.26	99.51%	652.39	100.00%	**42.09**		75	100.00%	478.65	99.48%	1645.21	100.00%	**106.68**
	100	100.00%	241.84	99.49%	1288.49	100.00%	**134.82**		100	100.00%	663.30	99.50%	3249.17	100.00%	**340.69**
4	25	100.00%	73.65	99.77%	63.07	100.00%	**6.93**	8	25	100.00%	258.15	99.76%	199.60	100.00%	**21.60**
	50	100.00%	162.45	99.50%	335.00	100.00%	**14.93**		50	100.00%	547.18	99.59%	1061.67	100.00%	**46.62**
	75	100.00%	241.88	99.46%	824.76	100.00%	**53.91**		75	100.00%	911.18	99.61%	2614.32	100.00%	**169.83**
	100	100.00%	335.59	99.41%	1628.70	100.00%	**171.35**		100	100.00%	1249.65	99.53%	5162.99	100.00%	**541.77**

B. Baselines

We select the following SOTA methods from each of the three subgraph matching methods, *i.e.*, search-based, GNN-based, and hybrid, as our baselines:

- VF3 [24]: Search-based method with tree-search and backtracking framework. The SOTA method for sparse graph matching [31]. Our experiments are performed using the Python interface of VF3 [32].
- NeuroMatch [27]: GNN-based method with partial order learning.
- D2Match [28]: GNN-based method with subtree matching degeneracy.
- MLCAD'24 [19]: Hybrid approach with HGNN-based filtering and VF3-based verification.

As previous methods are only capable of handling graph structures, we convert the netlists into graph representations via clique expansion. For the graph learning baselines, we adopt the hyperparameters provided in the original studies to ensure consistency.

C. Comparisons against SOTA Methods

We evaluate the performance of our graph learning method against SOTA methods on the validation set, with the accuracy and number of FN listed in TABLE II. Our method maintains an accuracy of over 98.5% across different sizes of query graphs, outperforming other baselines by an average of at least 4.31%. On the validation set, our method maintains zero false negatives, demonstrating the effectiveness of simplifying subgraph matching to sub-DAG matching.

TABLE III provides a runtime comparison between our methods and SOTA baselines, focusing on the average time of matching a positive and a negative example. Due to the two-stage message-passing process, HGNN-based methods, *e.g.*, MLCAD'24, require longer matching time compared to GNN-based methods NeuroMatch. However, in our method, the efficient subgraph sampling strategy reduces the size of the sub-target, which in turn minimizes the required HGNN layers and shortens the matching time. Consequently, our method slightly outperforms NeuroMatch on small-scale graphs and remains competitive on query graphs with 100 nodes. In terms of exact matching, we conduct the isomorphism check using our method and VF3. Our method consistently exceeds VF3 for query graphs of all sizes, achieving an average acceleration of 4.16×. In our method, by extracting the prior knowledge from approximate matching, up to 80% of the nodes in the query graph can establish an initial solution, thereby improving the convergence speed by at least twofold.

Due to the limited accuracy of GNN-based methods, which restricts their application in circuit identification, we compare our approach against existing search-based and hybrid methods, as illustrated in TABLE IV. For the eight test cases, our method demonstrates significant improvements in runtime compared to existing methods. Specifically, for query graphs of 25, 50, 75, and 100 nodes, our method is on average 10.93×, 10.71×, 4.68×, 2.00× faster than VF3, and 9.24×, 22.76×, 15.39×, 9.53× faster than MLCAD'24. Meanwhile, our method exhibits zero false negatives on the test cases, resulting in an accuracy of 100%.

V. Conclusion

In this work, we present H3Match, a novel subgraph isomorphism approach for subcircuit identification. We propose a heterogeneous hypergraph representation for transistor-level circuits, simplify the subgraph matching to sub-DAG matching, and design a hybrid approach with HGNN-based filtering and MINLP-based verification. Compared to prior works, H3Match significantly outperforms the SOTA methods in terms of runtime, accuracy, and FNs.

Acknowledgment

This project was supported by Zhejiang Provincial NSF (Grant No. D24F040002), NSFC (Grant No. W2412034), and Scientific Research Fund of Zhejiang Provincial Education Department (Grant No. Y202457026).

REFERENCES

[1] P.-H. Wu, M. P.-H. Lin, T.-C. Chen, C.-F. Yeh, X. Li, and T.-Y. Ho, "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 199–212, 2014.

[2] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. K. Sharma, W. Xu, S. M. Burns, J. Hu, R. Harjani, and S. S. Sapatnekar, "Gnn-based hierarchical annotation for analog circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[3] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2435–2448, 2021.

[4] X. Gao, C. Deng, M. Liu, Z. Zhang, D. Z. Pan, and Y. Lin, "Layout symmetry annotation for analog circuits with graph neural networks," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021.

[5] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "S 3 det: Detecting system symmetry constraints for analog circuits with graph similarity," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[6] H.-Y. Su, C.-H. Hsu, and Y.-L. Li, "Subhunter: A high-performance and scalable sub-circuit recognition method with prüfer-encoding," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[7] D. Cheremisinov and L. Cheremisinova, "Subcircuit pattern recognition in transistor level circuits," *Pattern Recognition and Image Analysis*, vol. 30, pp. 160–169, 2020.

[8] N. Rubanov, "Subislands: The probabilistic match assignment algorithm for subcircuit recognition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 22, no. 1, pp. 26–38, 2003.

[9] T. Chen, Q. Sun, C. Zhan, C. Liu, H. Yu, and B. Yu, "Analog ic aging-induced degradation estimation via heterogeneous graph convolutional networks," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021.

[10] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather, "Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm," in *Proceedings of the 30th International Design Automation Conference*, 1993.

[11] W. Kim and H. Shin, "Hierarchical lvs based on hierarchy rebuilding," in *Proceedings of 1998 Asia and South Pacific Design Automation Conference*, 1998.

[12] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. Sharma, W. Xu, S. M. Burns, J. Hu, R. Harjani, and S. S. Sapatnekar, "Gana: Graph convolutional network based automated netlist annotation for analog circuits," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020.

[13] R. S. Rajarathnam, Y. Lin, Y. Jin, and D. Z. Pan, "Regds: A reverse engineering framework from gdsii to gate-level netlist," in *2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, 2020.

[14] Z. Wu, I. Song, and I. Savidis, "Hybrid utilization of subgraph isomorphism and relational graph convolutional networks for analog functional grouping annotation," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023.

[15] Q. Xu, H. Geng, S. Chen, B. Yuan, C. Zhuo, Y. Kang, and X. Wen, "Goodfloorplan: Graph convolutional network and reinforcement learning-based floorplanning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 10, pp. 3492–3502, 2021.

[16] X. Dong, Y. Chen, X. Yin, and C. Zhuo, "Worst-case dynamic power distribution network noise prediction using convolutional neural network," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[17] Y. Chen, H. Pei, X. Dong, Z. Jin, and C. Zhuo, "Application of deep learning in back-end simulation: Challenges and opportunities," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2022.

[18] X. Dong, Y. Chen, J. Chen, Y. Wang, J. Li, T. Ni, Z. Shi, X. Yin, and C. Zhuo, "Worst-case power integrity prediction using convolutional neural network," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 4, pp. 1–19, 2023.

[19] B. Li, S. Wang, T. Chen, Q. Sun, and C. Zhuo, "Efficient subgraph matching framework for fast subcircuit identification," in *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024.

[20] A. Teliti, "Graph neural networks for topology recognition of ams integrated circuits," Ph.D. dissertation, Politecnico di Torino, 2023.

[21] S. Bai, F. Zhang, and P. H. Torr, "Hypergraph convolution and hypergraph attention," *Pattern Recognition*, vol. 110, p. 107637, 2021.

[22] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[23] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[24] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 804–818, 2017.

[25] B. Bhattarai, H. Liu, and H. H. Huang, "Ceci: Compact embedding cluster index for scalable subgraph matching," in *Proceedings of the 2019 International Conference on Management of Data*, 2019.

[26] A. Jüttner and P. Madarasi, "Vf2++—an improved subgraph isomorphism algorithm," *Discrete Applied Mathematics*, vol. 242, pp. 69–81, 2018.

[27] Z. Lou, J. You, C. Wen, A. Canedo, J. Leskovec *et al.*, "Neural subgraph matching," *arXiv preprint arXiv:2007.03092*, 2020.

[28] X. Liu, L. Zhang, J. Sun, Y. Yang, and H. Yang, "D2match: leveraging deep learning and degeneracy for subgraph matching," in *International Conference on Machine Learning*, 2023.

[29] "NetworkX," https://networkx.org/.

[30] "HyperNetX," https://github.com/pnnl/HyperNetX.

[31] L. Zeng, Y. Jiang, W. Lu, and L. Zou, "Deep analysis on subgraph isomorphism," *arXiv preprint arXiv:2012.06802*, 2020.

[32] "VF3Py," https://pypi.org/project/vf3py/.

Easz: An Agile Transformer-based Image Compression Framework for Resource-constrained IoTs

Yu Mao*, Jingzong Li†, Jun Wang‡, Hong Xu§, Tei-Wei Kuo¶, Nan Guan‡, Chun Jason Xue*
*Mohamed bin Zayed University of Artificial Intelligence, UAE, {yu.mao, jason.xue}@mbzuai.ac.ae
†The Hang Seng University of Hong Kong, Hong Kong, jingzongli@hsu.edu.hk
‡City University of Hong Kong, Hong Kong, {jwang699-c, nanguan}@my.cityu.edu.hk
§The Chinese University of Hong Kong, Hong Kong, hongxu@cuhk.edu.hk
¶National Taiwan University, Taiwan, ktw@csie.ntu.edu.tw

***Abstract*—Neural image compression, necessary in various machine-to-machine communication scenarios, suffers from its heavy encode-decode structures and inflexibility in switching between different compression levels. Consequently, it raises significant challenges in applying the neural image compression to edge devices that are developed for powerful servers with high computational and storage capacities. We take a step to solve the challenges by proposing a new transformer-based edge-compute-free image coding framework called Easz. Easz shifts the computational overhead to the server, and hence avoids the heavy encoding and model switching overhead on the edge. Easz utilizes a patch-erase algorithm to selectively remove image contents using a conditional uniform-based sampler. The erased pixels are reconstructed on the receiver side through a transformer-based framework. To further reduce the computational overhead on the receiver, we then introduce a lightweight transformer-based reconstruction structure to reduce the reconstruction load on the receiver side. Extensive evaluations conducted on a real-world testbed demonstrate multiple advantages of Easz over existing compression approaches, in terms of adaptability to different compression levels, computational efficiency, and image reconstruction quality.**

***Index Terms*—Image Compression, Erase-and-Squeeze, Transformer-based Auto-Encoder.**

I. Introduction

The need for advanced lossy image compression is raised by the explosive development of edge devices equipped with high-resolution cameras, such as industrial-inspections system [8], wildlife observation system [5], and autonomous driving [31], by analyzing massive data sensed by heterogeneously connected devices. Transmitting the huge volume of data generated by IoT devices can cause significant channel congestion, particularly in scenarios with limited bandwidth availability [11]. Neural-network (NN) based compressor can provide a better compression performance and outperform traditional image compression techniques like JPEG [10] and BPG [3]. However, due to its heavy, symmetric encoding and decoding structure and inflexible compression rate adjustment, current NN-based methods have not yielded practical use on resource-constrained edge devices.

Current edge NN-compression/decompression latency is unsatisfied due to the paucity of computational ability and storage on edge devices [25, 1]. As shown in Fig. 1, encoding an small 512×768 image can take as long as 18015 ms on high-end devices equipped with GPU like the NVIDIA Jetson TX2, not to mention current camera usually capture images to 4K size. Switching compression level can also cause $286 \sim 11600$ ms overhead, far exceeding the transmission time $151 \sim 163$ ms.

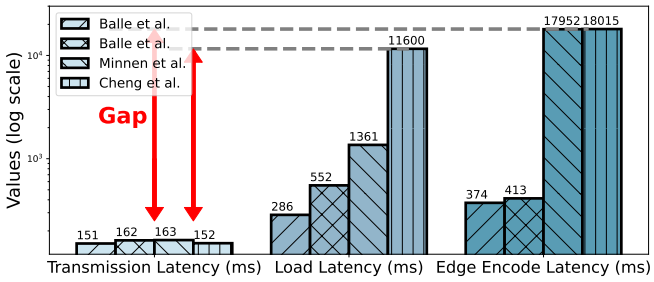


Fig. 1: NN-based compressors face challenges on edge devices like the Jetson TX2, where loading and encoding an image can take over 10 seconds compared to a transmission latency of about 0.1 seconds.

In this paper, we introduce Easz, an agile image coding framework that operates efficiently at both the edge and server side. To achieve agile encoding, Easz includes an erase-and-squeeze process, which removes the image patches based on a conditional random-based sampler. This technique can provide fine-grained compression level choices with fast adaptation to various scenarios. To improve the reconstruction quality, we introduce the novel transformer architecture to conduct fine-grained pixel-level reconstruction. However, the pixel-level reconstruction using naive transformer architecture is computationally intensive ($O(65536^2 \times 768)$ for a 256×256 image), even for a powerful 2080 Nvidia GPU server. We then propose a lightweight transformer architecture for efficient, high-quality reconstruction of erased patches on the receiver side. This involves a two-stage image patchify process to limit the scope of attention correlation calculations and a four-layer light weight transformer model for pixel-level local image

reconstruction.

The key contributions of this work can be summarized as:

- Generalized Erase-and-Squeeze Process: A new paradigm is introduced that offers more refined and flexible image reduction ratios;
- Receiver-Side Lightweight Transformer Architecture: A super-lightweight (8.7MB) transformer architecture is designed for efficient and high-quality reconstruction of erased patches;
- Compatibility with Existing Algorithms: Easz is compatible with all existing image compression algorithms and can also function independently;
- We conduct extensive experiments that verify the effectiveness of Easz.

II. Related Work

Image compression is experiencing significant growth [28, 29, 19, 6, 16]. Notable developments include the introduction of auto-regressive components [19], Gaussian Mixture Models for probability estimation [6], and general-purpose lossless compressiors using lightweight neural networks [16, 15, 18].

Despite progress, real-world applications still face challenges at the edge [14, 21, 11]. [21] leverages in-sensor compression and selective ROI prioritization to optimize high-resolution image processing for edge ML. Deep-learning-based compression methods take about 1$\sim$20 second per image (512×768) on NVIDIA Jetson TX2, and many real-life endpoints are less potent than the TX2 (considering Raspberry Pi 4) but still need to compress images. A primary issue is that most NN-based image compressors require a model switch when changing compression levels. [11] uses semantic-driven compressive sensing to enhance image compression for resource-constrained IoT systems. Another approach involves downsampling images at the edge and using super-resolution techniques to reconstruct them on the server [30]. These methods reduce computational load at the edge, but applying super-resolution directly in this context results in an inflexible downsizing rate and can degrade reconstruction performance [12].

III. System Design

The paper presents Easz, a novel edge-optimized image compression framework. It merges erase-based compression on the edge with a lightweight transformer-powered reconstruction on the server side, outperforming conventional codecs like JPEG and neural-network-driven compressors.

A. Erase-and-Squeeze Algorithm

The section outlines the proposed *Erase-and-Squeeze* strategy, including erase mask generation and the organization of un-erased image components on the edge/sender side.

Erase mask generation. An erasing mask is a binary matrix used to determine if subpatches in an image patch should be kept or erased, defined as $\mathbf{M}$ where each entry is set as follows:

$$\mathbf{M}[i,j] = 1 \text{ if } (i,j) \text{ is sampled, else } 0$$

Fig. 2 illustrates some mask examples with white blocks representing erased subpatches. As shown in Fig. 2(b), a simple approach to erase is to remove blocks along the diagonal, which allows for easy reorganization of the remaining sections. However, the diagonal mask is not generalized due to its fixed erase ratio. Super-resolution-based approaches also suffer from a fixed file size reduction ratio due to uniform down-sampling.

To achieve flexible size reduction, a naive solution is to randomly erase a portion of patches. However, this can cause significant distortion due to consecutive information loss. Fig. 2(a) shows a randomly generated mask that create large continuous erased areas, causing performance degradation in subsequent JPEG and reconstruction stages (See Fig. 3).

We then propose a generalized paradigm, a row-based conditional sampler for generating erase masks, covering various types including diagonal and uniform masks. This approach allows for a highly flexible sampling rate, while only need one model to reconstruct at any sampling rate. Also, it has better reconstruction behavior compared with a random masks.

Row-based Sampler Definition. In this sampler, each row i of the image is processed sequentially, and within each row, the column coordinate j is selected using a sampler. Thus, the Row-based Sampler is defined by two functions, g_r^1, which governs the random column selection from $\mathbf{X}$, while $g_r^0(i)$ represents the current row:

$$\hat{\mathbf{X}}[i,j] = \mathbf{X}[g_r^0(i), g_r^1(i,j)]$$

where $g_r^0(i)$ is a deterministic function representing row selection, and $g_r^1(i,j)$ is a random mapping for the column coordinate within row i.

Constraints for Row-based Sampler. When sampling from a matrix $X \in \mathbb{R}^{H \times W}$, where each row is sampled T times, the new sample $x_{i,t+1}$ is subject to:

1. Intra-row constraint (avoid proximity to previous samples in the same row):

$$\begin{aligned} &g_r^1(i,t+1) \sim \text{Uniform}(0, W-1) \\ &\text{s.t.} \quad \left| g_r^1(i,t+1) - g_r^1(i,t) \right| > \delta \end{aligned} \tag{1}$$

Here, δ is a threshold distance that ensures the newly sampled column $g_r^1(i,t+1)$ is sufficiently distant from the previously selected columns $\{g_r^1(i,0), \ldots, g_r^1(i,t)\}$.

2. Inter-row constraint (minimize adjacency to prior samples from the preceding row):

$$\begin{aligned} &g_r^1(i,t+1) \sim \text{Uniform}(0, W-1) \\ &\text{s.t.} \quad \left| g_r^1(i,t+1) - g_r^1(i-1,T) \right| > \Delta \end{aligned}$$

Similarly, Δ represents a minimum separation between the newly sampled column in row i and the previously selected columns $\{g_r^1(i-1,0), \ldots, g_r^1(i-1,T)\}$ from row $i-1$.

Under these constraints, the row-based random sampler can be formalized as:

$$G_r = \left\{ g_r^0(i) = I, g_r^1(i,t+1) \right\},$$

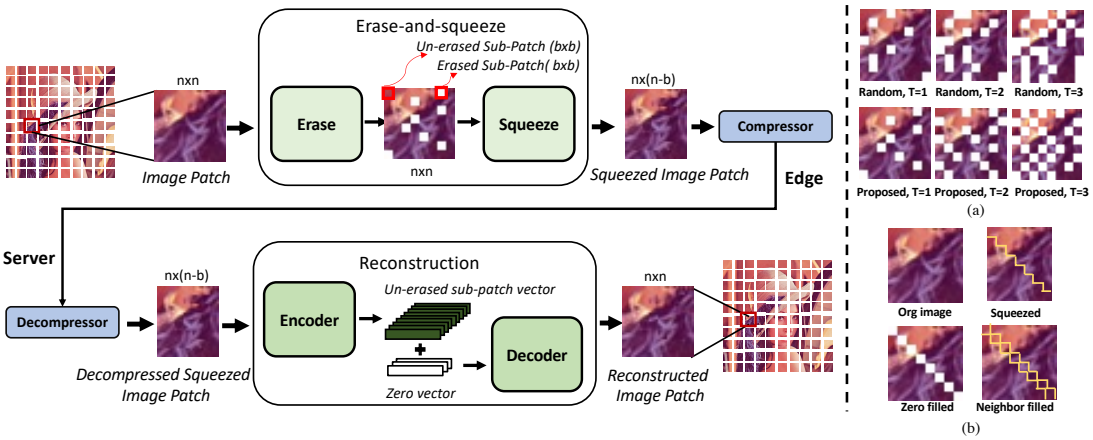
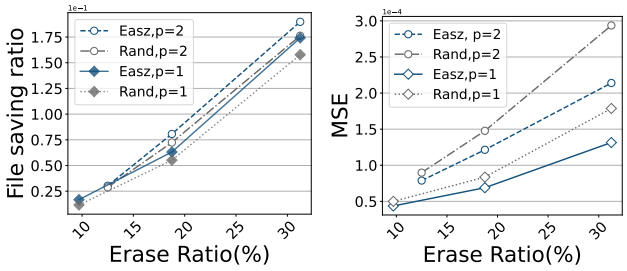


Fig. 2: Left: Easz system overview. A image is going through a two-stage image patchify process, erase-and-squezee process. The squeezed image is then compressed using existing compressor and transmit. On the decompression stage, the erased patch is recovered. Right: (a) Proposed erase methods compared with random erase methods. T indicates an erased item in each row. (b) Different methods to re-organize un-erased image components.



(a) Impact on JPEG (↑ better). (b) Impact on Recon (↓ better).

Fig. 3: The proposed method outperforms random masking in terms of JPEG impact and reconstruction, resulting in a higher file saving ratio and lower MSE on Kodak dataset. The variable p represents patch size.

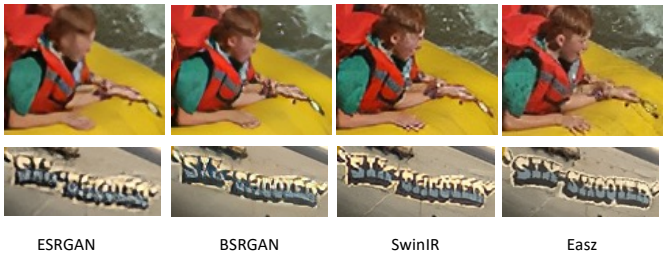


Fig. 4: Easz preserves details better than SR methods via direct pixel prediction, improving PSNR and SSIM.

where

$$g_r^1(i,t+1) \sim \text{clip}\left(\text{Uniform}(0, W-1)\right)$$
$$\text{s.t.} \quad |g_r^1(i,t+1) - g_r^1(i,t)| > \delta$$
$$|g_r^1(i,t+1) - g_r^1(i-1,T)| > \Delta$$

When restricted to T=1 with non-adjacent sampling in both row and columns, it becomes a diagonal mask. With patch=1, T=n/2, and non-adjacent sampling in each row and column, it degrades to 2x super-resolution. Also, we will show in experiment part that generating patches directly results in better reconstruction than existing super-resolution methods.

Squeeze. After applying the generated erase mask $\mathbf{M}$ on $\mathbf{X}$, the next step is to squeeze the non-zero (sampled) locations together to form a smaller image $\mathbf{X}_{\text{squeezed}} \in \mathbb{R}^{h' \times w' \times C}$, where $h' < h$ and $w' < w$ represent the dimensions. This can be achieved by filtering out the zero entries from $\mathbf{X}$:

$$\mathbf{X}_{\text{squeezed}}[i', j'] = \mathbf{X}[i, j] \quad \text{for all } (i, j) \text{ where } \mathbf{M}[i, j] = 1$$

After the process, the squeezed image $\mathbf{X}_{\text{squeezed}}$ would be encoded using compressors like JPEG, BPG, etc to get a compressed form $\hat{\mathbf{X}}_{\text{squeezed}}$ and send, as illustrated in Fig. 2.

B. Reconstruction

Recent work explores pixel-level generation for high-fidelity reconstruction [13]. Easz adopts a transformer to perform pixel-level reconstruction, achieving better results than super-resolution methods, as shown in Fig. 4. It also supports flexible downsampling ratios for adaptive compression. However, directly apply transformers to predict pixels is costly due to the nature of attention mechanisms. In our experiments, a 12GB Nvidia 2080Ti server cannot afford to run pixel-based prediction of a single 256×256 image. Rather than applying conventional model compression techniques [17], our approach is inspired by the block-based strategies employed in traditional image compressors such as JPEG, which use 32×32 or 64×64 blocks to enhance compression performance. We design a two-stage image patchification process followed by a lightweight (8.4MB) transformer for sub-patch restoration.

Complexity Analysis. Predicting pixel values for each element in the image, particularly in high-resolution images, proves to be computationally expensive. Consider a grayscale

image of size 256x256 with a 1x1 patch size. This configuration results in 65,536 pixels. Based on the attention complexity, an image of size (h, w) in grayscale would require $O((hw)^2 \times d_{model})$ calculations if using a pixel as a token. Therefore, calculating a single self-attention for this setup would necessitate $65536^2 \times d_{model} = 4,294,967,296 \times d_{model}$ calculations. Such a computational load is heavy to execute on today's hardware, and it becomes exponentially more challenging when dealing with higher-resolution images.

Two-Stage Image Patchify. In the Easz framework, a two-stage image patchify process is employed to reduce the computational complexity of the vision transformer. Given an input image X with dimensions (h, w), it is first divided into non-overlapping image patches of size (n, n). Each of these $n \times n$ image patches is further subdivided into smaller non-overlapping (b, b) sub-patches. Critical operations such as erasure, squeezing, and reconstruction are performed at the sub-patch level. This patchify process is essential to manage the computational challenge of the vision transformer.

Next, we will explain how the proposed two-stage image patchify process reduces attention complexity. For clarity, the following analysis omit d_{model} as it remains constant. The first stage patchify process split the image into $\frac{hw}{n^2}$ patches, with $n \times n$ size. The computational complexity on this stage is $O(\frac{(hw)^2}{n^4})$. Then, the second patchify process is executed, split the $n \times n$ patches into $b \times b$ sub-patches. Therefore there will be $\frac{(hw)^2}{n^2} \times \frac{n^2}{b^2}$ sub-patches, with $b \times b$ size. We configure the attention mechanism to operate within each image patch, thereby constraining the computational complexity to scale with the size of patches rather than the entire image, resulting in a computational complexity $O(\frac{hw}{n^2} \times \frac{n^4}{b^4} = O(\frac{(hw \times n^2)}{b^4})$. Therefore even when the token size is small, like when $b = 1$, the calculation complexity is still $O(hwn^2)$, which is much smaller than the original complexity $O(hw)^2)$.

Now recall that image of size $(256, 256)$, which would cost $4,294,967,296 \times d_{model}$ calculations originally, after the two-stage image patchify process where $n = 32$ and $b = 4$, would require $1,048,576 \times d_{model}$ calculations, reduced by 4096 times compared to the original.

Model Structure. Fig. 5 illustrates the architecture of our efficient transformer-based reconstruction network. The encoder and decoder are composed of two transformer blocks, each containing three layernorms, one attention layer, and one feedforward layer. Note that the model can work with a variety of input erase ratios, which are controlled by k and b, and hence, we do not need to train a specific model for each erase ratio. Sub-patches can execute in parallel for Easz due to the nature of transformer block calculations [26].

Training Process. Consider a batch of image patches $X_1, X_2, \ldots, X_n$ (each $n \times n$) randomly chosen from our training dataset, with patchification already completed. These patches are divided into smaller sub-patches of size $(b \times b)$, resulting in $(n/b, n/b)$ sub-patches per original patch. Positional embeddings are multiplied with these sub-patches to provide spatial context. An erase mask is then applied to remove k random sub-patches. The encoder E receives the remaining un-erased sub-patches $U = \{u_1, u_2, \ldots, u_m\}$ where $m = n/b \times n/b - k$. Each un-erased sub-patch u_i is projected into an embedding e_i. The two-layer encoder processes these embeddings to extract features: $F = E(e_1, e_2, \ldots, e_m)$. To compensate for erased blocks during reconstruction, we introduce zero vectors $\hat{F} = \{\hat{f}_1, \hat{f}_2, \ldots, \hat{f}_k\}$. We align $\hat{F}$ with the zero-value positions and F with one-value positions based on stored erase mask information. The combined feature sets $\{F, \hat{F}\}$ are fed into a two-layer decoder D which reconstructs the image: $\hat{X} = D(\{F, \hat{F}\})$. Our goal is to fine-tune encoders and decoders (E and D, respectively) so that they minimize discrepancies between original images (P) and their reconstructions ($\hat{X}$), despite interference from zero vectors.

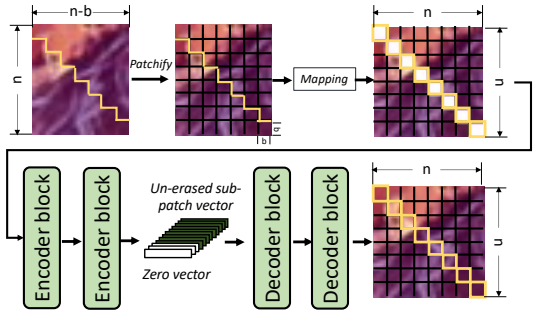


Fig. 5: Reconstruct process illustration.

To minimize the difference between the original image X and the reconstructed image $\hat{X}$, we adopt LPIPS [32], a well-known perceptual loss, along with L1 loss as training loss.

$$\text{Loss}(x, y) = \text{L1}(x, y) + \lambda * \text{LPIPS}(x, y) \tag{2}$$

The feature extraction model is selected as VGG and λ is chosen as 0.3 in our experiments.

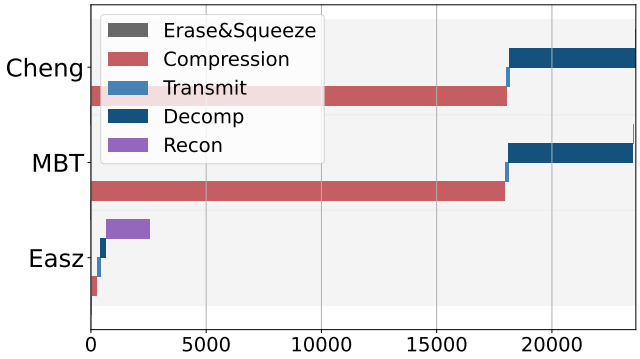
IV. Experiments

A. Experimental Setting

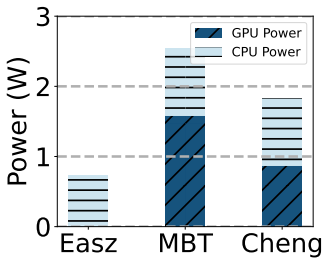
Training setting. The experiments consist of two phases: offline pretraining and online testing. In the pretraining phase, a specific loss function (Eq. 2) is used with these hyperparameters: learning rate of 2.8e-4, erase ratio of 0.25, batch size of 4096, weight decay of 0.05. Randomly generated erase masks are applied for model robustness during this stage. For online testing, a consistent mask is utilized on both edge and server sides. But note that transmission of the mask won't be costly due to its small size—a binary mask at dimensions 32×32 occupies only 128 bytes.

Hardware platforms. Our framework is implemented with ~1000 lines of Python. We use an NVIDIA Jetson TX2 as the edge device and a desktop with Intel i7-9700K CPU and RTX 2080Ti GPU as the server, which are physically connected to a Wi-Fi router and communicate via TCP.

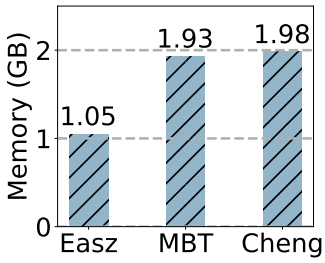
Datasets. During the offline pretraining phase, the CIFAR-10 [2] dataset is employed to pretrain the model, enabling



(a) End-to-End Latency Breakdown(ms).



(b) Encode Power consumption.



(c) Encode Memory footprint.

Fig. 6: Efficiency Evaluation on NVIDIA Jetson TX2.

it to acquire generative capabilities. In the testing phase, two common image compression datasets, Kodak [7] and CLIC [27], are utilized to assess the generative performance of the proposed method.

Metrics. To benchmark against other super-resolution approaches, we include PSNR and SSIM to demonstrate the superiority of our method. Since removing content would generally impact reference-based metrics such as PSNR and SSIM (a trend also observed in other downsampled-and-super-resolution methods), we employ non-reference perceptual metrics for comparison with other compression baselines: Brisque [20], Pi [4], and Tres [9]. Compression performance is evaluated by bits per pixel (BPP).

Baselines. We use four compression methods as baselines to demonstrate the effectiveness of the proposed method: JPEG, BPG, MBT[19], and Cheng-Anchor [6]. Among these, MBT and Cheng-Anchor are two NN-based compression methods.

B. Latency Analysis and Resource Consumption

We first report the latency breakdown of Easz with other neural network-based compression methods, using a Jetson TX2 for compression and a server for decompression. We repeat the runs 24 times and report the average on Fig. 6a. We observe that Erase-and-Squeeze only takes up 0.7% of the end-to-end latency, which induces minimal overhead on the edge device and proves Easz' efficiency, while both MBT and Cheng-Anchor are too compute-intensive to run the compression on the edge side. As expected, the reconstruction in Easz takes the longest time, accounting for 74% of the latency. We argue that the performance can be significantly improved by upgrading to a datacenter-class GPU such as the A100. Additionally, acceleration using FPGAs presents an interesting direction for future exploration [22, 23].

Resource consumption is another critical consideration when it comes to resource-constrained edge devices. To assess this, we measure three key metrics – CPU power, GPU power, and memory footprint – using the Tegrastats Utility [24] on the Jetson TX2. As illustrated in Fig. 6, our findings reveal that Easz excels in all metrics compared to other NN-based compression methods. Specifically, in contrast to MBT and Cheng-Anchor, Easz achieves a remarkable 71.3% and 59.9% reduction in total power consumption. It's noteworthy that Easz does not utilize any GPU power on the edge device, attributed to its lightweight yet effective erase-and-squeeze design. Furthermore, Easz reduces memory footprint by 45.8% and 47.1%, respectively. These results underscore the advantage of deploying Easz on wimpy edge devices.

TABLE I: Comparison with Super-Resolution on Kodak Dataset.

Metrics	Easz	SwinIR	realESRGAN	BSRGAN
PSNR	28.96	24.86	24.85	25.35
MS_SSIM	0.96	0.94	0.93	0.94
Recon Model Size	8.7MB	67MB	67MB	67MB

C. Comparison with Super-Resolution Methods

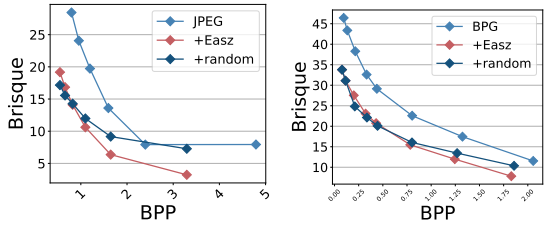
We compare the reconstruction effect of Easz with state-of-the-art super-resolution methods to demonstrate Easz's effectiveness. As shown in Tab. I, Easz outperforms super-resolution in pixel-level reconstruction metrics while having a much more flexible reduction ability. Note that Easz uses a model of only 8.7MB, while other models are 67MB.

D. Ablation Study

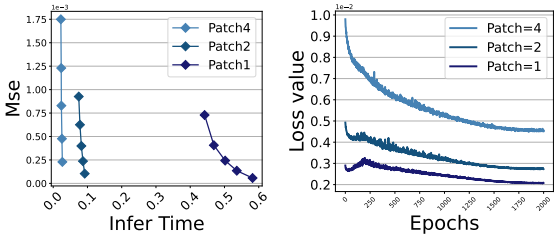
Effectiveness of erase strategy. Fig. 7a and Fig. 7b compare the proposed to erase mask generation method, the random erase mask method, and the baseline (JPEG and BPG) throughout the entire pipeline. It can be observed that the proposed erase mask generation method achieves better BPP at the same quality level on both JPEG and BPG, further substantiating the effectiveness of the proposed method.

Patch size selection. Fig. 7c examines the effects of two hyperparameters, erase block size (1, 2, and 4) and erase ratio (10% to 50%), on compression rate and quality. As the erase ratio increases, MSE rises, indicating lower reconstruction quality. Smaller patch sizes yield better reconstruction due to higher local correlations. Patch size=2 offers a balance between speed—being six times faster than size=1—and quality—with only a slight difference in MSE. Doubling the patch size from 2 to 4 also doubles both speed and MSE. The recommendation is to use smaller patch sizes for practical applications but consider size=2 for additional speed needs.

Effectiveness of fine-tuning. Our model, after pretraining on the CIFAR dataset for 5000 epochs, can be applied to various image compression tasks due to its ability to recognize similarities in local image features. Typically, models are first



(a) JPEG, Brisque(↓ better). (b) BPG, Brisque(↓ better).



(c) MSE(↓ better). (d) MSE(↓ better)

Fig. 7: (a)(b): Comparison between Easz with proposed mask strategy, Easz with random mask strategy, and conventional compression baselines (JPEG and BPG). (c) Patch size and erase ratio's impact on MSE. (d) MSE during Easz fine-tuning process with patch size=1,2,4 on Kodak dataset.

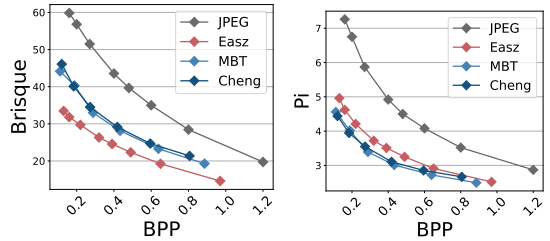
TABLE II: Compression Performance Enhancement on Kodak Dataset and Clic Dataset.

Metrics		JPEG		BPG		MBT		Cheng-anchor	
		Org	*+Proposed*	*Org*	*+Proposed*	*Org*	*+Proposed*	*Org*	*+Proposed*
Kodak	BPP	0.412	0.411	0.382	0.410	0.433	0.389	0.418	0.402
	Brisque	43.06	22.34	30.675	23.27	28.13	18.63	29.16	20.51
	Pi	4.84	3.33	3.07	3.04	3.01	3.00	3.11	3.05
	Tres	77.62	86.26	83.55	85.88	84.14	88.03	88.53	89.80
Clic	BPP	0.306	0.307	0.308	0.293	0.308	0.292	0.287	0.267
	Brisque	60.51	23.63	39.95	25.27	32.20	18.37	35.42	21.55
	Pi	8.51	5.02	4.85	4.66	4.33	4.35	4.58	4.50
	Tres	50.65	63.69	65.14	67.08	73.54	78.30	82.91	83.95

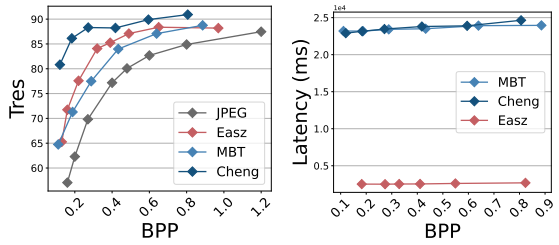
pre-trained on a large dataset and then fine-tuned for specific tasks. We tested if fine-tuning our pretrained model with the Kodak dataset would be beneficial and found that it indeed improves performance by reducing losses across different patch sizes (1×1, 2×2, and 4×4), as shown in Fig. 7d.

E. Improvement on Existing Compressors

To evaluate how well Easz works with leading compressors, we incorporated it into four established methods: JPEG and BPG (traditional compressors), as well as MBT and Cheng-anchor (neural network-based compressors). We used two datasets, Kodak and CLIC, to test the resilience of Easz across different types of image data. For the Kodak dataset, we aimed for a bit-per-pixel (BPP) rate of approximately 0.4; for the CLIC dataset, we targeted a BPP of around 0.3 to gauge Easz's efficacy at varying levels of compression. The results showing how each baseline method performs on its own and when combined with Easz are detailed in Tables II. The data clearly shows that integrating these baselines with Easz consistently enhances perceptual quality without increasing BPP.



(a) Brisque (↓ better). (b) Pi (↓ better).



(c) Tres (↑ better). (d) End-to-end Latency.

Fig. 8: Compression performance of Easz, JPEG, MBT and Cheng on three perceptual metrics (a-c). Fig. 8d evaluates the end-to-end latency on our testbed.

F. End-to-End Compression Performance

In this experiment, we use JPEG+Easz as the baseline and observe changes in three perceptual metrics at different bitrates (BPP). Notably, JPEG alone underperforms compared to two deep-learning compression methods at all compression levels. However, with Easz enhancement, JPEG shows a marked improvement. For the BRISQUE metric specifically, JPEG+Easz exceeds both deep-learning methods. Regarding the Pi metric, JPEG+Easz matches the performance of these methods. With the Tres metric, while JPEG+Easz outdoes MBT, it falls short of Cheng-anchor's results. Overall, Easz boosts JPEG to compete effectively with other state-of-the-art deep-learning compression techniques in each perceptual measure. Easz also outperforms two neural network-based methods in latency, with an average end-to-end latency of 2568ms across different bitrates per pixel, marking an 89% reduction compared to MBT and Cheng's methods.

V. Conclusion

This paper proposes Easz, which overcomes the challenges faced by modern neural image compression on edge devices. Easz is a comprehensive and efficient data compression framework, which shifts the computational burden to the server side. Easz includes an Erase-And-Squeeze process on the edge, coupled with a transformer-based encoder-decoder architecture on the server. Through careful design of the Erase-and-Squeeze strategy, Easz enhances the performance of existing methods while enabling dynamic adjustment of compression levels. As a result, Easz eliminates the computational and storage burdens on the edge while simultaneously boosting compression rates and transmission efficiency. Our real-world evaluation in an edge-server testbed demonstrates Easz's improvement.

References

[1] *AI and Compute*. 2018. URL: https://openai.com/blog/ai-and-compute/.

[2] Krizhevsky Alex et al. "Learning multiple layers of features from tiny images". In: *2009* (2009).

[3] Fabrice Bellard. *BPG*. https://bellard.org/bpg/. 2014.

[4] Yochai Blau et al. "The 2018 PIRM challenge on perceptual image super-resolution". In: *ECCV Workshops*. 2018.

[5] *Bringing Cutting-Edge Technology to Wildlife Conservation*. https://www.wildlifeinsights.org/. 2023.

[6] Zhengxue Cheng et al. "Learned Image Compression with Discretized Gaussian Mixture Likelihoods and Attention Modules". In: *CVPR*. 2020.

[7] Eastman Kodak Company. *Kodak Lossless True Color Image Suite*. https://r0k.us/graphics/kodak/. 1993.

[8] Shilpa George et al. "Towards Drone-Sourced Live Video Analytics for the Construction Industry". In: *Proceedings of the 20th International Workshop on Mobile Computing Systems and Applications*. ACM. 2019, pp. 3–8.

[9] S Alireza Golestaneh et al. "No-reference image quality assessment via transformers, relative ranking and self-consistency". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2022, pp. 1220–1230.

[10] JPEG Group. *JPEG*. https://jpeg.org/. 1986.

[11] Yujun Huang et al. "Compressive sensing based asymmetric semantic image compression for resource-constrained IoT system". In: *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 2022, pp. 877–882.

[12] Charles Laroche et al. "Deep model-based super-resolution with non-uniform blur". In: *Proceedings of the IEEE/CVF winter conference on applications of computer vision*. 2023, pp. 1797–1808.

[13] Zhenglin Li et al. "Mapping new realities: Ground truth image creation with pix2pix image-to-image translation". In: *arXiv preprint arXiv:2404.19265* (2024).

[14] Yu Liang et al. *Ariadne: A Hotness-Aware and Size-Adaptive Compressed Swap Technique for Fast Application Relaunch and Reduced CPU Usage on Mobile Devices*. 2025. arXiv: 2502.12826 [cs.OS]. URL: https://arxiv.org/abs/2502.12826.

[15] Yu Mao et al. "Accelerating General-Purpose Lossless Compression via Simple and Scalable Parameterization". In: *Proceedings of the 30th ACM International Conference on Multimedia*. 2022, pp. 3205–3213.

[16] Yu Mao et al. "Faster and Stronger Lossless Compression with Optimized Autoregressive Framework". In: *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 2023, pp. 1–6.

[17] Yu Mao et al. "On the compressibility of quantized large language models". In: *arXiv preprint arXiv:2403.01384* (2024).

[18] Yu Mao et al. "TRACE: A Fast Transformer-based General-Purpose Lossless Compressor". In: *Proceedings of the ACM Web Conference 2022*. 2022, pp. 1829–1838.

[19] David Minnen et al. "Joint Autoregressive and Hierarchical Priors for Learned Image Compression". In: *NeurIPS, Montréal, Canada*. 2018, pp. 10794–10803.

[20] Anish Mittal et al. "No-reference image quality assessment in the spatial domain". In: *IEEE Transactions on image processing* 21.12 (2012), pp. 4695–4708.

[21] Brendan Reidy et al. "HiRISE: High-Resolution Image Scaling for Edge ML via In-Sensor Compression and Selective ROI". In: *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 2024, pp. 1–6.

[22] Dongdong Tang et al. "p LPAQ: Accelerating LPAQ Compression on FPGA". In: *2022 International Conference on Field-Programmable Technology (ICFPT)*. IEEE. 2022, pp. 1–6.

[23] Dongdong Tang et al. "Stem: Streaming-based fpga acceleration for large-scale compactions in lsm kv". In: *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE. 2024, pp. 3893–3905.

[24] *Tegrastats Utility*. https://docs.nvidia.com/drive/drive_os_5.1.6.1L/nvvib_docs/DRIVE_OS_Linux_SDK_Development_Guide/Utilities/util_tegrastats.html. 2023.

[25] *The Future of Computing is Distributed*. [Accessed on October 14, 2023]. 2020. URL: https://www.datanami.com/2020/02/26/the-futureof-computing-is-distributed/.

[26] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[27] Workshop et al. *Challenge on Learned Image Compression*. https://compression.cc/. 2022.

[28] Li Xie et al. "Gaussian Rate-Distortion-Perception Coding and Entropy-Constrained Scalar Quantization". In: *arXiv preprint arXiv:2409.02388* (2024).

[29] Li Xie et al. "Output-constrained lossy source coding with application to rate-distortion-perception theory". In: *IEEE Transactions on Communications* (2024).

[30] Guanghao Yin et al. "Online Streaming Video Super-Resolution with Convolutional Look-Up Table". In: *arXiv preprint arXiv:2303.00334* (2023).

[31] Jingyu Zhang et al. "Research on the application of computer vision based on deep learning in autonomous driving technology". In: *International conference on wireless communications networking and applications*. Springer. 2023, pp. 82–91.

[32] Richard Zhang et al. "The unreasonable effectiveness of deep features as a perceptual metric". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 586–595.

MEEK: Re-thinking Heterogeneous Parallel Error Detection Architecture for Real-World OoO Superscalar Processors

Zhe Jiang†, Minli Liao¶,Sam Ainsworth‖ Dean You†, Timothy Jones¶
†National Center of Technology Innovation for EDA, School of Integrated Circuits, South East University, People's Republic of China,
¶University of Cambridge, United Kingdom, ‖University of Edinburgh, United Kingdom

***Abstract*—Heterogeneous parallel error detection is an approach to achieving fault-tolerant processors, leveraging multiple power-efficient cores to re-execute software originally run on a high-performance core. Yet, its complex components, gathering data cross-chip from many parts of the core, raise questions of how to build it into commodity cores without heavy design invasion and extensive re-engineering.**

We build the first full-RTL design, MEEK, into an open-source SoC, from microarchitecture and ISA to the OS and programming model. We identify and solve bottlenecks and bugs overlooked in previous work, and demonstrate that MEEK offers microsecond-level detection capacity with affordable overheads. By trading off architectural functionalities across codesigned hardware-software layers, MEEK features only light changes to a mature out-of-order superscalar core, simple coordinating software layers, and a few lines of operating-system code. The Repo. of MEEK's source code: https://github.com/SEU-ACAL/reproduce-MEEK-DAC-25

I. Introduction

Hardware faults, both permanent and transient, can induce system anomalies and execution errors, which become more common with the increasing number of transistors and lower operating voltages in modern processors [1], [2], [3], [4], [5]. To mitigate the errors caused by hardware faults, protection mechanisms exist, ranging from error codes to fault-tolerant architectures. Detection is always key: once an error is detected, the system can transition to a safe state, enabling corrective actions (e.g., system recovery or fault isolation). As mandated by global safety standards from industry, e.g., ISO26262 for automotive [6] and DO-178C [7] for avionics, hardware faults must be addressed before escalating into hazards, i.e., within the Fault Tolerance Time Interval (FTTI), often measured in milliseconds [8].

Software mechanisms (e.g., multithreading [4], [9] and software scanner [10], [11]) typically incur significant performance degradation or offer limited fault coverage [12], making them insufficient for processors that require stringent reliability standards (e.g., ASIL-D in ISO26262 [6]). Hardware mechanisms often employ a dedicated core to execute a program copy, enabling the comparison of the core's pins at each clock cycle (e.g., locksteps [13], [14], [15]). By replaying everything on a separate, synchronized core and performing run-time verification at the signal level, full coverage and real-time guarantees are achieved. Although dual- and triple-core locksteps have been successfully applied to microcontroller-sized processor cores in many life-critical application scenarios [16], [17], they have been shown impractical for Out-of-Order (OoO) superscalar cores due to prohibitive energy, area, and thermal costs [18], [19], [20].

Heterogeneous parallel error detection. As a promising alternative, heterogeneous parallel error detection [21], [22], [2] leverages strong induction to divide a software program running on an OoO superscalar high-performance core (big core) into multiple discrete segments using Register Check Points (RCPs) and re-execute them on sets of smaller, power-efficient cores (little cores) for verification.

To replay memory and other non-repeatable operations, the addresses and data of relevant instructions (e.g., `load` and `store`) are extracted from the program stream at the commit stage within the big core, generating a partitioned, distributed log of load and store operations. When the log of a segment is filled, or an instruction timeout is reached, a new RCP is triggered, and the corresponding little core begins verifying the segment between Start RCPs (SRCPs) and End RCPs (ERCPs). By overlapping verification jobs, little cores collectively offer sufficient computational capacity to keep pace with the big core, ensuring full coverage with low overheads [21].

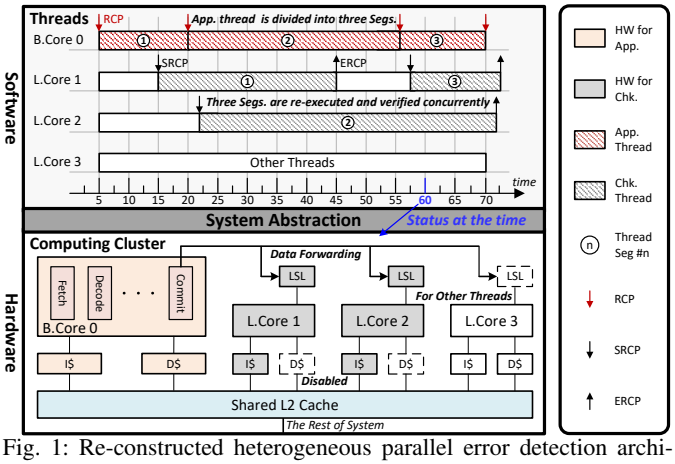


Fig. 1: Re-constructed heterogeneous parallel error detection architecture *(RCP: Register Checkpoint; S/ERCP: Start/End RCP LSL: Load-Store Log)*: an application thread on big core 0 is divided into three Segs. using RCPs, replayed and verified on little core 1 and 2.

Challenges. Unlike conventional lockstep cores, which are typically implemented using identical processor cores with multiplexers and comparators [23], heterogeneous error detection relies on intricate asynchronous interactions between the big core and a sea of little cores. This involves continuously collecting and distributing loads, stores, and periodical register checkpoints from a big core and flexibly managing little cores. This data needs to be preserved from execution time until commit, before transmission in order, and handling superscalar commit. This also means that the collected data must be prioritized and routed to the corresponding little cores at high bandwidth, to prevent backpressure from stalling the big core.

Existing work [21], [22] on the architecture has been studied through abstract simulation, modeled upon idealistic assumptions in microarchitecture and without an OS, imposing barriers for practical deployment. For instance, it is unclear how to collect such massive data from different locations of a real core without heavy microarchitectural changes, route it cross-chip while handling contention, and provide a configurable framework for little cores' management.

Contributions. We show it is possible to build heterogeneous error detection into real cores with minimal changes, by re-constructing the concept via hardware/software codesign, trading off functionalities across system layers. This allows an application thread on a big core to be segmented using RCPs, replayed, and verified using any number of checker threads deployed on any little cores while still allowing other threads to be executed (Fig. 1). We build a full-stack framework, **Make Each Error Count (MEEK)**, upon an open-source SoC [24], demonstrating the architecture can be realized with minimal design invasion within mature cores, and a few line changes in a full Linux.

II. MEEK: A CPU/OS Codesigned Approach

To implement MEEK at minimal complexity while achieving high performance, we had to make careful design-choice partitions between hardware and software (figure 2). On one hand, we had little choice but to implement load- and store-logging, for replay of execution on little cores, in hardware: using software-based instrumentation [25] would have meant infeasibly high overheads. On the other hand, since little cores perform page-table walks asynchronously with

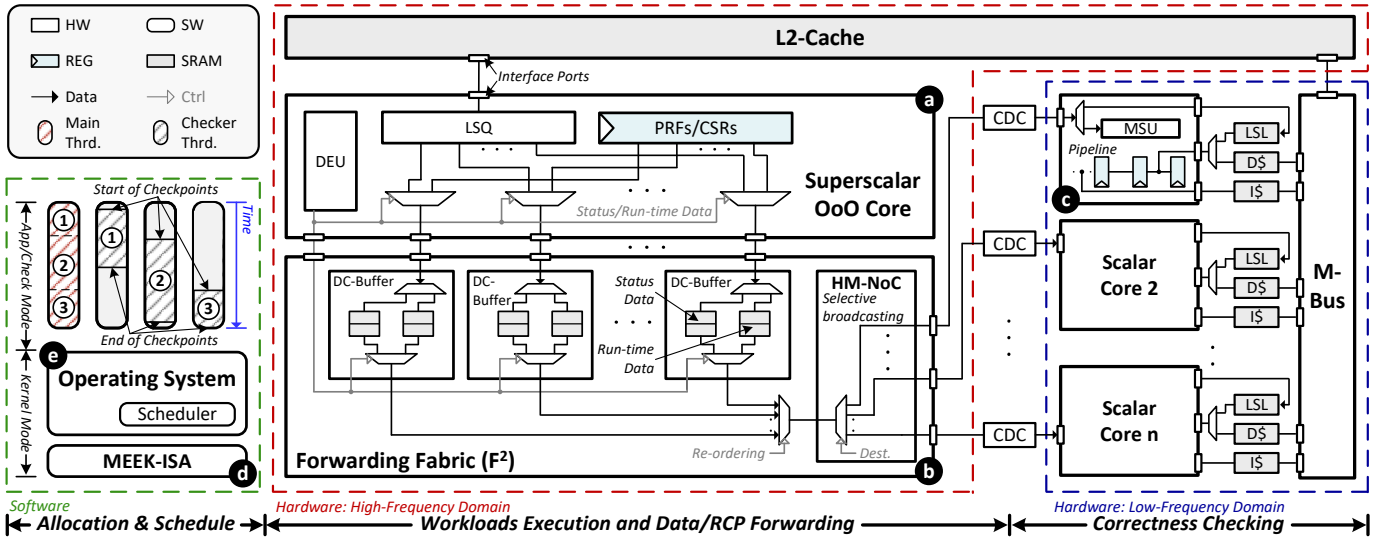


Fig. 2: An overview of MEEK. *(DEU: Data Extraction Unit; LSQ: Load-Store Queue; PRFs: Physical Register Files; CSRs: Control and Status Registers; HM-NoC: Half-duplex Multicast Network-on-Chip; MSU: Mode Switch Unit; LSL: Load-Store Log; M-Bus: Memory Bus).* At hardware: ⓐ a non-intrusive DEU is deployed in the big core, collecting data from various locations without disrupting the core's execution; ⓑ a bespoke forwarding fabric F^2 is developed, prioritizing and distributing the data only to the relevant little cores; ⓒ dual-mode little cores are designed for correctness checking or workload execution. At software: ⓓ a customized ISA abstracts the control interface for ⓔ, the scheduler in the OS, enabling flexible management of verification and workload execution on the little cores.

the big core, the OS had to be aware of them in some form, and so we let the OS fully control their scheduling, avoiding implementing complex decisions that were not on the performance critical path in hardware, and allowing little cores to execute standard processes as well. Rather than a fully transparent interface, and to avoid expensive full error correction in hardware [22], programs interact with the MEEK-ISA by being wrapped with coordinator functions inserted before main, which request checker resources from the OS, verify checking outputs, and call fault-handling code if needed.

To do so, we slightly modified the big core's microarchitecture to insert a read-only observation channel at the commit stage (Fig. 2 ⓐ), collecting the big core's *status data* (i.e., architectural, control and status register files) at each RCP and the *run-time data* (addresses and data of memory and other non-repeatable operations) between RCPs. We built a dedicated data fabric (Fig. 2 ⓑ), selectively broadcasting/routing the extracted data to the little core(s), minimizing the backpressure from the data communications. In little cores, the received data is buffered in a Load-Store Log (LSL), replacing the L1 cache during program replay, allowing the little core to reset its architectural state to a given SRCP, replay the exact instructions between the RCPs, and verify execution correctness at the ERCP, using the different types of data (Fig. 2 ⓒ). Our ISA interface (Fig. 2 ⓓ and Tab. I) (re-)configures the little cores' checking characteristics, i.e., the operational mode (application or check mode).

Detection approach. Error checking is parallelized using checker threads (Fig. 2 (b)): an application thread is segmented using RCPs, taken when the targeted LSL is full, an instruction timeout is reached, or the kernel mode is trapped. These segments are run in a second time by checker threads, assuming all previous segments are correct. After re-execution, the checker thread compares its architectural registers against the ones provided by the application thread at the same RCP. If the registers match, the segment is considered correct. If all segments pass check, the entire execution is deemed correct[1].

III. The Microarchitecture

Even with the reduced hardware feature-set discussed in section II, MEEK required careful microarchitectural engineering to allow efficient, low-overhead execution, particularly around data transmission. We identify and ameliorate many key bottlenecks that were missing from the analyses and high-level simulations presented by previous work [21], [22], [26] due to a lack of RTL implementation. We also identify inefficiencies in terms of redundant data storage, where much of the information required by the forwarding paths is already buffered inside the core until commit time, meaning dedicated structures are not needed; instead we can forward data from the existing structures. Despite bottlenecks, we demonstrate that it is possible to build MEEK's microarchitecture from a mature heterogeneous SoC with very minor changes, avoiding heavy engineering efforts.

We build MEEK into an open-source heterogeneous SoC (Rocket Chip [24]), with both high-performance and energy-efficient cores (BOOM and Rocket). We leverage Rocket Chip to demonstrate the applicability of our methodology to the other heterogeneous SoCs, e.g., ARM's big.LITTLE [27] and Intel's P- and E-class cores [28].

TABLE I: MEEK ISA *(Priv 1/0: kernel/user modes).*

Instruction	Priv	Description
`b.hook` rs1, rs2	1	Hook big core rs1 with little core rs2.
`b.check` rs1	1	Enable/Disable checking capacity.
`l.mode` rs1, rs2	1	Switch little core rs1's mode to rs2.
`l.record` rs1	0	Record arch. registers to address rs1.
`l.apply` rs1	0	Apply arch. registers from address rs1.
`l.jal` rs1	0	Jump to rs1 (PC of main thread).
`l.rslt` rd	0	Return the check results.

A. OoO Superscalar Core (The Big Core)

Program replay requires the collection of both status data and run-time data from the big core. The data is temporarily stored within the core during the program executions but distributed across various locations. The status data is buffered in the Physical Register Files (PRFs) and Control and Status Registers (CSRs), while the run-time data is stored in the Load-Store Queue (LSQ)[2] and CSRs. All can be directly extracted but must occur in a narrow window, at commit

[1]Memory operations, however, cannot be verified at ERCPs. In the re-execution, addresses and data of memory operations are compared directly in the LSL. A similar design method is used for Control and Status Registers (CSRs) to verify non-repeatable instructions.

[2]By building MEEK, we noticed that Ainsworth et al. [21]'s policy, of storing a dedicated buffer of load data, indexed by Re-Order Buffer (ROB) entry, was unnecessary. All the need was already buffered in LSQ. Yet, light changes were needed to preserve redundancy and thus fault tolerance, as while we assume data is protected via parity in the cache [29], and via full duplication by the point of reaching the LSL, there is a small window in the LSQ where it is otherwise protected by neither. We copy the cache's parity bits into the LSQ, double-checking them once the data is forwarded.

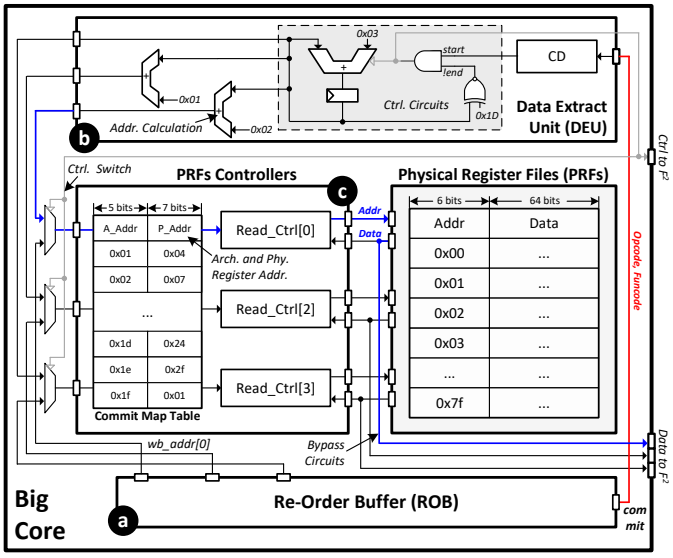


Fig. 3: Big core microarchitecture, extracting status data *(red: ROB to DEU; blue: DEU to PRFs)*: (a) at commit time, opcode and function code are routed; (b) DEU determines whether to extract status data; (c) if so, a signal is routed and preempts the PRF controller for reading.

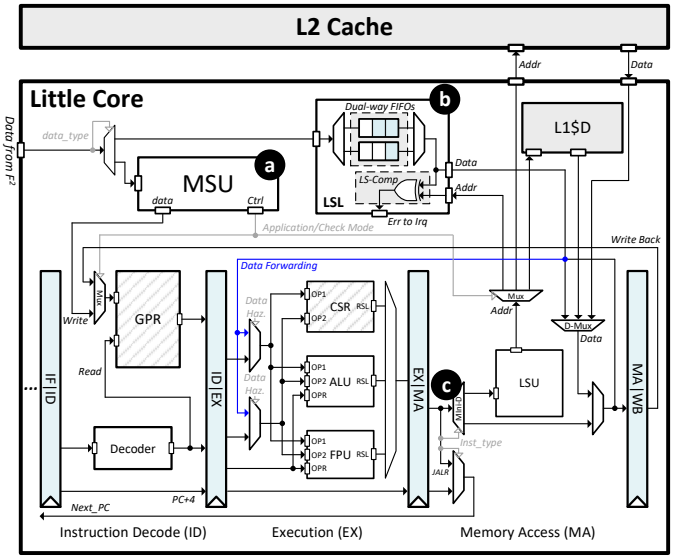


Fig. 4: Little core microarchitecture, upgraded with the MSU and LSL: the (a) MSU servers as the control engineer, and the (b) LSL buffers packets from F^2. While running the checker thread, the MSU manages its status, and memory data is fetched from the LSL.

time rather than execution time to reflect ordering, and before being overwritten by next instructions. In light of this, we develop a *non-intrusive* DEU to establish an observation channel (Fig. 3), containing a Commit Detector (CD), control circuits and bypass circuits, injected at the PRFs, CSRs, and LDQ. The CD monitors instruction commits from the ROB and selects the bypass circuits to extract data at RCPs or run-time data between RCPs. This enables timely data extraction without bringing extra buffers or altering existing register paths.

PRF example. Fig. 3 reveals the microarchitecture using the PRFs (used to collect RCPs) as an example, where the DEU interfaces the ROB, PRFs, and F^2. At each instruction's commit, the opcode and function code are routed from the ROB (Fig. 3 (a)) to the CD, allowing the CD to determine whether to extract data (Fig. 3 (b)). If an RCP is reached, a signal is generated by the control circuits, preempting the PRF controller to read register files and forward them to the F^2 (Fig. 3 (c)). The PRF controllers are multiplexed between the ROB and the DEU, where the DEU has priority access when required, enabling immediate data reading and preventing data overwriting[3].

B. Forwarding Fabric (F^2)

Previous simulations adopted a naive model for data forwarding, ignoring contention and critical paths [21], [22], [30], [31]. Our attempt to implement such a mechanism revealed a stumbling block impeding system performance, even with the deployment of a full-featured AXI interconnect [32], [33] (Sec. V-D). This is caused by the huge amount of run-time data generated by the big core's parallel commits, in congestion with the frequent reaches of RCPs, which tends to occur in bursts, particularly when the core retires multiple `load` or `store` instructions at RCP boundaries, requiring many data transfers within a single cycle and high throughput.

We design F^2 with Dual-Channel Buffers (DC-Buffers) and a Half-duplex Multicast NoC (HM-NoC), storing and routing the extracted data (Fig. 2 (b)). A DC-Buffer is connected to each commit path, adding independent FIFOs for status and run-time data. This ensures that all run-time data can be stored at the same cycle of commit, even when status data is generated simultaneously, avoiding data needing to be stored inside the core's own structures for longer than in the original core design. HM-NoC uses a half-duplex (1-to-N) Manhattan grid. To achieve high enough throughput, this NoC allows the transmission of two packets per cycle, while preserving ordering[4].

C. In-order Scalar Core (The Little Core)

To enable thread-level error detection and allow the co-existence of the checker and application threads, the little core's microarchitecture must support different operational modes and abstract an interface for software control. Hence, we upgrade the little core's microarchitecture (Fig. 4) with a Mode Switch Unit (MSU, Fig. 4 (a)) and a Load-Store Log (LSL, Fig. 4 (b)). The MSU serves as the control engine, and the LSL buffers received packets. While replicating the big core's states, the MSU records the little core's architectural register files and replaces them with the status data from the LSL. The MSU also manages the little core's operational mode by inspecting the Thread ID (TID) of the running thread with that of the checker thread. When the checker thread is scheduled, the MSU switches the operational mode, where the results of `load` and non-repeatable instructions are fetched from the LSL. Given that the little core accesses the LSL in an in-order manner, we implement the LSL bank using dual-way FIFOs, reducing complexity than conventional way-associative architectures.

Pipeline integrations. Fig. 4 shows the integration of LSL and MSU into a 5-stage pipelined little core. The LSL is added into the Memory Access (MA) stage (Fig. 4 (b)) by deploying a multiplexer and connecting it to the address port of the Load-Store Unit (LSU). The multiplexer selectively routes memory accesses to the LSL based on the operational mode (returned by the MSU) and combines the virtual index and physical tag (returned by the TLB) into the address port of the LSL. Also, a demultiplexer is integrated into the MA stage to direct the read data back to the next stages. A pair of multiplexer and demultiplexer are deployed into the Instruction Decode (ID) stage to allow the recording and updating of the architectural registers. Lastly, we deploy a Mini-Decoder (Mini-D, Fig. 4 (c)) at the MA stage to differentiate conventional RISC-V and MEEK-ISA.

Performance-gap mitigation. In previous work [21], [22], [26], a lack of microarchitectural detail meant bridging the performance gap

[3]A similar microarchitecture is implemented for the CSRs, allowing the data extraction from arbitrary CSR addresses. In contrast, because the top of the LSQ consistently holds data from the most recently retired instructions, when the CD decides to forward the run-time data, the bypass circuits directly transmit from the queue top, minimizing the design complexity.

[4]Unlike previous work, where all data of a segment is buffered and forwarded collectively at an RCP, F^2 enables immediate data transmission and use upon collection time, allowing earlier re-execution by the little core. Also, as the same status data might be required by two little cores (respectively used as the SRCP and ERCP), the data is selectively broadcast to the little cores when they are capable of data receiving, eliminating redundant transactions.

Algorithm 1: Big core's context switch *(Blue: added code)*.

```
1  ▷ Scheduler
2  Function Context_Switch(task *current, core core_index):
3      MEEK.b.check(DISABLE);
4      Kernel.Intr(DISABLE);
5      task *next = NULL;
6      /* switching current task to the next task */
7      Kernel.Context.save(current);
8      next = Kernel.Find_next();
9      if (next→new_release) then
10         for i = 0 to (size_of(next→checker_index) −1) do
11             /* hook little cores to the big core */
12             MEEK.b.hook(core_index, next→checker_index[i]);
13         end
14         Kernel.Context.init(next);
15     else
16         Kernel.Context.restore(next);
17     end
18     current = next;
19     Kernel.Intr(ENABLE);
20     MEEK.b.check(ENABLE);
21     Kernel.Context.jalr(current→pc);
22 End Function
```

Algorithm 2: Little core's context switch and checker thread.

```
1  ▷ Scheduler
2  Function Context_Switch(task *current, core core_index):
3      MEEK.l.mode(MODE_APPLICATION);
4      /* switching current task to the next task */
5      ...
6      if (next→checker_thread) then
7          MEEK.l.mode(MODE_CHECK);
8      end
9      Kernel.Context.jalr(current→pc);
10 End Function
11 ▷ Checker Thread, Newly Developed
12 Function Checker_Thread():
13     /* initializing checker thread using P-Thread */
14     ...
15     MEEK.l.record(sp); // after checking, returns here
16     if (!MEEK.l.rslt()) then
17         MEEK.ReportErr();
18     end
19     while (MEEK.NewSRCP()→invalid);
20     MEEK.l.apply(LSL);
21     MEEK.l.jal(MEEK.NewSRCP()→pc);
22 End Function
```

between the big and little cores involved merely scaling the core count of the little cores. Unfortunately, when we build MEEK using real RTL, the Rocket little cores are each a significant fraction of the size of the big core, and thus using twelve of them as in the original deisgn [21] would be infeasible (section V-E). In particular, we discovered that there was a wide variety in the little cores' ability to keep up with the big core depending on instruction distribution [34]. With this, we realized that the best way to minimize core overhead while achieving good performance was not to use the smallest Rocket cores available, but rather to *balance* their bottlenecks with respect to BOOM. To reduce the performance gap, we optimize little cores by increasing the size of bottlenecked components, e.g., increasing divider unrolling and extending FPU pipeline.

IV. The ISA, Operating System, and Programming Model

We detail ISA support needed to add to augment programs with fault-tolerance support. With just a few lines-of-code changes to the kernel, it can schedule and reserve resources for checker threads while allowing standard scheduling and context switching of other threads.

A. ISA Support

The new ISA is classified into two categories (Tab. I) for the big core (`b.x()`) and the little core (`l.x()`). We use `b.hook()` to set the association between the big and little cores, followed by `b.check()` to enable/disable the checking capacity via switching on/off the DEU. For little core, `l.mode()` sets its operational mode, and a pair `l.record()` and `l.apply()` record and apply the architectural registers from a given source. To re-direct the PC to an application thread, we develop a `l.jal()`, modified from the vanilla jump instruction with an alteration on the target. By treating a checkpoint-end as branch-like, the pipeline handles control hazards from the PC-change, without further changes. Lastly, `l.rslt()` indicates whether an RCP mismatch is detected. As `b.hook()` and `b.check()` can lead to contention in use of the little cores, and the `l.mode()` can cause erroneous execution from unintended memory accesses, they are privileged instructions, executed via OS syscall.

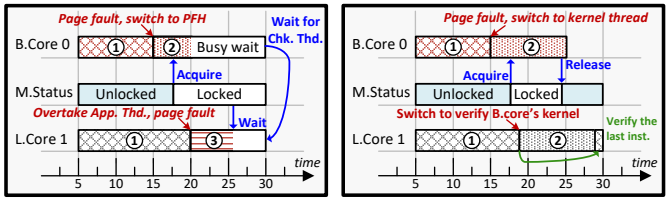
B. Checker Thread and Its Programming Model

The checker thread is initialized with the application thread by augmenting the application thread's main function using constructor and destructor functions [35] (Al. 1: line 14). Since the checker thread relies on the data in LSL to replay memory operations and the LSL is designed using FIFOs, context switches of the log are undesirable. Hence, during the scheduling time, LSL is reserved for a single checker thread (Al. 1: line 12), even if multiple threads can be scheduled on the core. Once LSL is reserved, only data relevant to the associated checker thread is forwarded until re-execution is complete. Likewise, a checker thread pinned to a specific application thread cannot be migrated before the re-execution completes Since each checkpoint is finite in size (5000-instruction maximum), and since ownership returns to the OS after the end of each checkpoint for reallocation, this does not cause resource starvation.

Programming model. We develop the checker thread based on the new ISA, ensuring minimal coding efforts: initially, an `l.record()` is employed to record the current architectural register state (Al. 2: line 15), allowing the core to return after verification. Then, a busy loop is created to await the arrival of status data in the LSL (Al. 2: line 19). Upon receiving it, `l.apply()` is invoked to modify the core's architectural state in accordance with the application segments (Al. 2: line 20), with a `l.jalr()` being invoked to redirect PC to the replicated target (Al. 2: line 21). Lastly, a `l.rslt()` is used to return the verification status. If an error is detected, an interrupt is triggered to notify the OS for corrective actions (Al. 2: line 16 - 18).

C. OS Kernel and Its Verification

With the new ISA, kernel modifications can be constrained to the context switch function within the scheduler, allowing the configuration of checking capabilities and the management of checker threads [36], [37]. In accordance with the distinct roles of the big and little cores, the context switches for application and checker threads are modified differently. While entering the context switch from an application thread, `b.check(DISABLE)` is invoked to deactivate the checking functionality (Al. 1: line 3), and upon leaving the context switch, `b.check(ENABLE)` is called to reactivate it (Al. 1: line 20). Furthermore, if a newly released thread is scheduled for execution, the `b.hook(core_index, next→checker_index[x])` is used to associate it with the little cores (Al. 1: lines 10 - 13). For the



(a) Little core overtakes big core, and attempt to acquire the lock held by big core for PFH, causing a waiting chain.

(b) By making the little core one inst. behind, page fault is always handled by the big core, avoiding dead lock.

Fig. 5: A deadlock occurs when the little core tries to acquire the lock held by the big core. By building synchronization between cores and I/Os, preventing the checker ever needing to claim locks, the deadlock is resolved *(M.Status: Memory Status; PFH: Page Fault Handler)*.

context switch of application threads, the only required modification is using the `l.mode()` to switch the little core's operational mode (Al. 1: lines 3 and 7), leaving the remainder unchanged.

Kernel verification and deadlock resolving. Since MEEK enables error checking at the thread level, the OS kernel can be treated as a specialized application thread and verified like the other application threads. Yet, during the development, we observed a ***deadlock*** missed in previous literature [21], [22], [26] due to a lack of evaluation with OS. Because a checker thread can block a main thread until the little core has finished, due to SRAM logs being finite, this behavior acts as a lock that the little core holds and the big core needs[5].

This deadlock can be fixed by making it impossible for a checker thread to fault on instructions. Ensuring the checker thread is at least one instruction behind the main thread means the latter will reach faults first. Synchronizing on I/O makes it impossible to write out a page potentially used by an unfinished checker thread (Fig. 5 (b))[6].

V. Evaluation

We built MEEK upon an open-source heterogeneous SoC (Rocket Chip [24]), featuring both high-performance (BOOM [39]) and energy-efficient (Rocket [24]) cores. The BOOM was augmented with DEU and F^2 as the big core, and the Rocket was upgraded with LSL and MSU to support re-execution as the little core. We implemented the microarchitecture with Chisel (v3.4) and synthesized the RTL using Vivado toolchains (v2021.2). The generated netlist was deployed on AMD UltraScale+ FPGAs using FireSim [40], emulating the setup in Tab. II. We booted Linux kernel v5.7, executing full SPECint 06 [41] and Parsec V3 [42] with simmedium dataset.

A. Performance Overhead

Fig. 6 shows the slowdown experienced by the big core when running SPECint and Parsec in MEEK, compared to the software-based counterparts implemented in LLVM (Nzdc [43])[7] and LockStep with Equivalent Area (EA-LockStep). Nzdc was chosen for comparison as it is the only open-source software mechanism available, while lockstep is the most widely used hardware mechanism. However, simply duplicating the core in lockstep would consume double the area of the big core while achieving the same performance as a vanilla core, leading to uninteresting comparisons. Therefore, we scaled down the big core's configurations, through linear interpolation on each configurable BOOM component, to create a comparator with both cores combined matching the area overhead of MEEK. In all cases, MEEK is configured with four little cores.

Using four little cores is sufficient to execute SPEC with overheads of 1.4% and Parsec at 4.4% (geomean). For all workloads, except swaptions, the observed slowdown is below 5%. Swaptions, however, suffers the highest slowdown of 22%, due to the frequent divisions, where the Rocket core's divider is significantly less performant than the BOOM core's. For the comparators, Nzdc introduces geomean overheads of 60.2% on Parsec and 94.2% on SPEC, reflecting limitations associated with its software implementation. The hardware counterpart, EA-LockStep, incurred geomean overheads of 31.2% for Parsec and 48.7% for SPEC, approximately 6.1x and 33.7x higher than MEEK, evidencing the performance-area benefits of MEEK.

[5]If there are scenarios where the opposite occurs (the big core holds locks, e.g., software mutexes, required by the little core), then deadlock results. This is impossible during standard re-execution, as the checker thread itself does not access memory and thus cannot take locks; only replaying the memory previously read when the main thread took out locks. However, there are scenarios where the checker thread inadvertently requires real memory reads in order to continue: specifically, if an instruction fault on the little core (e.g., because it has overtaken the big core, or the instruction was paged out before the little core reaches it) [38], the page-fault handling may require a lock held by the main thread, causing deadlock (Fig. 5 (a)).

[6]More generally, care needs to be taken if kernel operations are inside the sphere of replication: similar effects could happen were the scheduler blocked via a lock held by a waiting main thread, in turn blocking checkers.

[7]For Nzdc, compilation fails in gcc, omnetpp, xalancbmk, and freqmine.

Big Core	
Core	4-Width, OoO superscalar SonicBoom, @3.2GHz
Pipeline	128-Entry ROB,96-entry IQ, 32-entry LDQ/STQ, 128 Int/FP Phy Registers, 2 Int ALUs, 1 FP/Mult/Div ALU, 2 MEM, 1 Jump Unit, 1 CSR Unit
Branch Pred.	TAGE algorithm, 256-entry BTB, 32-entry RAS, 6 TAGE table with 2 - 64 bits history
Memory Hierarchy	
L1 ICache	32 KB, 4-way, 8 MSHRs
L1 DCache	32 KB, 4-way, 8 MSHRs
L2 Cache	512 KB, 8-way, 12 MSHRs
LLC	4 MB, 8-way, 8 MSHRs
Memory	16 GB DDR3 @1066Mhz, max 32 requests
Little Cores	
Cores	4 × In-order Rocket, 5-stage pipeline, @1.6GHz, 8-Unroll DIV, 3-stage FPU
LSL	4 KB, 5000 instruction time-out
L1 Cache	4 KB, 2-way for both I- and D-Cache

TABLE II: Hardware configurations evaluated.

B. Detection Latency

To examine the detection latency, we inject errors in the forwarded data from the F^2 connected to the big core, e.g., data and address of memory operations and architectural register data, simulating the hardware faults without disrupting the big core's normal execution. For each workload, 5,000 - 10,000 faults are randomly generated, and the density of detection latencies is shown in Fig. 7.

Each distribution features a long, but very thin, tail extending to the far right: the average detection latency is below 1 μs, while the worst-case latencies are 5 to 10 times higher, reaching up to 2.7 μs (in the ferret). Despite the randomness of the fault injection, the extremely high number of sample points ($> 100,000$, in total) suggests empirically that 3 μs is sufficient to cover over 99.9% of hardware faults, which is multiple orders of magnitude lower than the millisecond-level FTTI requirement for ASIL-D compliance.

C. Scalability

Fig. 8 presents the slowdown observed with varying numbers of little cores. Using two little cores to verify the Parsec execution results in a 54.9% slowdown (geomean), which remains significantly lower than that of Nzdc. With four little cores, the geomean overhead is reduced to 4.4%, with only the swaptions workload experiencing more than 5% overhead (as explained above). Scaling the system to six cores further decreases the geomean overhead to 0.3%, with all workloads having less than 1% overhead. The trend of slowdown exhibits a superlinear decline as the number of little cores increases, which suggests that adding little cores can effectively mitigate performance overhead even for more complex workloads in the future.

D. Microarchitectural Bottleneck Analysis

By detailed implementation, we identified the performance bottlenecks related to data collection, forwarding, and thread re-execution in the heterogeneous error detection process (Sec. III). Here, we provide an in-depth analysis of these bottlenecks and examine the effectiveness of our proposed microarchitecture.

Backpressure decomposition. Fig.9 illustrates the overhead breakdown in MEEK using 4 little cores while running Parsec, employing a full-featured AXI-Interconnect [44] and our F^2. As shown in Fig. 9, the AXI-Interconnect introduces a significant 16.7% overhead (geomean), evidencing that the 128-bit narrow bus, only handling one packet per cycle, is the primary bottleneck of the entire system, particularly when the little cores operating at a lower clock frequency. This also motivates the development of a dedicated data forwarding fabric with a wider data path and improved packets' scheduling to mitigate these limitations (Sec. III-B). By replacing the AXI-Interconnect with a 256-bit fabric that supports multiple-packet transmissions per cycle while preserving ordering using FSMs, we effectively reduce the total overhead of data collection and forwarding to less than 5%. Our F^2 design alleviates this bottleneck by enabling parallel data transmission and improved clock-domain crossing efficiency, thereby minimizing contention. With F^2, MEEK shifts from being a system limited by data forwarding to one that is computation-bound, with performance depending on the combined compute power of its checker threads.

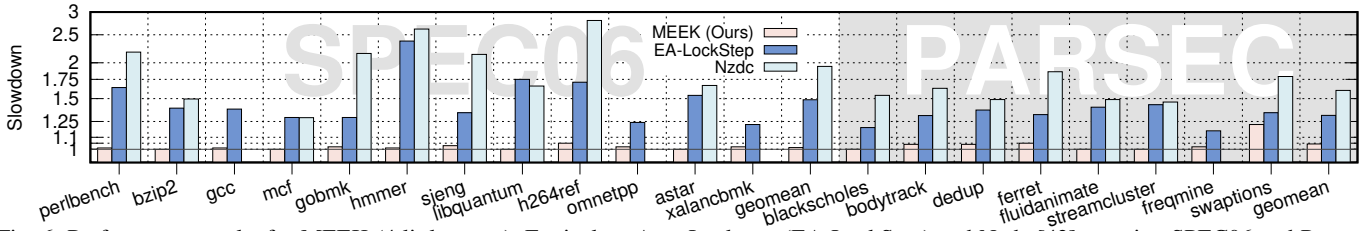


Fig. 6: Performance results for MEEK (4 little cores), Equivalent-Area Lockstep (EA-LockStep) and Nzdc [43], running SPEC06 and Parsec.

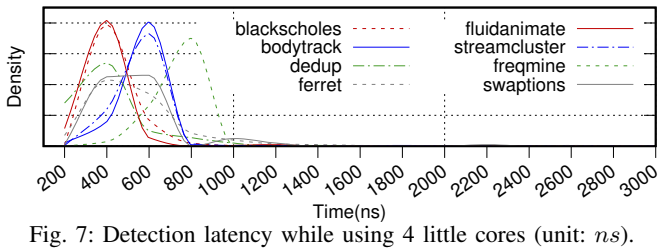


Fig. 7: Detection latency while using 4 little cores (unit: ns).

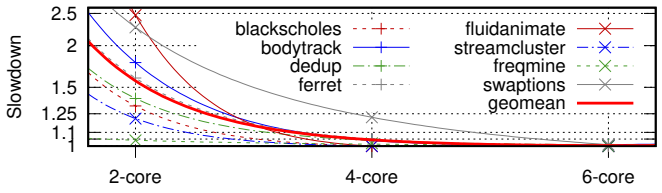


Fig. 8: Slowdown when using varying numbers of little cores.

Little core's performance-area efficiency. As discussed in Sec.III-C, we analyzed to understand performance gaps between the little and big cores and increased the size of bottlenecked components in the little cores. The evaluation shows that four optimized little cores can match the performance of six default little cores for the verification job. Given that our configurations increase the area of the little cores, we assess them using a normalized performance/area metric for fair comparison (Fig.10), revealing a 15.2% improvement (geomean) and demonstrating the performance-area efficiency of our design choice.

E. Hardware Overhead

We synthesized a physical implementation of MEEK with four little cores using TSMC 28nm PDKs [45]. The RTL is synthesized using Design Compiler (v2022.12), and the netlist is placed and routed via IC Compiler 2 (v2022.12), see Tab. III.

The BOOM's area is 2.811 mm^2, and each Rocket is 0.092 mm^2, excluding L1 D$ (since it is not required for re-execution [21], [22]). The total area for BOOM's data collecting and forwarding takes 0.122 mm^2, (4.3% of the BOOM): the DEU occupies 0.071 mm^2 and the F^2 consumes 0.051 mm^2. In summary, building MEEK with four Rocket upon a BOOM needs 0.726 mm^2 in total, i.e., 25.8% overhead.

F. Gap Analysis

We observe a discrepancy between the hardware overhead measured in MEEK and the estimation in previous high-level simulations [21]. While both work claims around 25% extra area is enough to enable heterogeneous parallel error detection for a Cortex-A57-level core[8], this is concluded based on different configurations: twelve Rocket cores in [21] versus four Rockets at double the frequency here.

Upon further analysis (Tab. III), several key factors were identified. First, BOOM is only 72.1% the size of an A57, likely due to differences in ISA complexity. This means that it is misleading to use Rocket to match the performance of an A57 in [21], as a little ARM core with similar performance would be larger than the RISC-V core. Second, our implementation required 17.9% more area per core than the synthesis results used in [21], where their L1 instruction cache configuration was also insufficient for large benchmarks like SPEC (also noted in [22]). Lastly, wrapper logic was previously ignored.

[8]IPCs of BOOM and A57 are similar, 0.95 and 0.97, measured using Parsec.

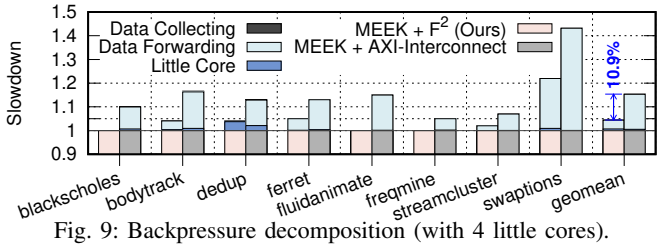


Fig. 9: Backpressure decomposition (with 4 little cores).

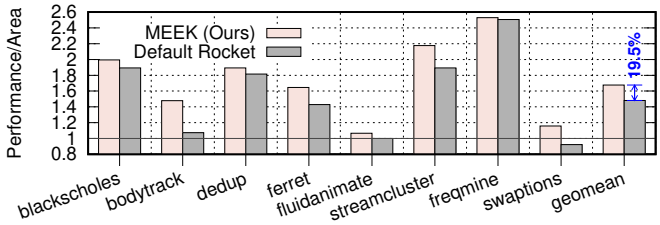


Fig. 10: Performance-area analysis of the little core.

		Big	Little		Big	Little
Core	Ours	BOOM	Rocket	DSN'18 [21]	Cortex-A57	Rocket
Number		1	4		1	12
Freq. (GHz)		3.2	2		3.2	1
Tech. (nm)		28	28		20	40
Area (mm^2)		2.811	0.092		2.050	0.160
@28nm		2.811	0.092		3.905	0.078
Wrapper (mm^2)		0.122	0.059		x	x
Overhead		**25.8%**			**24%**	

TABLE III: Hardware overhead in MEEK and DSN'18 [21], excluding L1 D$ in little cores (Grey shading indicates a key discrepancy).

Although the discrepancies in area estimation for individual modules may seem minor, the cumulative effect led to a significant overall mismatch between the simulated and real overhead, meaning we had the budget for only $\frac{1}{3}$ the little cores and so had to carefully optimize each one's performance (section V-D) rather than scaling the number of cores as was the perspective of the original papers.

VI. Conclusion

We have presented MEEK, a systematic implementation of heterogeneous parallel error detection, from the microarchitecture and ISA to the OS and programming model. While the typical architectural research employs abstract modeling (e.g., GEM5 [46]) for rapid prototyping, which is significantly faster than real implementations, this work shows the practical challenges of implementing even a well-modeled architecture, identifying bugs, performance bottlenecks, and inaccurate estimations that are invisible in high-level simulations, as well as opportunities for heavy simplification via codesign. We believe there is much new insight to be gained at this level of analysis for the movement of complex research concepts into full production.

VII. Acknowledgement

We appreciate the anonymous reviewers for their helpful feedback. This work is supported by the National Key Research and Development Program (Grant No. 2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086, 62204036) and the Basic Research Program of Jiangsu (Grants No. BK20243042).

REFERENCES

[1] C. Hernandez and J. Abella, "Timely error detection for effective recovery in light-lockstep automotive systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1718–1729, 2015.

[2] T. M. Austin, "Diva: A reliable substrate for deep submicron microarchitecture design," in *MICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture*. IEEE, 1999, pp. 196–207.

[3] B. F. Romanescu and D. J. Sorin, "Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 43–51.

[4] S. K. Reinhardt and S. S. Mukherjee, "Transient fault detection via simultaneous multithreading," in *Proceedings of the 27th annual international symposium on Computer architecture*, 2000, pp. 25–36.

[5] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson *et al.*, "Addressing failures in exascale computing," *The International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 129–173, 2014.

[6] I. ISO, "26262: Road vehicles-functional safety," 2018.

[7] L. Rierson, *Developing safety-critical software: a practical guide for aviation software and DO-178C compliance*. CRC Press, 2017.

[8] Z. Jiang, N. C. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 75–84.

[9] S. S. Mukherjee, M. Kontz, and S. K. Reinhardt, "Detailed design and evaluation of redundant multithreading alternatives," *ACM SIGARCH Computer Architecture News*, vol. 30, no. 2, pp. 99–110, 2002.

[10] P. Bernardi, R. Cantoro, S. De Luca, E. Sánchez, and A. Sansonetti, "Development flow for on-line core self-test of automotive microcontrollers," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 744–754, 2015.

[11] K. Batcher and C. Papachristou, "Instruction randomization self test for processor cores," in *Proceedings 17th IEEE VLSI Test Symposium (Cat. No. PR00146)*. IEEE, 1999, pp. 34–40.

[12] S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala, and S. A. Wender, "Predicting the number of fatal soft errors in los alamos national laboratory's asc q supercomputer," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 329–335, 2005.

[13] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (tcls) arm® cortex®-r5 processor for safety-critical and ultra-reliable applications," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*. IEEE, 2016, pp. 246–249.

[14] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The arm triple core lock-step (tcls) processor," *ACM Transactions on Computer Systems (TOCS)*, vol. 36, no. 3, pp. 1–30, 2019.

[15] B. Stolt, Y. Mittlefehldt, S. Dubey, G. Mittal, M. Lee, J. Friedrich, and E. Fluhr, "Design and implementation of the power6 microprocessor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 21–28, 2008.

[16] "Renesas: Rh850 micro-controller familiy," https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rh850.html.

[17] "Infineon: 32-bit TriCore Microcontroller," https://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-microcontroller/.

[18] K. Veeraraghavan, D. Lee, B. Wester, J. Ouyang, P. M. Chen, J. Flinn, and S. Narayanasamy, "Doubleplay: Parallelizing sequential logging and replay," *ACM Transactions on Computer Systems (TOCS)*, vol. 30, no. 1, pp. 1–24, 2012.

[19] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke, "The stagenet fabric for constructing resilient multicore systems," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2008, pp. 141–151.

[20] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques—part i: Fault diagnosis with model-based and signal-based approaches," *IEEE transactions on industrial electronics*, vol. 62, no. 6, pp. 3757–3767, 2015.

[21] S. Ainsworth and T. M. Jones, "Parallel error detection using heterogeneous cores," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 338–349.

[22] ——, "Paramedic: Heterogeneous parallel error correction," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 201–213.

[23] Á. B. de Oliveira, G. S. Rodrigues, F. L. Kastensmidt, N. Added, E. L. Macchione, V. A. Aguiar, N. H. Medina, and M. A. Silveira, "Lockstep dual-core arm a9: Implementation and resilience analysis under heavy ion-induced soft errors," *IEEE Transactions on Nuclear Science*, vol. 65, no. 8, pp. 1783–1790, 2018.

[24] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[25] K. Mitropoulou, V. Porpodas, and T. M. Jones, "Comet: communication-optimised multi-threaded error-detection technique," in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, ser. CASES '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2968455.2968508

[26] S. Ainsworth, L. Zoubritzky, A. Mycroft, and T. M. Jones, "Paradox: Eliminating voltage margins via heterogeneous fault tolerance," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 520–532.

[27] ARM, "big.little architecture," in *https://www.arm.com/technologies/big-little*, 2024.

[28] J. Doweck, W.-F. Kao, A. K.-y. Lu, J. Mandelblat, A. Rahatekar, L. Rappoport, E. Rotem, A. Yasin, and A. Yoaz, "Inside 6th-generation intel core: New microarchitecture code-named skylake," *IEEE Micro*, vol. 37, no. 2, pp. 52–62, 2017.

[29] B. Schroeder, E. Pinheiro, and W.-D. Weber, "Dram errors in the wild: a large-scale field study," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 193–204, 2009.

[30] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, "Understanding sources of inefficiency in general-purpose chips," in *Proceedings of the 37th annual international symposium on Computer architecture*, 2010, pp. 37–47.

[31] F. Restuccia, A. Biondi, M. Marinoni, G. Cicero, and G. Buttazzo, "Axi hyperconnect: A predictable, hypervisor-level interconnect for hardware accelerators in fpga soc," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[32] Z. Jiang, K. Yang, N. Fisher, I. Gray, N. C. Audsley, and Z. Dong, "Axi-ic^{RT}: Towards a real-time axi-interconnect for highly integrated socs," *IEEE Transactions on Computers*, vol. 72, no. 3, pp. 786–799, 2022.

[33] Z. Jiang, K. Yang, N. Audsley, N. Fisher, W. Shi, and Z. Dong, "Bluescale: a scalable memory architecture for predictable real-time computing on highly integrated socs," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1261–1266.

[34] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*, 2009, pp. 469–480.

[35] "IBM: Overview of constructors and destructors (C++ only)," https://www.ibm.com/docs/en/xl-c-and-cpp-aix/16.1?topic=only-overview-constructors-destructors-c.

[36] J. Aas, "Understanding the linux 2.6. 8.1 cpu scheduler," *Retrieved Oct*, vol. 16, pp. 1–38, 2005.

[37] R. Love, *Linux kernel development*. Pearson Education, 2010.

[38] D. P. Bovet and M. Cesati, *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.

[39] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020, pp. 1–7.

[40] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra *et al.*, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 29–42.

[41] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[42] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[43] M. Didehban and A. Shrivastava, "nzdc: A compiler technique for near zero silent data corruption," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[44] "Xilinx: AXI-Interconnect," https://www.xilinx.com/products/intellectual-property/axi_interconnect.html.

[45] "28nm PDKs," https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_28nm.

[46] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," vol. 39, no. 2. ACM New York, NY, USA, 2011, pp. 1–7.

Few-shot Learning on AMS Circuits and Its Application to Parasitic Capacitance Prediction

Shan Shen[1,2], Yibin Zhang[2], Hector Rodriguez Rodriguez[2], Wenjian Yu[2]
[1]*School of Microelectronics, Nanjing University of Science and Technology, Nanjing 210094, China*
[2]*Dept. Computer Science & Tech., BNRist, Tsinghua Univ., Beijing 100084, China*
Email: shanshen@njust.edu.cn, yu-wj@tsinghua.edu.cn

***Abstract*—Graph representation learning is a powerful method to extract features from graph-structured data, such as analog/mixed-signal (AMS) circuits. However, training deep learning models for AMS designs is severely limited by the scarcity of integrated circuit design data. In this work, we present CircuitGPS, a few-shot learning method for parasitic effect prediction in AMS circuits. The circuit netlist is represented as a heterogeneous graph, with the coupling capacitance modeled as a link. CircuitGPS is pre-trained on link prediction and fine-tuned on edge regression. The proposed method starts with a small-hop sampling technique that converts a link or a node into a subgraph. Then, the subgraph embeddings are learned with a hybrid graph Transformer. Additionally, CircuitGPS integrates a low-cost positional encoding that summarizes the positional and structural information of the sampled subgraph. CircuitGPS improves the accuracy of coupling existence by at least 20% and reduces the MAE of capacitance estimation by at least 0.067 compared to existing methods. Our method demonstrates strong inherent scalability, enabling direct application to diverse AMS circuit designs through zero-shot learning. Furthermore, the ablation studies provide valuable insights into graph models for representation learning.**

I. Introduction

Deep learning (DL) models rely on the availability of large-scale labeled datasets to train deep neural networks with millions of parameters. In many real-world scenarios, acquiring such extensive labeled datasets is impractical or impossible. For example, obtaining practical and real integrated circuit (IC) designs from the industry is notoriously difficult due to a combination of intellectual property concerns, security risks, and the complexity of the designs. IC designs are the result of extensive research and substantial investment. Strict intellectual property (IP) laws prevent the sharing of proprietary designs without proper authorization. Some IC designs are critical for defense and infrastructure, making them sensitive from a security perspective. Another reason for data scarcity in the IC field is the high cost of labeling data for tasks such as rare event prediction, layout optimization, or parasitic extraction. These tasks demand expert knowledge and the use of time-consuming EDA tools.

Consequently, there is a growing need for models that can learn effectively from limited amounts of design data.

This work is supported by the National Natural Science Foundation of China (NSFC) under grant (No. 62204141) and Beijing Natural Science Foundation (No. Z230002). W. Yu is the corresponding author. The code and the small datasets are available at https://github.com/ShenShan123/CirGPS.git

Few-shot learning (FSL) addresses this challenge by enabling models to generalize to new tasks or classes using only a small number of labeled examples [1]. Inspired by the human ability to learn new concepts from minimal information, FSL reduces the dependency on large datasets to make machine learning models more adaptable and efficient. This paradigm shows great potential in the EDA field [2].

In this work, we leverage several FSL techniques in the application of analog/mixed-signal (AMS) circuit design automation. AMS circuit design often demands substantial manual intervention. It relies heavily on manually selecting circuit topology and component sizes. In conventional AMS design workflows, circuit designers rely on pre-layout simulations for design optimization, followed by post-layout verification. However, as process technologies scale to smaller transistors and lower operating voltages, the reduced transistor driving capability exacerbates sensitivity to parasitic effects. For example, the coupling capacitance has become too significant to be overlooked in simulations, producing a substantial disparity between pre-layout and post-layout performance metrics [3], [4].

To address the challenges described above, we present a general, powerful, scalable circuit graph learning method (**CircuitGPS**) to perform various parasitic-related tasks in AMS circuits. The circuits are modeled as heterogeneous graphs, while the coupling effects are treated as links to be predicted. Inspired by SEAL [5], a small-hop subgraph sampling technique is proposed to sample target links from a large circuit graph. Then, we customize a graph Transformer (GT) [6] to extract the meta-embeddings at the subgraph level. The pre-trained meta-learner, combined with its learned embeddings, is subsequently adapted for the downstream task of coupling capacitance regression. Our major contributions are summarized as follows:

- A small-hop subgraph sampling technique is proposed to convert a link/node into a subgraph. This universal and scalable operation enables FSL by decoupling the target links/nodes from the specific circuit graph. Therefore, we can leverage more powerful and sophisticated DL models aimed at graph-level representation learning.
- A low-cost position encoding, double-anchor shortest path distance (**DSPD**), is defined to measure the relative position w.r.t the anchors of the subgraph. DSPD improves the model expressivity and leads to better

generalization.

- A powerful framework, **CircuitGPS**, is built on the top of the GraphGPS framework [6], which can perform multiple tasks on large circuit graphs. The framework was complemented with a fine-tuning strategy for the downstream capacitance regression.
- Through our ablation studies, we summarize two key insights into DL graph models trained on the AMS circuits: (1) involving the original node features during model training for link prediction will reduce generalization of the graph model; (2) classic message-passing graph neural networks (MPNNs) can achieve highly competitive performance, even outperforming GTs.

II. Background

Few-shot learning. FSL can be formally defined as a machine learning problem where a model must quickly adapt to a new task using only a few training examples or "shots" per class [7]. Typically, FSL involves a "meta-learning" phase, where the model learns how to learn across a variety of tasks, and an "adaptation" phase, where it applies this knowledge to a previously unseen task with limited data.

Subgraph sampling. Zhang and Chen [8] first extracted local enclosing subgraphs around links as the training data, and used a fully-connected neural network to learn which enclosing subgraphs correspond to link existence, which is called Weisfeiler-Lehman Neural Machine. The enclosing subgraph for a node pair (m, n) is the subgraph induced from the network by the union of the neighbors of m and n up to h hops. These enclosing subgraphs are very informative for link prediction – all first-order heuristics such as common neighbors can be directly calculated from the 1-hop enclosing subgraphs. However, it is shown that high-order heuristics often have much better performance than first and second-order ones [9], but the time and memory consumption are unsuitable for most practical applications. In work [5], Zhang et al. found that most high-order heuristics can be unified by a γ-decaying theory, and that even a small h can safely learn good high-order features.

Graph Transformers (GT). Considering the great successes of Transformers in natural language processing [10] and also in computer vision [11], it is natural to study their applicability in the graph domain. A fully-connected GT [12] was first introduced by Dwivedi et al. SAN [13] further improved this work by implementing invariant aggregation of the positional encoding (PE) Laplacian eigenvectors and conditional attention for the real and virtual edges of a graph. Concurrently, Graphormer [14] used pair-wise graph distances (or 3D distances) to define relative positional encodings, with outstanding success on large molecular benchmarks.

Positional and structural encodings. There have been many recent works on PE and structural encoding (SE), notably on Laplacian PE (LapPE) [12], [13], shortest-path-distance (SPD) [14], node degree centrality [14], random-walk SE (RWSE) [15], and more. Some works also proposed dedicated networks to learn the PE/SE from an initial encoding [13], [15]. To better understand the different PE/SEs and the contribution of each work, Rampasek et al. [6] categorized them into different types and examined their effect. In most cases, PE/SE serves as a soft bias, meaning they are simply provided as input features. However, in some cases, they can be used to direct messages [16] or create bridges between distant nodes [17].

Parasitic prediction. The DL models are gaining increasing attention and are being leveraged for predicting parasitic effects in AMS circuits. ParaGraph [18] converts circuit schematics into graphs and utilizes different types of MPNN layers to predict net capacitance and device layout parameters. It includes the ensemble modeling technique to improve the prediction accuracy via training three different sub-models to predict different capacitance magnitudes. In work [19], Shen et al. developed a DL-based multi-expert model (DLPL-Cap) to predict parasitic capacitance in the pre-layout stage. The model combines a GNN router and five expert regressors, effectively managing the imbalance of the net parasitic capacitance in SRAM circuits. They achieved state-of-the-art results on capacitance regression.

However, the aforementioned research [18], [19] treats the coupling capacitance as a part of the lumped-in capacitance. Furthermore, these ensemble models have very limited generalization and transferability when making predictions on unseen designs, as they do not employ FSL techniques (see Section IV for more details).

III. Methods

The proposed method contains four steps, as shown in Fig. 2: (1) AMS netlist conversion, (2) enclosing subgraph extraction, (3) position encoding, and (4) model pre-training/fine-tuning. CircuitGPS is first pre-trained on a large dataset for link prediction, taking local enclosing subgraphs around links as the input, and outputting the link existence likelihood. After the pre-training, we can obtain a "meta-learner", which can be directly tested on other unseen circuits with zero-shot or be further used as an initial model for other downstream tasks. For example, for the latter usage, we can select a regression task where the model predicts the coupling capacitance between the given nodes. This demonstrates that the meta-learner can be quickly adapted to other tasks.

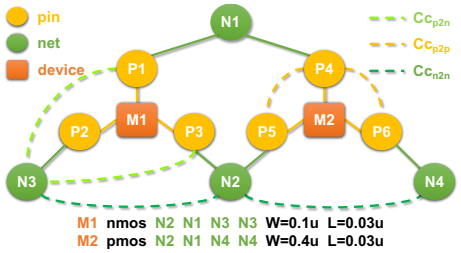


Fig. 1 A buffer netlist is converted to a heterogeneous graph composed of net, pin, and device nodes, where coupling effects are modeled as target links (dash lines).

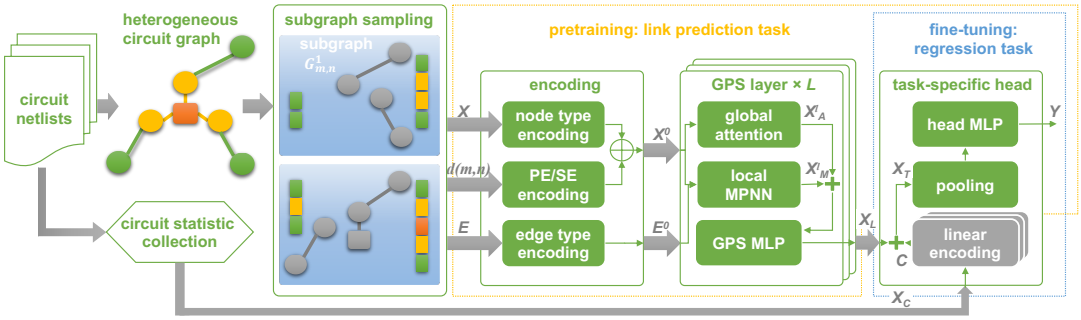


Fig. 2 **CircuitGPS** is based on a parallel MPNN+Transformer layer. The overall workflow includes: AMS netlist conversion, enclosing subgraph extraction, position encoding, and model pre-training/fine-tuning.

A. AMS Netlist Graph

The schematic netlist of an entire AMS circuit is modeled as a large heterogeneous graph $G = (V, E)$ with node attribute $\mathbf{X} = \{x_0, x_1, \cdots x_{N-1}\}^T$ and edge attribute $\mathbf{E} = \{e_0, e_1, \cdots e_{N_E-1}\}^T$. Fig. 1 shows an example of a buffer circuit containing three types of nodes. Green circles represent nets with $x_i = 0$, connecting to devices; orange squares represent instances of the transistor device with $x_i = 1$, which may also represent other devices, such as capacitors, resistors, or diodes; yellow circles represent pins of devices with $x_i = 2$. Edges describe the topology of a schematic, which can be "device-to-pin" connections with $e_i = 0$ or "net-to-pin" connections with $e_i = 1$. Links are the targets to be predicted, which are composed of "pin-to-net" coupling with $e_i = 2$, "pin-to-pin" coupling with $e_i = 3$, and "net-to-net" coupling with $e_i = 4$. Note that the terminology "link" is different from "edge" in this work, as "link" is our ground truth that can only be extracted from the post-layout netlist. We set all the edges and links to be undirected.

B. Subgraph Sampling

Here we describe our enclosing subgraph sampling technique. The sampling does not restrict the learned features to be in some particular form but instead learns general graph-level structural features for both node- and link-level targets. The sampling procedure contains three steps: (1) negative link generation, (2) enclosing subgraph extraction, and (3) node/edge attribute matrix construction.

Negative link generation. For a graph $G = (V, E)$ with an $N \times N$ adjacency matrix $\mathbf{A}$, a positive link (observed) exists from node i to node j if $\mathbf{A}_{i,j} = 1$, while a negative link (unobserved) corresponds to $\mathbf{A}_{i,j} = 0$ or $\mathbf{A}_{i,j} = -1$. Suppose that a graph has nodes $V = \{i, j, m, n\}$ and edges $E = \{(i, j), (m, n)\}$, where i, m are the source and j, n are the destination. For each type of positive link, the structural negative links are formed by permuting the source/destination nodes in E, resulting in $E_N = \{(i, n), (m, j)\}$. The negative and positive links have the same edge type, which means the sources/destinations of these links also share the same node types. In this case, $x_i = x_m$ and $x_j = x_n$.

In AMS circuits, a positive link refers to the presence of a coupling effect between the net/pin node pair, while a negative link indicates the absence of such an effect. The positive and negative links are assigned the labels '1' and '0', respectively. As depicted in Fig. 1, the target links have three types. We found that the "pin-to-net" links E^{p2n} often constitute the majority, whereas "net-to-net" links E^{n2n} are relatively fewer. The severe class imbalance can degrade the pre-trained model's performance [19]. To mitigate this, we balance the training dataset by randomly sampling $|E^{n2n}|$ instances from each link type.

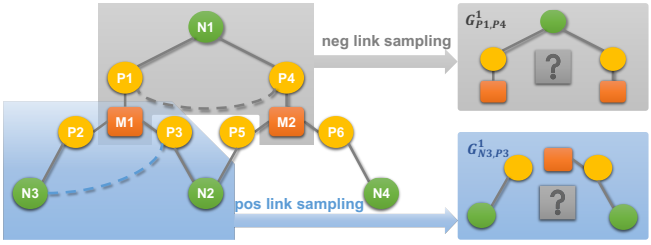


Fig. 3 One-hop enclosing subgraph sampling.

Enclosing subgraph extraction. Here we follow the definition of enclosing subgraph in SEAL [5].

Definition 1: **(Enclosing subgraph)** For a graph $G = (V, E)$, given two nodes $m, n \in V$, the h-hop enclosing subgraph for (m, n) is the subgraph $G^h_{m,n}$ induced from G by the set of nodes $\{i \mid d(i, m) \leq h \bigcup d(i, n) \leq h\}$.

The enclosing subgraph describes the "h-hop surrounding environment" of (m, n). $G^h_{m,n}$ contains all h-hop neighbors of m and n. Since the circuit graphs could be extremely large (see Table IV for details), we found that the 1-hop sampling achieves the best balance between performance and efficiency. This choice is also justified by the γ-decaying theory in SEAL, which supports the use of a small h to learn good high-order features. Figure 3 shows an example of the extracted subgraph $G^1_{m,n}$.

Attribute matrix construction. A MPNN typically takes $(\mathbf{A}, \mathbf{X})$ as input. Due to the heterogeneity of subgraphs, the node types form an attribute matrix $\mathbf{X} \in \{0, 1, 2\}^{N \times 1}$. Then, $\mathbf{X}$ is fed to the node-type encoder of CircuitGPS to learn embeddings $\texttt{Embed}(\mathbf{X}) \in \mathbb{R}^{N \times d_0}$. The model also considers edge attributes, where another edge-type encoder generates edge embeddings $\texttt{Embed}(\mathbf{E}) \in \mathbb{R}^{N_E \times d_0}$.

The aforementioned attributes are critical for pre-training

TABLE I Dimension Definition in $\mathbf{X}_C$.

Node Type	Dim.	Definition
$x_i = 0$ for net node	0	# of connected transistors
	1	# of connected gate terminals
	2	# of connected source/drain terminals
	3	# of connected base terminals
	4	Total width of connected transistor
	5	Total length of connected transistor
	6	# of connected capacitors
	7	Total length of connected capacitors
	8	Total # of connected capacitor fingers
	9	# of connected resistors
	10	Total width of connected resistors
	11	Total length of connected resistors
	12	# of connected ports
$x_i = 1$ for device node	0	Multiplier of transistors
	1	Length of the transistor
	2	Width of the transistor
	3	Multiplier of connected resistors
	4	Length of resistor
	5	Width of resistor
	6	Multiplier of connected capacitor
	7	Length of capacitor
	8	# of capacitor fingers
	9	# of ports in the device instance
	10	Type code of the device instance
$x_i = 2$ for pin node	0	Pin types (G/D/S/B for MOS)

a link prediction model. However, for the downstream regression, the coupling capacitance is also related to other information, such as the geometric size of devices and the number of devices connecting to a net. Thus, we collect more detailed statistics for each type of node, including the design parameters in AMS circuits. We pad these statistics into another node attribute matrix $\mathbf{X}_C \in \mathbb{R}^{N \times d_C}$ that is input into the task-specific head of CircuitGPS. Table I lists the definition of each dimension in $\mathbf{X}_C$ for different node types.

C. Positional Encoding

Positional encodings (PEs) are proposed to estimate the position in space of a given node within the graph. Hence, when two nodes are close to each other within a graph, their PE should also be close. A common approach is to compute the pair-wise distance between each pair of nodes or their eigenvectors in the entire graph as proposed in [13], [14], but this is not compatible with linear Transformers (such as Performer [20]) as it requires to materialize the entire attention matrix. A better fitting solution is to use the eigenvectors of the graph Laplacian or their gradient [13], [16], [21]. However, calculating those PEs is both time- and resource-consuming due to the large scale of subgraphs sampled from AMS circuits. For this reason, a more computationally efficient PE is demanded by our application.

The proposed PE, named double anchor shortest path distance (DSPD), is a relative PE according to the PE/SE categories proposed in [6], [22]. In each subgraph $G^1_{m,n}$ with anchors m and n, we calculate the shortest path distance $(d(i,m), d(i,n))$ to the two anchors for each node i. The DSPD is inspired by the double-radius node labeling (DRNL) in SEAL, where the authors iteratively assign larger labels to nodes with a larger distance (radius) with respect to both anchor nodes. The labeling principle is a perfect hashing function that allows fast closed-form computations. Differently, the DSPD encoding does not compute a unique label; it directly stores the distance vector $(d(i,m), d(i,n))$. The PE encoder then uses the distances to generate $\mathbf{D}_0$ and $\mathbf{D}_1$. We concatenate them with the node-type embeddings to form the final node attribute matrix,

$$\mathbf{X}^0 = \mathbf{D}_0 \oplus \mathbf{D}_1 \oplus \texttt{Embed}(\mathbf{X}), \tag{1}$$

which is the input of the first GPS layer.

TABLE II Comparison of Different PEs in Link Prediction.

PE	Acc.	F1	AUC	Time/G (s)
w/o PE	0.8867	0.9120	0.9393	N/A
$\mathbf{X}_C$	0.9066	0.9261	0.9629	N/A
DRNL	0.9505	0.9640	0.9698	**0.0170**
RWSE	0.8931	0.9255	0.8612	0.1296
LapPE	0.9561	0.9680	0.9697	0.1934
DSPD	**0.9618**	**0.9720**	**0.9774**	0.0173

Table II shows the prediction accuracy of CircuitGPS on link prediction with different PEs, including circuit statistic matrix $\mathbf{X}_C$, RWSE proposed by [15], LapPE used by [12], [13], DRNL proposed by SEAL, and our DSPD. We take the AMS design SSRAM [23] as a small training dataset and the circuit DIGITAL_CLK_GEN as a test dataset to evaluate CircuitGPS in a zero-shot learning setting. The proposed DSPD demonstrates the highest accuracy while ensuring computational efficiency.

Observation 1: Taking the circuit statistic matrix $\mathbf{X}_C$ as a positional encoding reduces the model performance compared to employing other PEs.

This counterintuitive phenomenon was also reported by Shi et al. [24]. In their work, removing original node features of neighbor nodes led to slight performance gains for SEAL on multiple link prediction datasets. While seemingly paradoxical, this result is explicable: in link prediction tasks, neighbor node features may introduce task-irrelevant noise. For instance, in our context, the specific device types connected to a target net are irrelevant for predicting coupling effects between nodes. This critical observation motivated our architectural redesign, transitioning from GraphGPS to the simplified CircuitGPS framework.

D. Hybrid Graph Transformer

The hybrid GT model used in this work is built from a general, powerful, and scalable architecture named GraphGPS [6]. Each GPS layer is a parallel MPNN+Transformer layer, as shown in Fig. 2. The update function is

$$\mathbf{X}^{\ell+1}, \mathbf{E}^{\ell+1} = \texttt{GPS}^{\ell}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}\right), \tag{2}$$

and the hidden features are computed as

$$\mathbf{X}_M^{\ell+1}, \mathbf{E}^{\ell+1} = \texttt{MPNN}_e^{\ell}\left(\mathbf{X}^{\ell}, \mathbf{E}^{\ell}, \mathbf{A}\right), \tag{3}$$

$$\mathbf{X}_A^{\ell+1} = \texttt{GlobalAttn}^{\ell}\left(\mathbf{X}^{\ell}\right), \tag{4}$$

$$\mathbf{X}^{\ell+1} = \texttt{MLP}^{\ell}\left(\mathbf{X}_M^{\ell+1} + \mathbf{X}_A^{\ell+1}\right), \tag{5}$$

TABLE III Ablation Study of Different GPS Layer Configuration on Link Prediction (**best**, second best).

MPNN	Attention	Acc.	F1	AUC	Time(s)	#Param.
None	Performer	0.9458	0.9602	0.9668	1663.0	762,390
None	Transformer	0.9456	0.9601	0.9187	3490.0	778,833
GatedGCN	Performer	0.9618	0.9720	0.9774	1446.1	752,785
GatedGCN	Transformer	**0.97011**	**0.9780**	**0.9980**	2832.9	540,337
GatedGCN	None	0.9693	0.9775	0.9848	**965.6**	724,854

where $\mathbf{A} \in \mathbb{R}^{N\times N}$ is the adjacency matrix; $\mathbf{X}^\ell \in \mathbb{R}^{N\times d_\ell}$, $\mathbf{E}^\ell \in \mathbb{R}^{E\times d_\ell}$ are the d_ℓ-dimensional node and edge attributes, respectively; $\texttt{MPNN}_e^\ell$ and $\texttt{GlobalAttn}^\ell$ are instances of an MPNN with edge attributes and a global attention mechanism, respectively; $\texttt{MLP}^\ell$ is a 2-layer MLP block. After each functional block (an `MPNN` layer, a `GlobalAttn` layer, or an `MLP`), we apply residual connections followed by batch normalization (BN), which are omitted in (5) for clarity. Edge attributes are only passed to the `MPNN` layer. Due to the scalability of GPS, both `MPNN` and `GlobalAttn` layers are modular, where `MPNN` can be any function that acts on a local neighborhood and `GlobalAttn` can be any fully connected layer.

Based on Observation 1, we customize the GraphGPS with a task-specific head. We add two linear projection layers and an embedding layer in the head to project circuit statistics $\mathbf{X}_C$ (Table I) into a unified attribute matrix $\mathbf{C} \in \mathbb{R}^{N\times d_L}$. For a node i in the subgraph with node type x_i and circuit statistic x_C^i, c_i is produced by

$$c_i = \begin{cases} \texttt{Linear}(x_C^i), & \text{if } x_i = 0; \\ \texttt{Linear}(x_C^i), & \text{if } x_i = 1; \\ \texttt{Embed}(x_C^i), & \text{if } x_i = 2. \end{cases} \tag{6}$$

Finally, the hidden input of the task-specific head `MLP` is

$$\mathbf{X}_H = \texttt{Pool}(\mathbf{X}^L + \mathbf{C}). \tag{7}$$

To find a better layer architecture, we produced an ablation study of different layer types on a small training set as shown in Table III. The alternatives are a global attention-based Transformer, a linear Transformer (Performer [20]), a pure MPNN (GatedGCN [25]), and their combinations.

Observation 2: Classic MPNN models achieve strong performance, matching or even exceeding that of pure Transformers or MPNN+Transformers.

This observation coincides with the results reported by Luo et al. in their recent work [26]. According to their findings, this strong performance is driven by normalization, dropout, and residual connections. All these features have been integrated into CircuitGPS and can be adjusted with the hyperparameters in configuration files. Beyond excellent performance, MPNN requires lower training overhead and converges more easily.

E. Model Fine-tuning

Models trained with few-shot learning techniques are often better at generalizing to new tasks or classes, as they focus on capturing transferable knowledge rather than memorizing specific data patterns. The pre-trained CircuitGPS on link prediction aims to learn an informational representation of subgraphs. Fine-tuning can be implemented in two ways: (1) freezing the encoders and GPS layers while training only the task-specific head, which significantly accelerates the convergence, or (2) continuing to train all parameters in CircuitGPS using the pre-trained ones as initialization, further improving edge regression accuracy (Table VI).

IV. Experiments

Our implementation is based on PyG [27], GraphGym [28], and GraphGPS [6] modules. All experiments were run in a shared computing cluster. The cluster has 40 Intel Xeon Silver CPUs with 128GB of memory and four Nvidia 4090 (24GB) GPUs. Each experiment was executed using four to six CPUs, a single GPU, and 128 GB of RAM.

Circuit statistics were obtained by converting the netlist to a graph. Before the enclosing subgraph sampling, we followed the setup of SEAL [5], where both the positive edges and the negative edges were injected into the original circuit graph. The sampling procedure was implemented with the multiprocessing package to reduce the sampling time overhead. We prepared the full schematic netlists to extract the graph and features, and the post-layout netlists (SPF files) to collect the ground-truth labels and targets. The configuration files for experiments in this section can be found in our Git repository.

A. Dataset

Table IV summarizes the AMS designs used as datasets in this work. All designs are under 28nm CMOS technology. SSRAM [23] is a small design with high energy efficiency, consisting of standard digital cells and static random access memory (SRAM) arrays. ULTRA8T SRAM [29] is a multi-voltage design that contains large analog modules and SRAM arrays. SANDWICH-RAM [30] is evenly divided between digital circuits for computing and SRAM arrays for storage, forming a sandwich-like structure. Due to the large scale of the datasets, training with all coupling links results in unacceptable time overhead. The redundancy of capacitance also leads to inefficient training [19]. Thus, we only sample a fraction of the links to reduce the training time per epoch, as shown in the "#links" column of Table IV.

In terms of test data sets, DIGITAL_CLK_GEN is composed of digital cells and SRAM columns and is used to generate the internal clock in SRAM. It is the most challenging test case of our experiments. TIMING_CONTROL consists of standard digital cells to produce control signals for the SRAM module. ARRAY_128_32 is a single 128-row 32-column SRAM array.

TABLE IV AMS Circuit Dataset Statistics.

Split	Dataset	N	N_E	#Links	$N/G^1_{m,n}$	$N_E/G^1_{m,n}$
Train	SSRAM	87K	134K	131K	153	917
	ULTRA8T	3.5M	13.4M	166K	257	1,476
	SANDWICH-RAM	4.3M	13.3M	154K	472	2,540
Test	DIGITAL_CLK_GEN	17K	36K	4K	417	2,403
	TIMING_CONTROL	18K	44K	5K	59	387
	ARRAY_128_32	144K	352K	110K	150	803

TABLE V Accuracy Comparison of Different Methods on Link Prediction.

Dataset	DIGITAL_CLK_GEN			TIMING_CONTROL			ARRAY_128_32		
Metrics	Acc.	F1	AUC	Acc.	F1	AUC	Acc.	F1	AUC
ParaGraph	0.768	0.847	0.870	0.754	0.841	0.865	0.720	0.776	0.823
DLPL-Cap	0.761	0.841	0.864	0.750	0.839	0.865	0.756	0.832	0.825
CircuitGPS	**0.972**	**0.979**	**0.992**	**0.989**	**0.992**	**0.998**	**0.980**	**0.985**	**0.999**

None of the test datasets are part of the training datasets, and we ensure that all test data is not visible during training and validation. Thus, all results in this work are reported from zero-shot learning.

B. Link Prediction Task

In this subsection, we show the prediction performance of CircuitGPS on the link prediction task. We adapted the two existing MPNN-based models, ParaGraph [18] and DLPL-Cap [19], to our coupling prediction task for comparison purposes. These two models didn't involve any sampling technique or PE, and they directly used the entire graphs and the circuit statistic matrix $\mathbf{X}_C$ as input. We tuned the model hyperparameters to improve their performance. Table V lists the performance comparison. After combining enclosing subgraph sampling and DSPD encoding, CircuitGPS has demonstrated a huge improvement in terms of all performance metrics, improving accuracy by at least 20%.

C. Edge Regression Task

The edge regression task is to predict the exact coupling capacitance of the given node pair. The negative links injected during pre-training were assigned zero capacitance. As capacitance values spanned several magnitudes, we only kept the nets with the capacitance value $1^{-21}F \leq \boldsymbol{y}_C \leq 1^{-15}F$. Then we normalized the statistic attribute matrix $\mathbf{X_C}$ and the capacitance values to $[0, 1]$ to avoid numerical instability. As Table VI lists, CircuitGPS reduces the MAE of capacitance estimation by at least 0.067 compared to other methods.

We also conducted an ablation study of different GPS layer configurations for edge regression on the small training set SSRAM and the test dataset DIGITAL_CLK_GEN. Table VII lists the results. Again, the classic GatedGCN performs strongly, demonstrating Observation 2. Pure Transformer layers have the worst performance due to the limited number of parameters that our computing platform can handle.

D. Extending Study on Node Regression

Beyond link-level tasks, CircuitGPS can also fulfill node-level tasks. We trained CircuitGPS to predict the ground parasitic capacitance on each net/pin. In this task, we didn't involve any negative link injection, and a 2-hop subgraph sampling was adopted to obtain more neighbors around the singular anchor node. The DSPD became identical distances $\mathbf{D}_0 = \mathbf{D}_1$. The architecture remained the same as that in the link-level task. Table VIII shows the predicted errors. CircuitGPS achieves the best performance compared to the other two models in the node regression task. DLPL-Cap has the largest prediction error for the test dataset. This method first learns from a node classification task and then trains a specific regressor for each node class. Therefore, it has data-sensitive characteristics that limit generalization.

Finally, we conducted the circuit simulations using SPICE to validate the predicted capacitance of CircuitGPS (without involving any parasitic resistance). Simulated energy consumption can further demonstrate the effectiveness of CircuitGPS, as this metric is primarily determined by parasitic capacitance in AMS circuits. In Fig. 4, the mean absolute percentage error of three test cases is 14.5%.

TABLE VI Error Comparison of Different Methods on Edge Regression.

Dataset	DIGITAL_CLK_GEN			TIMING_CONTROL			ARRAY_128_32		
Metrics	MAE↓	RMSE↓	R^2	MAE↓	RMSE↓	R^2	MAE↓	RMSE↓	R^2
ParaGraph	0.153	0.212	0.470	0.154	0.214	0.590	0.181	0.260	0.211
DLPL-Cap	0.160	0.223	0.414	0.157	0.217	0.579	0.176	0.239	0.331
CircuitGPS	0.083	0.130	0.801	0.043	0.097	0.915	0.048	0.120	0.831
CircuitGPS$_{\text{head-ft}}$	0.086	0.125	0.816	0.085	0.131	0.847	0.075	0.120	0.831
CircuitGPS$_{\text{all-ft}}$	**0.072**	**0.120**	**0.833**	**0.042**	**0.093**	**0.923**	**0.040**	**0.074**	**0.936**

TABLE VII Ablation Study of Different GPS Layer Configuration on Edge Regression.

MPNN	Attention	MAE↓	RMSE↓	R^2	Time(s)	#Param.
None	Performer	0.0854	0.1439	0.7563	1437.3	736,871
None	Transformer	0.1051	0.1502	0.7351	2203.6	480,167
GatedGCN	Performer	**0.0705**	0.12974	0.8019	2667.9	751,311
GatedGCN	Transformer	0.0772	0.1358	0.7831	4765.2	506,703
GatedGCN	None	0.0718	**0.1233**	**0.8212**	**931.5**	723,380

TABLE VIII Error Comparison of Different Methods on Node Regression.

Dataset	DIGITAL_CLK_GEN			TIMING_CONTROL			ARRAY_128_32		
Metrics	MAE↓	RMSE↓	R^2	MAE↓	RMSE↓	R^2	MAE↓	RMSE↓	R^2
ParaGraph	0.101	0.144	0.313	0.112	0.154	0.462	0.114	0.174	0.002
DLPL-Cap	0.137	0.208	0.364	0.096	0.137	0.379	0.097	0.136	0.390
CircuitGPS	**0.072**	**0.104**	**0.643**	**0.088**	**0.132**	**0.602**	**0.078**	**0.101**	**0.637**

V. Conclusion

This work introduces CircuitGPS, a few-shot learning framework designed to estimate parasitic effects in AMS circuits. The method incorporates subgraph sampling, a computationally efficient PE, and a customized GT. The robust generalization and transferability of CircuitGPS make it adaptable to a variety of tasks, including link prediction (for coupling effect analysis), edge regression (to estimate coupling capacitance), and node regression (for ground capacitance estimation).

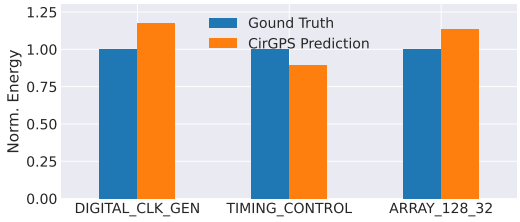


Fig. 4 The simulated energy consumption with ground truth parasitic capacitance vs. predictions of CircuitGPS.

REFERENCES

[1] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM computing surveys (csur)*, vol. 53, no. 3, pp. 1–34, 2020.

[2] K. Hakhamaneshi, M. Nassar, M. Phielipp, P. Abbeel, and V. Stojanovic, "Pretraining graph neural networks for few-shot analog circuit modeling and design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 2163–2173, 2022.

[3] W. Yu and X. Wang, *Advanced field-solver techniques for RC extraction of integrated circuits*. Springer, 2014.

[4] W. Yu, C. Hu, and W. Zhang, "Variational capacitance extraction of on-chip interconnects based on continuous surface model," in *Proceedings of the 46th Annual Design Automation Conference*, 2009, pp. 758–763.

[5] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[6] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a General, Powerful, Scalable Graph Transformer," *Advances in Neural Information Processing Systems*, vol. 35, 2022.

[7] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *International conference on learning representations*, 2017.

[8] M. Zhang and Y. Chen, "Weisfeiler-lehman neural machine for link prediction," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 575–583.

[9] L. Lü and T. Zhou, "Link prediction in complex networks: A survey," *Physica A: statistical mechanics and its applications*, vol. 390, no. 6, pp. 1150–1170, 2011.

[10] K. S. Kalyan, A. Rajasekharan, and S. Sangeetha, "Ammus: A survey of transformer-based pretrained models in natural language processing," *arXiv:2108.05542*, 2021.

[11] K. Han, Y. Wang, H. Chen, X. Chen, J. Guo, Z. Liu, Y. Tang, A. Xiao, C. Xu, Y. Xu *et al.*, "A survey on vision transformer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[12] V. P. Dwivedi and X. Bresson, "A generalization of transformer networks to graphs," *arXiv:2012.09699*, 2020.

[13] D. Kreuzer, D. Beaini, W. L. Hamilton, V. Létourneau, and P. Tossou, "Rethinking graph transformers with spectral attention," in *Advances in Neural Information Processing Systems*, 2021.

[14] C. Ying, T. Cai, S. Luo, S. Zheng, G. Ke, D. He, Y. Shen, and T.-Y. Liu, "Do transformers really perform badly for graph representation?" in *Advances in Neural Information Processing Systems*, 2021.

[15] V. P. Dwivedi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, "Graph neural networks with learnable structural and positional representations," in *International Conference on Learning Representations*, 2022.

[16] D. Beaini, S. Passaro, V. Létourneau, W. Hamilton, G. Corso, and P. Liò, "Directional graph networks," in *International Conference on Machine Learning*. PMLR, 2021, pp. 748–758.

[17] J. Topping, F. Di Giovanni, B. P. Chamberlain, X. Dong, and M. M. Bronstein, "Understanding over-squashing and bottlenecks on graphs via curvature," *arXiv:2111.14522*, 2021.

[18] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proc. DAC*, 2020, pp. 1–6.

[19] S. Shen, D. Yang, Y. Xie, C. Pei, B. Yu, and W. Yu, "Deep-learning-based pre-layout parasitic capacitance prediction on sram designs," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, 2024, pp. 440–445.

[20] K. M. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Q. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," in *International Conference on Learning Representations*, 2020.

[21] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking graph neural networks," *arXiv:2003.00982*, 2020.

[22] M. Black, Z. Wan, G. Mishne, A. Nayyeri, and Y. Wang, "Comparing graph transformers via positional encodings," *arXiv preprint arXiv:2402.14202*, 2024.

[23] S. Shen, T. Shao, X. Shang, Y. Guo, M. Ling, J. Yang, and L. Shi, "TS cache: A fast cache with timing-speculation mechanism under low supply voltages," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 252–262, 2019.

[24] L. Shi, B. Hu, D. Zhao, J. He, Z. Zhang, and J. Zhou, "Structural information enhanced graph representation for link prediction," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 13, 2024, pp. 14 964–14 972.

[25] X. Bresson and T. Laurent, "Residual gated graph convnets," *arXiv preprint arXiv:1711.07553*, 2017.

[26] Y. Luo, L. Shi, and X.-M. Wu, "Classic gnns are strong baselines: Reassessing gnns for node classification," *arXiv preprint arXiv:2406.08993*, 2024.

[27] M. Fey and J. E. Lenssen, "Fast graph representation learning with PyTorch Geometric," in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[28] J. You, R. Ying, and J. Leskovec, "Design space for graph neural networks," in *Advances in Neural Information Processing Systems*, 2020.

[29] S. Shen, H. Xu, Y. Zhou, M. Ling, and W. Yu, "Ultra8t: A sub-threshold 8t sram with leakage detection," *Integration*, vol. 98, p. 102233, 2024.

[30] J. Yang, Y. Kong, Z. Wang, Y. Liu, B. Wang, S. Yin, and L. Shi, "24.4 sandwich-RAM: An energy-efficient in-memory BWN architecture with pulse-width modulation," in *Proc. Int. Solid-State Circuits Conf. (ISSCC)*, 2019, pp. 394–396.

FLAG: An FPGA-Based System for Low-Latency GNN Inference Service Using Vector Quantization

Yunki Han, Taehwan Kim, Jiwan Kim, Seohye Ha, Lee-Sup Kim
School of Electrical Engineering, KAIST
Daejeon, Republic of Korea
{yunki.han, th.kim, jiwan_kim, seohye, leesup}@kaist.ac.kr

***Abstract*—Enabling real-time GNN inference services requires low end-to-end latency to meet service level agreements. However, intensive preparation steps and the neighborhood explosion problem pose significant challenges to efficient GNN inference serving. In this paper, we propose FLAG, an FPGA-based GNN inference serving system using vector quantization. To reduce preparation overhead, we introduce offline preprocessing to precompute and compress hidden embeddings for serving. A dedicated FPGA accelerator leverages the precomputed data to enable lightweight aggregation. As a result, FLAG achieves average speedups of 154×, 176×, and 333× on three GNN models compared to the baseline system.**

***Index Terms*—Graph Neural Network, Hardware Accelerator, Large-scale Graph**

I. Introduction

The performance of applications based on graph-structured data has been significantly improved through the adoption of Graph Neural Networks (GNNs) [1]–[4]. Since GNNs have been effectively applied to graphs across various domains, including social media, e-commerce, and financial services, numerous real-world applications can now be implemented using GNNs. Consequently, graph databases integrated with GNN frameworks can support web services such as recommendation systems [5], [6], traffic prediction [7], and fraud detection [8], [9].

Despite the promising potential of GNN-powered applications, previous works on GNN accelerators and systems have paid less attention to the practical implementation of real-time services. Real-time GNN inference systems must achieve sufficiently low latency to meet Service Level Agreements (SLAs) for each arrival of inference request. However, prior efforts to accelerate GNN inference have commonly focused on full-batch inference [10]–[13], which has a significant gap from the requirements of practical real-time services. Industrial graphs often contain billions of nodes, with sizes reaching terabyte-scale or beyond. In such cases, computing the entire graph to obtain inference results for requested nodes is highly inefficient.

To address this inefficiency, computing the inference results only for the requested target node batch is a practical approach, widely adopted across both academia and industry. The workflow of this GNN inference serving system can be divided into two steps: the preparation step and the computation step. Industrial-scale graphs are typically vast and dynamic, often managed by graph databases. During the preparation step, the system retrieves the computational graph and input node embeddings from the graph database. Subsequently, in the computation step, these input data for the GNN models are transferred to accelerator devices (e.g., GPUs) and processed to produce inference results.

Although this solution appears reasonable, there are two significant challenges that hinder the achievement of low-latency serving. First, the entire graph is often much larger than the available device memory, causing the preparation step to suffer from low host memory bandwidth and limited I/O bus bandwidth. As a result, the preparation step consumes a dominant portion of the end-to-end latency. The second challenge is the neighbor explosion problem [2], [14], [15]. In each GNN layer, nodes aggregate information from their neighbors, and stacking multiple GNN layers leads to an exponential increase in the number of required input nodes. This issue significantly raises computation and memory access costs, resulting in prolonged latency.

To address this problem, we first identify the specific characteristics of real-time, user-facing GNN services. Real-time graphs are dynamic, with nodes and edges being continuously added or deleted. However, from the perspective of each timestamp when a request arrives, the portion of target nodes occupies only an extremely small part of the entire graph. The remaining part of the graph becomes static after new nodes are added. For this reason, we focus on these existing nodes, dubbed background nodes, for efficient inference.

In this paper, we propose FLAG, an FPGA-based low-latency GNN inference serving system that employs vector quantization. To minimize heavy data transfer, hidden embeddings of background nodes are precomputed and compressed using vector quantization during the offline preprocessing phase. For online GNN inference serving, we introduce a lightweight aggregation method that leverages the precomputed background nodes to eliminate the neighbor explosion problem. The dedicated FLAG hardware is designed and implemented on an FPGA to support both computation and embedding updates. As a result, FLAG system achieves average speedups of 154×, 176×, and 333× on three GNN models compared to the baseline system.

II. Background and Motivation

A. Fundamentals of GNNs

A Graph Neural Network (GNN) is a category of neural network models designed to handle graph-structured inputs. A common characteristic of modern GNN models is the graph convolution operation, which conducts aggregation of features from adjacent nodes. In each GNN layer, node embeddings are updated through two distinct computation phases: *aggregation* and *combination*.

$$a_v^l = Aggr(\{h_u^{l-1} : u \in N(v)\}), \quad h_v^l = Comb(h_v^{l-1}, a_v^l) \quad (1)$$

As shown in Equation 1, firstly, the target node v gathers embeddings (h_u^{l-1}) from its neighbor nodes ($N(v)$) and reduce them into single embedding (a_v^l) using an aggregation function (*Aggr*). The aggregation function, which is predefined according to the specific GNN model, can be operations such as sum, mean, or weighted sum. The output of the aggregation phase is then transformed into a new embedding (h_v^l) in a different dimensional space using a combination function (*Comb*), typically implemented as matrix multiplication.

$$X^l = \sigma(C^l X^{l-1} W^l) \quad (2)$$

Most of GNN models can be simply formulated as matrix multiplication, as shown in Equation 2. $X^{(l)}$ represents the node embedding matrix resulting from the l-th layer. In each GNN layer, node embeddings are aggregated and transformed through multiplication by a convolution matrix C^l and a weight matrix W^l. For instance, if we apply GCN [1] to this model , the convolution matrix becomes the normalized adjacency matrix $C = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}$.

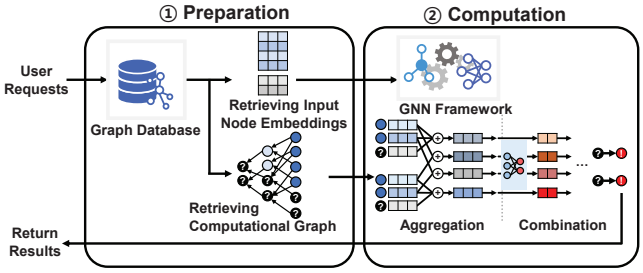


Fig. 1. Real-time GNN inference system workflow.

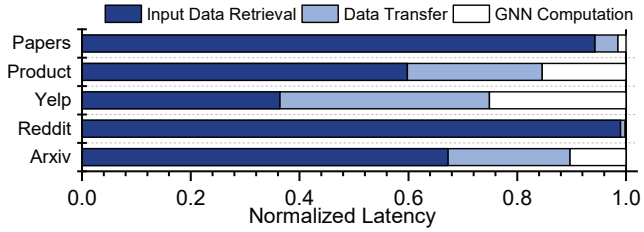


Fig. 2. Breakdown of GNN inference

B. Real-time GNN Inference Workflow on Large-scale Graph

Graph data in industrial domains is typically extremely large and dynamic, so graph-structured data is commonly managed using graph databases. Real-time GNN inference serving solutions [16] are implemented using GNN frameworks integrated with these graph databases. Although GNNs have a wide range of applications, this paper specifically focuses on node classification.

In the given solution, we can divide the end-to-end inference workflow into two distinct steps. Figure 1 illustrates the abstraction of the real-time GNN inference system workflow. When a user request arrives for target nodes, the system prepares the necessary input data, including input node embeddings and the computational graph, from the graph database for GNN model computation. These data are then packed and sent to the GNN framework for inference, and the inference results are subsequently returned to the user. The detailed process of serving inference results is described as follows. Important processes are highlighted using numbering.

The overall workflow begins with a user transaction or behavior that typically affects the existing graph, prompting the system to ❶ **update the graph**. During this step, newly added nodes and edges may be inserted into or deleted from the graph data. Following these updates, ❷ **subgraphs are extracted** to construct the computational graph for the target nodes, preparing the input for the GNN model. Additionally, ❸ **node embeddings are retrieved** from the graph database according to the computational graph. These two sets of input data are then ❹ **transferred to the accelerator devices**. Finally, the devices, supported by the GNN framework, ❺ **compute the GNN model** and return the inference results.

Figure 2 shows the portion of each step in the end-to-end GNN inference workflow. The Input Data Retrieval process encompasses steps 1 to 3, while Data Transfer and GNN Computation correspond to steps 4 and 5, respectively. The experiment was conducted using a 2-layer GraphSAGE model implemented with the DGL framework. In most cases, the preparation phase—comprising input data retrieval and data transfer—dominates the end-to-end latency. Notably, input data retrieval accounts for the largest portion across all datasets, consuming up to 99% of the overall latency. This analysis highlights the critical need to mitigate the substantial preparation time to achieve low-latency, practical GNN inference serving. Therefore, our work proposes a direction to alleviate this challenge.

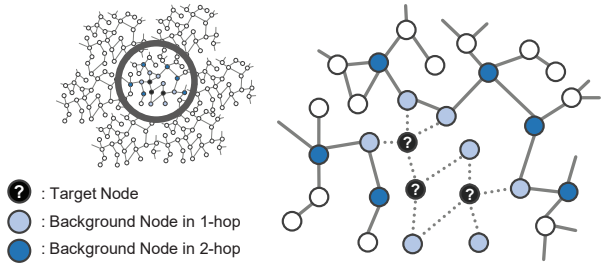


Fig. 3. Target nodes and background nodes

C. Opportunities in Background Nodes

The challenges of real-time GNN services arise from the need to handle dynamic, large-scale graphs efficiently. When the graph is static and includes the target nodes—referred to as the transductive setup—inference can be performed during the training phase. In this case, the system only needs to retrieve the inference output from the database for each request without additional computation. However, in real-time, user-facing environments, requests often involve newly added nodes. This inductive nature of GNN inference serving introduces the aforementioned challenges.

Although the graph is dynamic, at each time step, only the portion containing the target nodes and their edges significantly differs from the previous time step. In other words, most of the graph remains relatively static over small time intervals. In this paper, we refer to these existing nodes as ***background nodes***, borrowing and extending the concept from [17]. As illustrated in Figure 3, target nodes occupy only a small portion of the entire graph and require 2-hop neighboring background nodes for computing inference results. This implies opportunities to accelerate GNN inference. If unseen, newly added nodes are requested, most of the input operand nodes are already known, existing nodes. Therefore, we leverage offline preprocessing of these background nodes through precomputation and compression methods before the service.

D. Efficient GNN Inference Serving with Data Compression

Caching compressed data in device memory is an intuitive solution to reduce data transfer, but careful selection of the compression method and targets is essential to achieve this objective. For instance, some previous works [17], [18] employ graph condensation for efficient GNN inference serving. This approach condenses the original graph into a smaller synthetic graph that maintains similar representational properties. Nodes that are not target nodes are compressed using condensation, thereby reducing memory requirements and lowering computation costs. However, this approach is unsuitable for real-time, user-facing environments. Because graph condensation loses the detailed structure of the original graph, it cannot handle dynamic graphs that require frequent updates.

To address this problem and achieve our goal, we employ vector quantization, a widely used data compression method, to reduce dimensionality in our GNN inference serving system. In contrast to graph condensation, which compresses both structural and feature data, vector quantization focuses solely on the node embedding matrix. Given a set of vectors, the process begins with finding centroid vectors, known as codewords, through k-means clustering. Once the codebook, which consists of these codewords, is prepared, the original vectors are replaced by the index of the closest codeword. Previous works [19], [20] have applied vector quantization to address similar challenges, but primarily in the context of training. In this paper, we propose a vector quantization-based solution specifically designed for a practical inference environment.

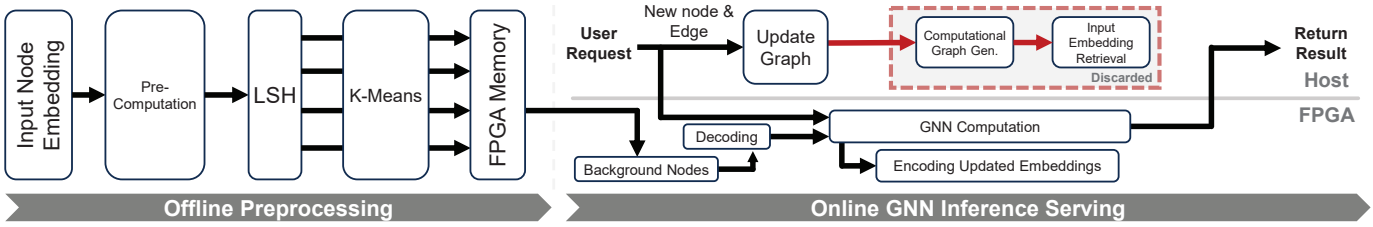


Fig. 4. Overview of the FLAG System Workflow

III. Proposed Work

In this section, we introduce our proposed system, FLAG, an FPGA-based solution for low-latency GNN inference service utilizing vector quantization.

A. System Overview

As shown in Fig. 4, the overall workflow of the FLAG system is divided into offline preprocessing and online GNN inference serving. Before the service begins, hidden embeddings of existing nodes, which act as background nodes during inference, are precomputed and compressed. The compression method employs vector quantization, incorporating locality-sensitive hashing to consider data locality and layout. Once preprocessing is complete, online GNN inference serving is performed on the FLAG system. A dedicated accelerator architecture is deployed on the FPGA, efficiently handling both computation and embedding updates. Furthermore, to mitigate the neighbor explosion problem, we propose a lightweight aggregation method that leverages precomputed hidden embeddings. As a result, the proposed FLAG system achieves low-latency GNN inference service by addressing the aforementioned challenges.

B. Vector Quantization with Locality-Sensitive Hashing

The objective of offline preprocessing is to precompute the inference results for background nodes and compress them so that they fit into the device memory. Figure 5 illustrates the FLAG offline preprocessing process, and Algorithm 1 provides a detailed description.

Given the input embeddings of background nodes X_B^0 and the pretrained GNN model, we first precompute the hidden embeddings of the background nodes (lines 1-5). During this step, we store the hidden embeddings after multiplying them by the weight matrix (line 3). Since the output dimension is generally smaller than the input dimension, this reduces the storage overhead for the precomputed outputs. The precomputation for each layer includes an additional step to generate threshold vectors via a column-wise mean operation on the hidden embeddings (line 4), which are subsequently used for hashing nodes into sub-groups.

After completing the precomputation stage, vector quantization is applied. Our vector quantization method is a modified form of product quantization. Initially, each embedding matrix is divided into M submatrices to minimize computation costs and reconstruction errors (line 6). The row vectors in each submatrix are then compared with the previously acquired thresholds, and the subvectors are hashed into $2^{(f/M)}$ buckets (line 9). This process is similar to locality-sensitive hashing using random projection, providing two key benefits for performance optimization. First, the online encoding process for updated or newly added embeddings can be significantly lightweight due to the hash function. Second, data locality is improved by allocating similar codewords to adjacent physical addresses. Given that GNN datasets exhibit high correlation [17], [21], [22] among

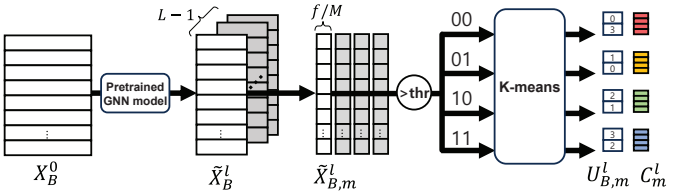


Fig. 5. FLAG offline preprocessing

Algorithm 1 FLAG Offline Preprocessing Algorithm

INPUT: Input Embeddings X_B^0, Weight Matrices $W^1, \cdots W^L$
OUTPUT: Index Matrices $U_{B,m}^l$, Codebooks C_m^l

1: **for** l in 1 to L **do**
2: $\quad X_B^l \leftarrow GNNLayer^l(X_B^{(l-1)})$
3: $\quad \tilde{X}_B^l \leftarrow X_B^{(l-1)} W^l$
4: $\quad thr^l \leftarrow \tilde{X}_B^l.column_wise_mean()$
5: **end for**
6: $\tilde{X}_{B,m}^l, thr_m^l := \tilde{X}_B^l[\frac{(m-1)f^l}{M} : \frac{mf^l}{M}], thr^l[\frac{(m-1)f^l}{M} : \frac{mf^l}{M}]$
7: **for** l in 1 to $L-1$ **do**
8: $\quad$ **for** m in 1 to M **do**
9: $\quad\quad idx_m^l \leftarrow element_wise_compare(\tilde{X}_{B,m}^l, thr_m^l)$
10: $\quad\quad$ **for** i in 0 to $2^{(f/M)} - 1$ **do**
11: $\quad\quad\quad u_i \leftarrow idx_m^l == i$
12: $\quad\quad\quad U_{B,m}^l[u_i], C_m^l[ik:+k] \leftarrow k_means(\tilde{X}_{B,m}^l[u_i])$
13: $\quad\quad$ **end for**
14: $\quad$ **end for**
15: **end for**
16: **return** $U_{B,m}^l, C_m^l$

connected nodes, nodes with similar embeddings are likely to be clustered together. After hashing, the subvectors within the same buckets are encoded using k-means clustering (line 12). Consequently, the codebooks C_m^l and the index matrices $U_{B,m}^l$ are returned.

With this offline preprocessing step, we obtain a compressed representation of the original large-scale embedding matrices. This addresses one of the challenges associated with heavy data retrieval on the host side and the substantial data transfer to the device. However, the compressed data includes not only the input node embeddings but also the precomputed hidden embeddings. The role of these precomputed hidden embeddings is introduced in the next subsection.

C. Light-weight Aggregation with Precomputed Background Nodes

To achieve low-latency inference, one of the biggest challenges, the neighbor explosion problem, must also be addressed. In this subsection, we propose a lightweight aggregation method that leverages precomputed background hidden embeddings and efficient approximated computations.

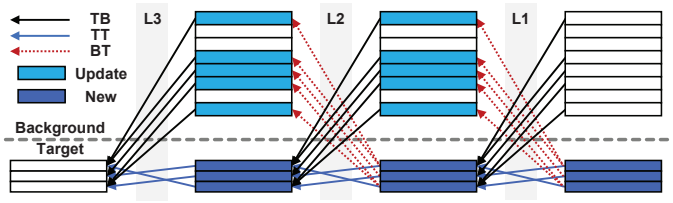


Fig. 6. Target node aggregation and Background node update

The baseline system generates the computational graph for the target nodes before inference and retrieves the input node embeddings for the first layer of the GNN model. In this process, the receptive field of the target nodes expands to their L-hop neighbors. The memory access and operation complexity of the first layer is $O(bd^L)$, where b denotes the batch size of the target nodes, d denotes the average degree, and L denotes the number of layers. In the FLAG system, the hidden embeddings of the background nodes are precomputed, eliminating the need for most of the aggregation computation that leads to the neighbor explosion problem. Although it may seem that all the issues causing the neighbor explosion problem have been resolved, one latent issue remains.

The problem is that while background nodes contribute to the hidden embeddings of target nodes, the target nodes also influence the background nodes. For clear explanation, let us denote a target node as **T** and a background node as **B**. Figure 6 illustrates an example computational graph of a 3-layered GNN model. To avoid confusion, we illustrate the embedding matrices of the background nodes in their decoded form.

For the target nodes, the aggregation is straightforward, involving the aggregation of input node embeddings from the target nodes and their neighboring nodes in the background. Equation 3 describes this operation, which we refer to as *target aggregation*. Two types of aggregation, including TB (black line) and TT (blue line), are also illustrated in Figure 6.

$$X_T^l \leftarrow Aggr^l(X_T^{l-1}, \widetilde{X}_{B\cap N(T)}^{l-1}) \tag{3}$$

Before a target node is added to the existing graph, the hidden embeddings of the background nodes are complete. However, once the target node is added, not only do the background nodes influence the target node, but the target node also affects the background nodes, as illustrated in Figure 6. Consequently, the current values of the background nodes connected to the newly added target node must be updated. Equation 4 describes this operation, referred to as the *background update*. In Figure 6, BT (red line) represents this type of aggregation.

$$\widetilde{X}_{B\cap N(T)}^l \leftarrow Aggr^l(\widetilde{X}_{B\cap N(T)}^l, X_T^{l-1}) \tag{4}$$

The proposed lightweight aggregation method, consisting of target aggregation and background update, maximizes the benefits of pre-computation, enabling the avoidance of unnecessary aggregation. In addition, the most significant advantage of this method is that it does not require an L-hop propagated computational graph. Consequently, we can eliminate the computational graph generation process and the exponential increase in both memory access and computational complexity.

Despite the advantages of the proposed aggregation method, it does have two drawbacks. The first issue is that some background updates, which need to be scattered from the updated background nodes, may be neglected when the layer depth is 3 or more. As more GNN layers are stacked, errors can accumulate. However, no noticeable impact on accuracy has been observed when the number of layers is 3. Since GNN models typically do not use a layer depth of 4 or more, this issue is not a significant concern. The second drawback is the need for encoding updated and newly introduced embeddings, which introduces overhead due to the use of a vector-quantized compressed format. To mitigate potential performance degradation, we design hardware that supports both computation and the encoding/decoding processes without introducing additional latency.

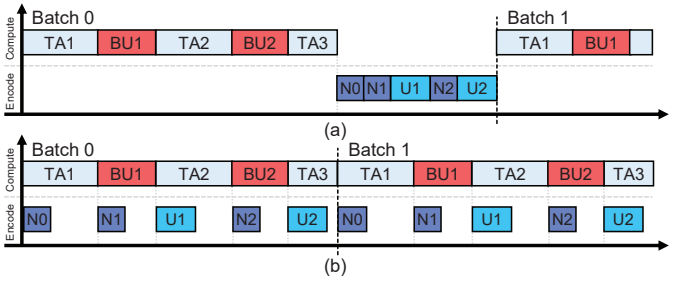


Fig. 7. Computation and encoding (a) without pipelining, and (b) with pipelining

D. FLAG Hardware Architecture Design

Computation and Encoding Pipeline. The objective of the FLAG architecture design is to support the proposed computation framework while minimizing the overhead associated with non-computation processes, such as embedding encoding for updates. To achieve this, the FLAG architecture is divided into two main blocks: the computation block and the encoding block. As described in the previous subsection, the proposed computation flow necessitates encoding the embeddings of updated background nodes and newly added target nodes. This encoding process would not be a concern if inference were performed only once. However, we assume a scenario where continuous inference requests are received.

Figure 7 illustrates two scenarios: (a) without pipelining and (b) with pipelining. In this example, TA denotes Target Aggregation, and BU denotes Background Update, while N and U represent New and Update, respectively. For clarity, the color scheme used is similar to that in Figure 6. As shown in Figure 7, the latency of the encoding step accumulates on top of the computation latency when pipelining is not used. If the next request batch arrives before batch 0 has completed processing, stalls occur. However, with pipelining, the encoding latency is hidden within the computation latency. From the perspective of latency, this effectively eliminates the overhead of encoding.

FLAG Architecture Overview. Figure 8 describes an overview of the FLAG architecture. To enable computation and encoding pipelining, the architecture is divided into a computation engine and an encoding engine. The computation engine consists of two main units: the combination unit and the aggregation unit. The combination unit is implemented using a systolic array to efficiently combine the embeddings of the requested target nodes. The aggregation unit is a SIMD array specialized for vector computation, executing both target aggregation and background updates. When precomputed hidden embeddings are used, the decoder reads the codeword corresponding to each node's code index and concatenates them to form the input operands for the aggregation unit. Several types of buffers store different data sets, such as the input buffer, weight buffer, and edge buffer. The global buffer is primarily used for storing unencoded hidden embeddings but can also store various types of data for multiple purposes.

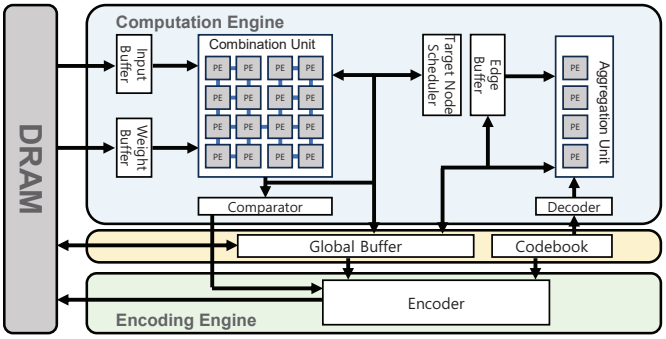


Fig. 8. FLAG Hardware Architecture

Target node scheduling and encoding. When the embeddings of a target node batch arrive at FLAG, the first step is the combination phase, which involves projecting the embeddings through multiplication with the weight matrix. Because we utilize a threshold-based compression method during offline preprocessing, the binary code produced by thresholding the outputs from the combination unit in the comparator becomes highly useful. First, this binary code can be employed by the encoder to search for the most similar codeword. Since we use the same thresholds and the same projection matrix (the weight matrix) during online serving, FLAG enables lightweight encoding without requiring complex comparisons. Another way to leverage this binary code is to schedule the computation order of the target nodes within a batch. The characteristics of the GNN dataset are also applicable to newly added target nodes. In other words, target nodes hashed into the same buckets are likely to exhibit similar connection patterns. By reordering these nodes, FLAG enhances the temporal locality of computing nodes. Since symmetric TB aggregation and BT update are performed at each layer, the impact of target node scheduling is effectively doubled.

IV. Evaluation

A. System Configuration for Experiments

Baseline System. The baseline system consists of a machine with 2 Intel Xeon Silver 4210R CPUs (384GB) and 4 NVIDIA RTX 3090 GPUs (24GB, 936GB/s). We implement the baseline workflow similar to [16] using the Deep Graph Library (DGL) [23].

FPGA system setup. The FLAG system consists of a machine with 2 Intel Xeon Gold 6240 CPus (384GB) and AMD Xilinx u55c FPGA card (16GB, 460GB/s) [24]. The FLAG hardware design is implemented on a U55C card using Vitis and Vivado. For comparison, we also evaluate the state-of-the-art real-time inference accelerator, FlowGNN [25]. The experimental system environment for FlowGNN is identical to that used for FLAG.

B. Benchmarks

We select node property prediction benchmark datasets Arxiv, Products, Papers [26], Reddit [2], and Yelp [27] containing more than 100K nodes. Table I shows detailed information about these benchmark datasets. The GNN models used for evaluation include three representative architectures: GCN [1], GraphSAGE [2], and GAT [3]. In this work, the hidden embedding dimension for all models is set to 128, and the size of heads for GAT is set to 16. To avoid the stochastic fluctuation of the inference accuracy, we do not use neighbor sampling during evaluation. The default batch size is 128, and the model depth is 2.

TABLE I
Testbench Datasets

Dataset	Nodes	Edges	Input dim.	Classes
Arxiv	169,343	1,166,243	128	40
Reddit	232,965	114,848,857	602	41
Yelp	716,847	13,954,819	300	100
Products	2,449,029	61,859,140	100	47
Papers	111,059,956	1,615,685,872	128	172

TABLE II
Accuracy and Compression Ratio

		Arxiv		Reddit		Yelp		Products		Papers	
Model		CR	Acc(%)	CR	Acc(%)	CR	Acc(%)	CR	Acc(%)	CR	Acc(%)
GCN	Baseline	1	71.5	1	95.6	1	62.8	1	77.6	1	65.8
	FLAG	37.1	71.3	173.7	95.5	45.2	62.5	18.5	77.6	16.8	65.4
GraphSAGE	Baseline	1	71.1	1	95.9	1	63.6	1	78.3	1	66.2
	FLAG	37.1	71.1	195.8	96.1	45.2	63.4	18.5	78.2	16.8	65.9
GAT	Baseline	1	71.3	1	95.3	1	63.1	1	79.5	1	63.5
	FLAG	20.7	71.0	91.8	95.1	36.1	62.7	9.3	79.3	8.4	63.0

C. Accuracy and Compression Ratio

For the proposed approach to be practical, sufficient compression must be achieved without significant accuracy degradation. To ensure this, we determine the marginal compression ratio that results in less than a 0.5% accuracy loss. The compression ratio is calculated by dividing the size of the original embedding matrix by the size of the vector-quantized data. Table IV-C presents the data compression results, obtained using 2-layer GNN models, except for Products and Papers, which use 3-layer models.

The Reddit dataset achieves the highest compression ratio. This is because the FLAG compression method compresses the dimension-reduced embeddings, obtained by multiplying the original embeddings with the weight matrix, rather than compressing the original embeddings directly. As a result, Reddit's large input dimension of 602 significantly contributes to its compression ratio. Conversely, the Papers and Products datasets experience a disadvantage in compression ratio due to the increase in input dimensions for similar reasons.

D. End-to-end System Latency

We evaluate the end-to-end latency of the FLAG system in comparison to the baseline system and the FlowGNN system. Since the FLAG system employs offline preprocessing to avoid heavy preparation time, it significantly outperforms the existing systems. As shown in Figure 9, the FLAG system achieves up to 3224×, 3245×, 3322× speedup for each GNN models compared to the baseline system, with an average speedup of 154×, 176×, and 333×. The Reddit dataset, which has the largest input dimension (602) and the highest average degree (493), exhibits particularly remarkable speedup. In contrast, the system equipped with the state-of-the-art accelerator, FlowGNN, achieves only a modest speedup of up to 1.25x. This result underscores the importance of comprehensive, systemic solutions to address the aforementioned challenges for achieving low-latency GNN inference service.

E. Computation only latency

To prove FLAG is also optimized to low latency GNN computation, we evaluate the GNN computation time compared to baseline and FlowGNN. Since the FLAG architecture replace the most portion of aggregation by precomputation, it significantly outperforms not only GPU but also FlowGNN. As shown in Figure 10, the FLAG architecture achieves up to 381×, 318×, 1281× speedup for each GNN models compared to the baseline system, with an average speedup of 186×, 151×, and 468×. FlowGNN demonstrates significant speedup compared to the baseline GPU for computing GNN inference on mini-batches, but it fails to achieve similar results on the Reddit dataset. FlowGNN is specialized for low-latency processing of

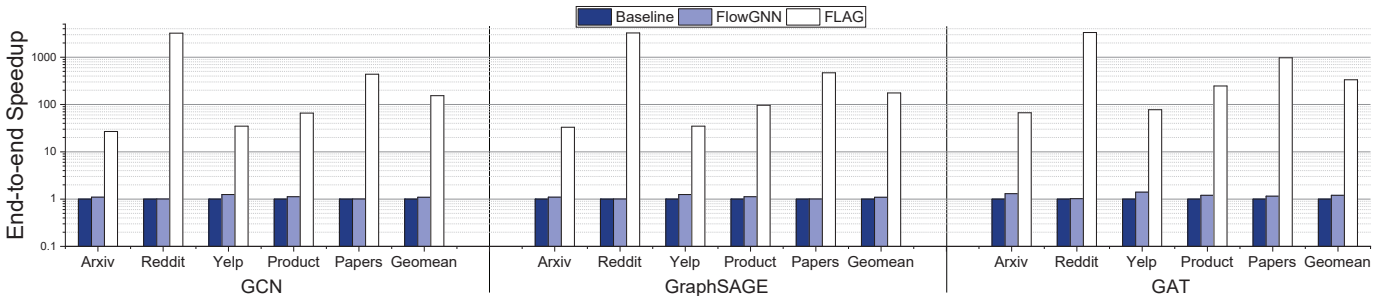


Fig. 9. End-to-end speedup of three system

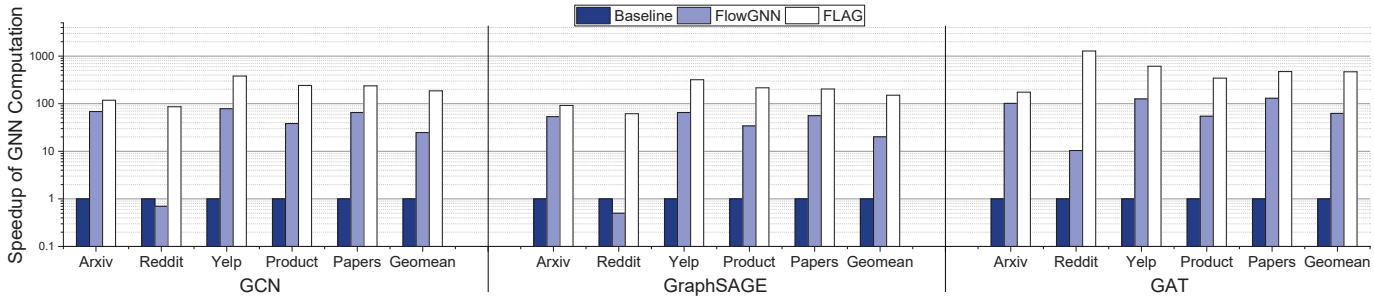


Fig. 10. Speedup of GNN computation

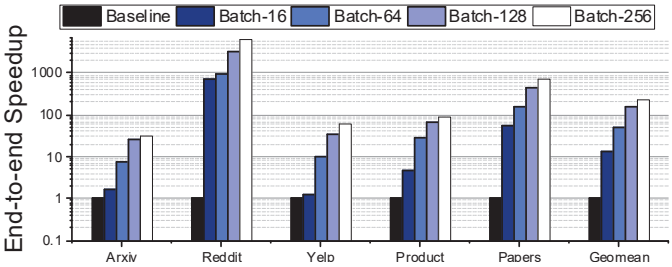


Fig. 11. End-to-end speedup according to batch size

streams of relatively small graphs. However, under our experimental condition, which samples full neighbors, nearly full-batch inference is required for the first layer of the Reddit dataset. In contrast, FLAG achieves superior performance on all GNN models compared to FlowGNN, as its precomputation-based lightweight aggregation method effectively avoids the neighborhood explosion problem.

F. Effect of Batch Size

We perform the evaluation using different batch sizes, ranging from 16 to 256, on the GraphSAGE model. All latency measurements are normalized to the baseline, which is evaluated under the same conditions as in Figure 9. As shown in Figure 11, the end-to-end speedup increases as the batch size becomes larger. Since the complexity of the neighbor explosion problem is $O(bd^l)$, a larger batch size results in greater preparation overhead. Consequently, the latency portion of the preparation step increases as the batch size grows, eventually constituting the majority of the end-to-end latency. The speedup of FLAG is achieved by minimizing the heavy preparation step.

To address this problem, neighbor sampling can be used to reduce the average degree d to a sample size s. However, this sampling method probabilistically loses important information and does not fundamentally solve the neighborhood explosion problem, as it remains exponentially proportional to s. On the other hand, our precomputation-based lightweight aggregation efficiently addresses this issue.

V. Related Work

Constructing an efficient machine learning application serving system is a crucial problem in the AI era. GNN applications are also applied to critical domains where time inefficiency can lead to severe consequences, such as fraud detection [8], [9] in financial services. This work focuses on a single FPGA-equipped server node; however, many cases of computing environments, such as fog computing [28] or cloud computing [29], should be considered to transition toward real-world systems. To this end, both hardware optimization and algorithm optimization are needed. Although they are not addressed in this paper, many methods can be applied orthogonally to other ideas. Examples include quantization techniques [30], [31] and efficient edge pruning methods [32], [33]. [34] and [35] offer effective solutions for co-optimization, but questions remain regarding their adoption for real-time services.

VI. Conclusion

The practical implementation of GNN inference services on large-scale industrial graphs is a challenging problem due to the need for low end-to-end latency to meet service level agreements. In this paper, to enable low-latency GNN inference serving, we analyzed the end-to-end workflow and identified the bottlenecks of the baseline system. Most of the latency is consumed by the inevitable preparation step, and the neighborhood explosion problem causes latency to increase exponentially with the number of layers. To address the two primary challenges, we proposed FLAG, an FPGA-based system for low-latency GNN inference service using vector quantization. In the offline preprocessing step, we introduced a hidden embedding precomputation and vector quantization method combined with locality-sensitive hashing. For the online GNN serving step, we designed a dedicated hardware architecture for FPGA, which supports our proposed ideas. As a result, we achieved average speedups of 154×, 176×, and 333× on three GNN models compared to the baseline.

Acknowledgment

This research was supported by Samsung Advanced Institute of Technology (SAIT), Samsung Electronics Co., Ltd.

REFERENCES

[1] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[2] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[3] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *6th International Conference on Learning Representations*, 2017.

[4] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[5] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," in *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 639–648, 2020.

[6] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 974–983, 2018.

[7] S. Guo, Y. Lin, N. Feng, C. Song, and H. Wan, "Attention based spatial-temporal graph convolutional networks for traffic flow forecasting," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 922–929, 2019.

[8] Z. Liu, C. Chen, X. Yang, J. Zhou, X. Li, and L. Song, "Heterogeneous graph neural networks for malicious account detection," in *Proceedings of the 27th ACM international conference on information and knowledge management*, pp. 2077–2085, 2018.

[9] D. Wang, J. Lin, P. Cui, Q. Jia, Z. Wang, Y. Fang, Q. Yu, J. Zhou, S. Yang, and Y. Qi, "A semi-supervised graph attentive network for financial fraud detection," in *2019 IEEE international conference on data mining (ICDM)*, pp. 598–607, IEEE, 2019.

[10] M. Yan, L. Deng, X. Hu, L. Liang, Y. Feng, X. Ye, Z. Zhang, D. Fan, and Y. Xie, "Hygcn: A gcn accelerator with hybrid architecture," in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 15–29, IEEE, 2020.

[11] T. Geng, A. Li, R. Shi, C. Wu, T. Wang, Y. Li, P. Haghi, A. Tumeo, S. Che, S. Reinhardt, and M. C. Herbordt, "Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 922–936, 2020.

[12] J. Li, A. Louri, A. Karanth, and R. Bunescu, "Gcnax: A flexible and energy-efficient accelerator for graph convolutional neural networks," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 775–788, 2021.

[13] R. Hwang, M. Kang, J. Lee, D. Kam, Y. Lee, and M. Rhu, "Grow: A row-stationary sparse-dense gemm accelerator for memory-efficient graph convolutional neural networks," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 42–55, 2023.

[14] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 257–266, 2019.

[15] J. Chen, J. Zhu, and L. Song, "Stochastic training of graph convolutional networks with variance reduction," in *International Conference on Machine Learning*, pp. 942–950, PMLR, 2018.

[16] H. W. Jian Zhang and M. Zhu, "Build a gnn-based real-time fraud detection solution using amazon sagemaker, amazon neptune, and the deep graph library." https://aws.amazon.com/cn/blogs/machine-learning/build-a-gnn-based-real-time-fraud-detection-solution-using-amazon-sagemaker-amazon-neptune-and-the-deep-graph-library/, 2022. Accessed: (2024-11-06).

[17] L. Cao, H. Deng, Y. Yang, C. Wang, and L. Chen, "Graph-skeleton: 1% nodes are sufficient to represent billion-scale graph," in *Proceedings of the ACM Web Conference 2024*, WWW '24, (New York, NY, USA), p. 570–581, Association for Computing Machinery, 2024.

[18] S. Si, F. Yu, A. S. Rawat, C.-J. Hsieh, and S. Kumar, "Serving graph compression for graph neural networks," in *The Eleventh International Conference on Learning Representations*, 2023.

[19] M. Ding, K. Kong, J. Li, C. Zhu, J. Dickerson, F. Huang, and T. Goldstein, "Vq-gnn: a universal framework to scale-up graph neural networks using vector quantization," in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21, (Red Hook, NY, USA), Curran Associates Inc., 2024.

[20] Y. Ma, P. Gong, T. Wu, J. Yi, C. Yang, C. Li, Q. Peng, G. Xie, Y. Bao, H. Liu, and Y. Xu, "Eliminating data processing bottlenecks in gnn training over large graphs via two-level feature compression," *Proc. VLDB Endow.*, vol. 17, p. 2854–2866, Aug. 2024.

[21] D. Lim, F. Hohne, X. Li, S. L. Huang, V. Gupta, O. Bhalerao, and S. N. Lim, "Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20887–20902, 2021.

[22] J. Zhu, Y. Yan, L. Zhao, M. Heimann, L. Akoglu, and D. Koutra, "Beyond homophily in graph neural networks: Current limitations and effective designs," *Advances in neural information processing systems*, vol. 33, pp. 7793–7804, 2020.

[23] M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," 2020.

[24] X. Inc., "Alveo u55c high performance compute card," 2024. Accessed on May 01, 2024.

[25] R. Sarkar, S. Abi-Karam, Y. He, L. Sathidevi, and C. Hao, "Flowgnn: A dataflow architecture for real-time workload-agnostic graph neural network inference," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 1099–1112, IEEE, 2023.

[26] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," *arXiv preprint arXiv:2005.00687*, 2020.

[27] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. K. Prasanna, "Graphsaint: Graph sampling based inductive learning method," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, OpenReview.net, 2020.

[28] L. Zeng, P. Huang, K. Luo, X. Zhang, Z. Zhou, and X. Chen, "Fograph: Enabling real-time deep graph inference with fog computing," in *Proceedings of the ACM Web Conference 2022*, WWW '22, (New York, NY, USA), p. 1774–1784, Association for Computing Machinery, 2022.

[29] H. Hu, F. Liu, Q. Pei, Y. Yuan, Z. Xu, and L. Wang, "grapher: A resource-efficient serverless system for gnn serving through graph sharing," in *The Web Conference 2024*, 2024.

[30] Z. Liu, K. Zhou, F. Yang, L. Li, R. Chen, and X. Hu, "EXACT: Scalable graph neural networks training via extreme activation compression," in *International Conference on Learning Representations*, 2022.

[31] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, "Degree-quant: Quantization-aware training for graph neural networks," *arXiv preprint arXiv:2008.05000*, 2020.

[32] Z. Liu, K. Zhou, Z. Jiang, L. Li, R. Chen, S.-H. Choi, and X. Hu, "Dspar: An embarrassingly simple strategy for efficient gnn training and inference via degree-based sparsification," *Transactions on Machine Learning Research*, 2023.

[33] H. Seo, J. Yun, and E. Yang, "Teddy: Trimming edges with degree-based discrimination strategy," in *The Twelfth International Conference on Learning Representations*.

[34] Z. Zhu, F. Li, G. Li, Z. Liu, Z. Mo, Q. Hu, X. Liang, and J. Cheng, "Mega: A memory-efficient gnn accelerator exploiting degree-aware mixed-precision quantization," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 124–138, IEEE, 2024.

[35] C. Chen, K. Li, X. Zou, and Y. Li, "Dygnn: Algorithm and architecture support of dynamic pruning for graph neural networks," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 1201–1206, 2021.

FireGuard: A Generalized Microarchitecture for Fine-Grained Security Analysis on OoO Superscalar Cores

Zhe Jiang[†] Sam Ainsworth[‖], Timothy Jones[¶]
[†]National Center of Technology Innovation for EDA, School of Integrated Circuits, South East University, People's Republic of China,
[¶]University of Cambridge, United Kingdom, [‖]University of Edinburgh, United Kingdom

***Abstract*—High-performance security guarantees rely on hardware support. Generic programmable support for fine-grained instruction analysis has gained broad interest in the literature as a fundamental building block for the security of future processors. Yet, implementation in real out-of-order (OoO) superscalar processors presents tough challenges that cannot be explored in highly abstract simulators. We detail the challenges of implementing complex programmable pathways without critical paths or contention. We then introduce FireGuard, the first implementation of fine-grained instruction analysis on a real OoO superscalar processor. We establish an end-to-end system, including microarchitecture, SoC, ISA and programming model. Experiments show that our solution simultaneously ensures both security and performance of the system, with parallel scalability. We examine the feasibility of building FireGuard into modern SoCs: Apple's M1-Pro, Huawei's Kirin-960, and Intel's i7-12700F, where less than 1% silicon area is introduced. The Repo. of FireGuard's source code: https://github.com/SEU-ACAL/reproduce-FireGuard-DAC-25.**

I. Introduction

With ever-growing computation capacity, modern systems increasingly execute applications on shared platforms [1], [2]. The latest in-vehicle information systems from BYD, the world's largest EV manufacturer [3], allow installation and execution of less-verified third-party workloads with life-critical workloads, threatening system-wide safety and trustworthiness [4], [5]. Similarly, Android phones allow untrustworthy applications to coexist with security-critical banking software [6]. Modern systems thus need the ability to analyze, detect and mitigate vulnerabilities in an always-on and comprehensive way.

Existing work. Always-on, comprehensive security analysis relies on hardware support [7]–[16]. Current implementations, e.g., Arm's MTE [7] and BTI [17], and Intel's LAM [18] and CET [19], have very limited flexibility, allowing an attacker to bypass them by simply shifting their targets. Also, deployment against the latest threats requires lengthy development, leading to long vulnerability windows.

Hardware-assisted fine-grained instruction analysis [20]–[24] presents a new paradigm, adding observation channels into cores to filter and analyze execution (e.g., committed instructions, memory accesses and function calls), in programmable analysis engines, (e.g., microcontrollers, accelerators or FPGAs). Through reconfigurability and parallelism, they can adapt to cover a wide range of attacks.

Challenges. Existing efforts have been conducted through software simulation [21], [22], [24] or on simple in-order processors [23], [25]. They do not show how to build analysis into real OoO superscalar cores, or consider whether this is possible without a full overhaul of core design or at palatable overhead. When we tried to build such a mechanism into a real core, we hit bottlenecks and contention at every stage of the processing pipeline, from data collection and filtering to distribution and analysis. Typical programming models [21] require full generality: any and all instructions can be monitored simultaneously, with any and all data selected from them, and sent to multiple analyses at once. Inside the core, such data must be collected in a narrow time window, at commit-time rather than execute-time to reflect ordering, but before the data is overwritten. While data can be filtered down if it is irrelevant for currently running analyses, if every instruction could be monitored, the mechanism that decides which instructions are processed must be highly parallel and high throughput to avoid back-pressure slowing down the core. Even when data volume has been reduced using filtering, the generality required of distribution made it a challenge for us to send data to multiple analysis engines at once without unscaleable broadcast.

Contributions. We introduce the *first microarchitecture* for hardware-assisted fine-grained instruction analysis, FireGuard, implemented in RISC-V BOOM [26]. To this end, we present:

- a buffer-free data-forwarding channel by inserting bypass circuits at key locations within the main core, giving fine-grained visibility of execution without significant microarchitectural invasion;
- a superscalar event filter, utilizing a fleet of SRAM-based *mini-filters* to handle commit of arbitrary instruction types at the same width as the core, shrinking the content volume for later analysis and preventing extra performance degradation.
- broadcast-free communication channels, partitioning a task mapper into a distributed fabric network and a scalable allocator. The former enables independent data paths per transaction, while the latter uses multiple *Scheduling Engines* to simultaneously route data to all interested engines;
- a microarchitecture-assisted programming model for analysis engines, with optimizations across the hardware-software stack via new queue-communication instructions, a novel tightly coupled ISA extension (ISAX) interface to minimize data hazards, and unrolling-aware custom instructions.

We implement FireGuard in Chisel, deploying on Virtex UltraScale+ FPGAs using FireSim [27]. We boot full Linux and deploy different workloads with various security schemes: a Custom Performance Counter (PMC) [21], a shadow stack [28], AddressSanitizer [29], and Use-after-Free (UaF) detection [30]. The results show that employing four analysis engines is sufficient for running a PMC with 2.5% performance overhead, a shadow stack with 2.1% overhead, AddressSanitizer with 39% overhead (reducing to 6% with 12 engines), and UaF detection with 42% overhead (16% with 12 engines). Overhead can be completely removed by further integrating hardware accelerators. We evaluate the feasibility of building FireGuard into modern SoCs: M1-Pro, Kirin-960 and i7-12700F, where less than 1% silicon area is required.

II. Fine-Grained Instruction Analysis

Although studies on fine-grained instruction analysis converge on the same top-level idea, the architecture and APIs across them slightly differ. We adopt the Guardian Council [21] as a generalized example, build FireGuard upon it and discuss the feasibility of the others.

The Guardian Council. The architecture features a generic frontend that can analyze any instruction type. It inserts a data-forwarding channel into the compute core's commit stage, collecting data (e.g., opcode, operands) from program execution. An event filter performs a pre-check on all observed data, selecting relevant information (dependent on the analyses being run) and sending it to the analysis engines through a mapper. The mapper chooses the target according to which analyses (*guardian kernels*) are running on each analysis engine. Using the passed information, the engines run guardian kernels in parallel, validating security and finding vulnerabilities.

The other architectures. The key difference between the Guardian Council and similar mechanisms [22]–[24] is the analysis engines in the backend. It deploys a sea of microcontroller-sized cores (μcores) to run user-programmable validations (running on conventional *main cores*) in parallel. This allows for highly efficient, updateable and upgradeable analysis, by exploiting parallelism within validation tasks and between multiple independent tasks at once, and makes use of the fact that μcores consume significantly less hardware, by multiple

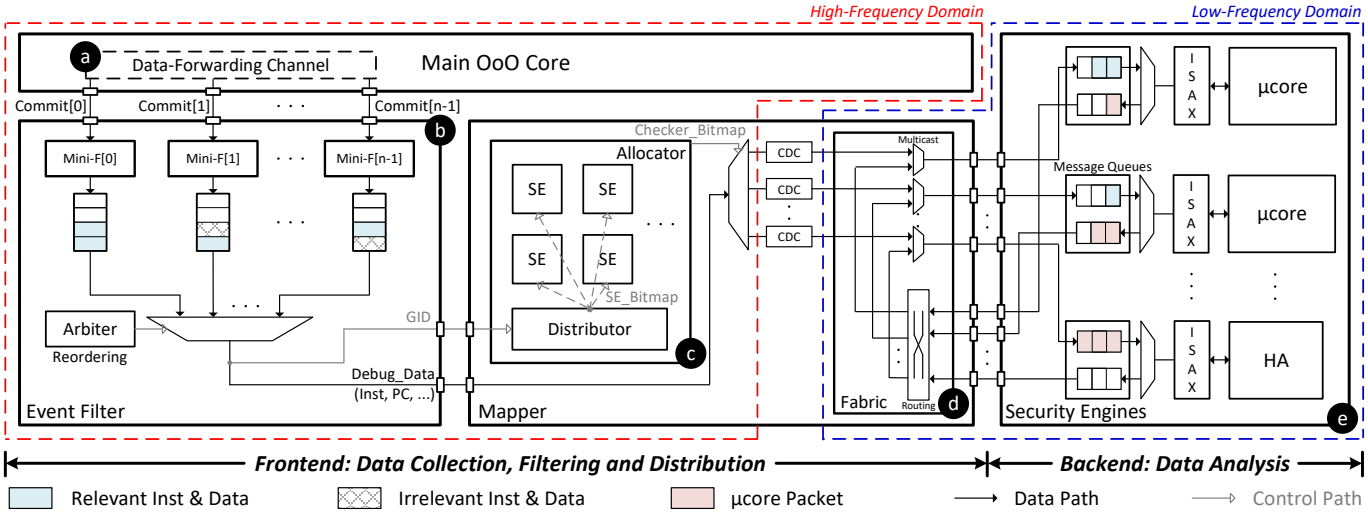


Fig. 1: Overview of FireGuard *(Mini-F: Mini-Filter; GID: Group Index*[1]*; HA: Hardware Accelerator; SE: Scheduling Engine)*: ⓐ buffer-free data-forwarding channel extracts the main core's execution events; ⓑ a superscalar filter pre-checks extracted events, identifying relevant instructions and selecting channels for analysis; ⓒ an allocator associates an SE to each kernel to distribute contents, and ⓓ a distributed fabric network transmits contents to the µcores or HAs; ⓔ kernels running on µcores or HAs fetch contents and validate their security.

orders of magnitude, compared to OoO superscalar main cores, which must achieve high single-threaded performance unnecessary for a µcore. Deploying to an FPGA [23] or to multiple large analysis engines [22], [24] involves similar challenges of distribution.

III. FireGuard

We show the concept of fine-grained instruction analysis is feasible by building a real system (FireGuard) upon the the Guardian Council architecture. Figure 1 fleshes out the data-forwarding channel, filter, mapper and analysis engines, with a careful redesign allowing practical implementation that can handle generality at each stage[2].

To do so, we integrate simple read-only bypass circuits at various locations within the main core (figure 1 ⓐ) to extract data while avoiding new buffers between execute and commit. These forward on debug data associated with committed instructions for detailed analysis. Since the design only involves minor microarchitectural changes to the main core via adding read-only interfaces, it keeps invasion into the main core low and avoids adding significant hardware overhead. With that, we deploy a *superscalar* set of SRAM-based mini-filters to handle high commit widths in parallel (figure 1 ⓑ). It can be programmed to be sensitive to any arbitrary group of instructions, and guarantees that filtering can be completed while the data is still available in the core. To timely deliver the data to the analysis engines, we divide the mapper into a scalable allocator (figure 1 ⓒ) and a distributed fabric network (figure 1 ⓓ). The allocator associates a Scheduling Engine (SE) to each guardian kernel to independently transmit contents while avoiding broadcast, while the fabric network establishes dedicated paths for filtered contents and inter-checker communication to mitigate contention. In each analysis engine, we develop a data-hazard-aware ISAX interface (figure 1 ⓔ) and integrate it into the Memory Access (MA) stage of the µcore's pipeline, connecting its message queues. This allows us to develop drivers and programming models for guardian kernels, including instructions designed for efficient, hazard-minimizing design patterns.

A. Data-forwarding channel

The data-forwarding channel is deployed at the main core's commit stage, extracting, selecting, and transporting debug data associated

[2]We partition the microarchitecture into two clock domains. The main core, along with its associated modules (e.g., L1-cache), the data-forwarding channel, the filter, and the allocator, are within a high-frequency domain driven by a fast clock source. This avoids any resulting slowdown caused by data-forwarding, filtering, and allocating activities. The more parallel µcores and fabric network are within a low-frequency domain driven by a slower clock source, with handshake-based clock-domain crossing. This ensures energy efficiency and prevents the simple µcores from becoming the critical path.

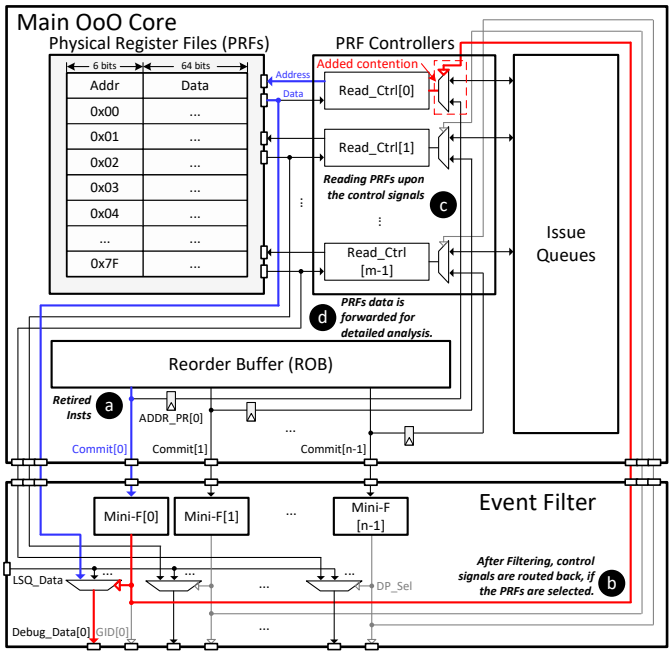


Fig. 2: Data-forwarding channel, using PRFs as an example *(blue lines: data-forwarding paths; red lines: filtering paths; gray lines: control paths; DP_Sel: Data Path Selection)*: ⓐ commit paths from the ROB are hooked, forwarding retired instructions to mini-filters and storing the PRF access addresses in temporary registers; ⓑ mini-filters pre-check the forwarded instructions, sending control signals to PRF controllers when PRF data is selected; ⓒ the data-forwarding channel preempts the controllers and feeds the addresses temporarily stored; ⓓ the read data is routed back for in-depth analysis.

with every retired instruction. Since the relevant data is already stored in different locations within the main core, we design a buffer-free implementation (i.e., adding no new intermediate storage between out-of-order execute and in-order commit), by inserting bypass circuits at the Reorder Buffer (ROB), Physical Register Files (PRFs), Load Store Queue (LSQ), and Fetch Target Queue (FTQ). This transports PC address, instruction data, operand data, and memory and jump addresses during commit for any instruction selected by the filter, avoiding reads of information not selected. These points cover all locations that store outputs of arithmetic-logic units, enabling fine-grained visibility with minimal contention.

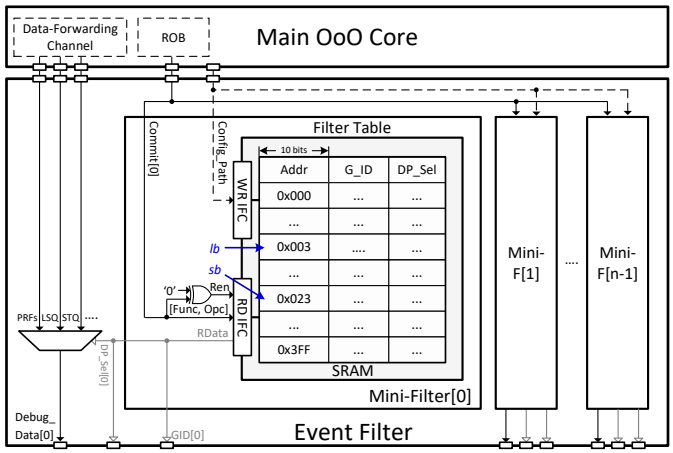


Fig. 3: Microarchitecture of a mini-filter *(gray lines: control paths; WR/RD IFC: Write/Read Interface)*.

PRF Example. Figure 2 shows the microarchitecture using PRFs as an example, connecting the ROB, filter, and PRFs. On the ROB side, we add logic to hook onto each commit, transmitting the retired instructions to the mini-filters, as well as address registers storing the PRF indices accessed by each instruction (figure 2 ⓐ). In the cycle following retirement, the mini-filters identify Group Indexes (GIDs) of the transmitted instructions and select data based on programmed settings (section III-B). If the PRF data is selected, a control signal is routed back, preempting the PRF controller (figure 2 ⓑ).

The PRF's read controllers are statically multiplexed (figure 2 ⓒ) between the issue queue (for executing instructions) and the mini-filters. Mini-Filter`[x]` has priority access to Read_Ctrl`[x]` should it require it, to allow the data to be read and transported immediately (figure 2 ⓓ), avoiding any buffering or delays in freeing the physical register. This means that an instruction attempting to use the same port will be delayed until the next cycle, resulting in contention[3].

B. Event Filter

The event filter pre-checks all retired instructions. If instructions are selected for analysis, it returns their Group Indexes (GIDs) for the mapper and programs the data-forwarding channel to select data. We give a superscalar implementation that enables simultaneous filtering of all instructions, ensuring the filter keeps up with the main core.

Figure 1 ⓑ shows its design, including a set of mini-filters, FIFO queues, and an arbiter. A mini-filter is connected to each superscalar commit path of the ROB. Each is indexed by the instruction opcode from the data-forwarding channel, and returns programmed GIDs and selects debug data from the chosen channel for instructions analyzed. Filtered contents are buffered into paired FIFO queues, allowing a shared arbiter to arrange the output into sequence.

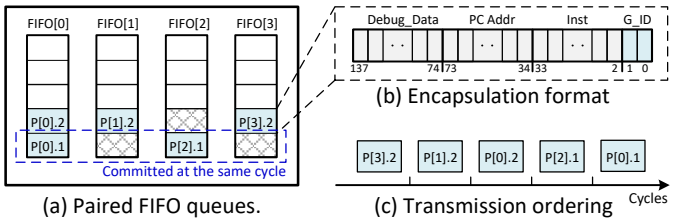


Fig. 4: Reordering with 4-width FIFOs. P[x].y: yth packet in FIFO[x].

Mini-filters. Mini-filters use a SRAM-based look-up table (figure 3). The address (10-bit) is an index formed of the concatenated RISC-V opcode (lower 7 bits) and function code (higher 3 bits) for all possible instructions, and stores the mapper's GID and the desired data paths (PRF, LSQ and/or FTQ) for each instruction. For instance, 0x03 and 0x23 index RISC-V `lb` and `sb`, respectively. We route the instructions' opcodes and function codes to the address port of the SRAM's read interface, and direct the read data to the data-forwarding channels and the mapper, assisting in data selection and allocation[4].

[3]In LDQ, STQ, and FTQ, similar microarchitectures are deployed to obtain memory access or jump addresses. Unlike PRFs, where forwarded data can be stored at arbitrary addresses, the tops of these queues consistently hold the data associated with the most recently retired instructions. When a mini-filter decides to forward a load, store, or jump address, the bypass circuits directly transmit from the relevant queue's top, avoiding contention.

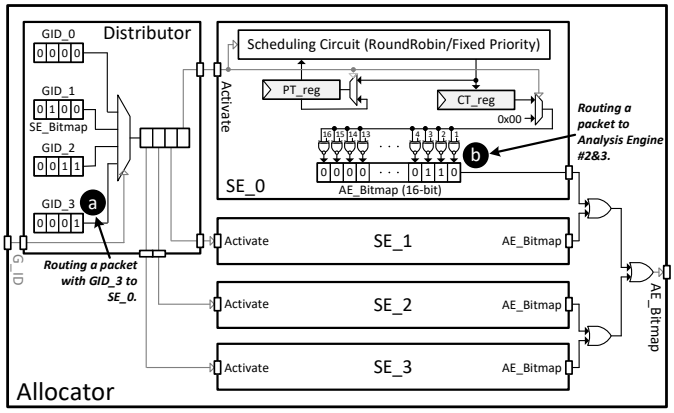


Fig. 5: Allocator microarchitecture. 3 GIDs and 4 SEs allocate contents to 16 engines *(gray lines: control paths)*.

C. Mapper

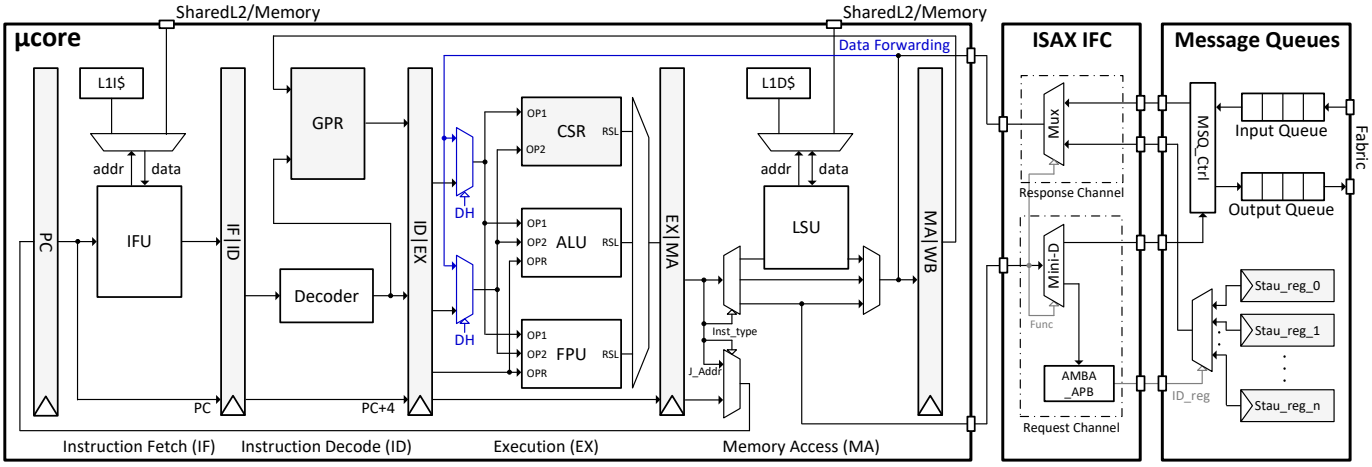
The mapper routes filtered packets to engines based on configured parallelization policies, and also enables data exchange between the analysis engines, fostering complex parallelism schemes for guardian kernels. It is fully programmable, allowing any instruction to flow to all interested guardian kernels and be scheduled to an analysis engine core for each. Figure 1 ⓒ and ⓓ illustrate the microarchitecture of the mapper, which we partition into a scalable *allocator* followed by a distributed *fabric network*. The mapper transitions FireGuard's processing from superscalar to scalar: unlike the filter, it only handles one packet per cycle. This rarely impedes a 4-wide BOOM's performance (we saw less than 0.5% slowdown)[5].

Allocator. The allocator uses a 2-level indirection bitmap to allocate packets across the analysis engines. Figure 5 shows the allocator, including a distributor and a set of Scheduling Engines (SEs). The distributor manages a bitmap between GIDs and SEs, deciding which SE(s) should be activated during packet transmission. SEs are *one-to-one* associated to a guardian kernel, and each maintains *another* bitmap between itself and analysis engines, allocating filtered packets to groups of analysis engines executing one guardian kernel.

In the distributor, an SE_Bitmap register is assigned to each GID, and individual register bits are used to index the SEs that are interested in that specific GID. For instance, if packets with GID 3 should be sent to SE 0, bit 0 in SE_Bitmap[3] is set (figure 5 ⓐ). SEs use a scheduling circuit, two scheduling registers (PT_reg and CT_reg), and an AE_Bitmap (Analysis Engine Bitmap) register. The scheduling circuit implements several policies, e.g., fixed, round-robin, and block mode [21], where the latter is used to send all messages to one µcore until it is full before moving to the next, for when message locality is important, e.g., shadow stack. When the scheduling circuit is activated, the PT_reg (*previous target*) is

[4]We place a pair of FIFO queues to connect the mini-filters, buffering the filtered contents (figure 4(a)). Although these filtered contents are produced by the mini-filters in parallel, they must be sent sequentially, aligning with their commit order, since analyses can be sensitive to program order (e.g., shadow stack [28]). We encapsulate filtered contents to achieve this (figure 4(b)). If an instruction is discarded, an invalid packet is generated and also pushed into the FIFO queue in order to preserve ordering at the end of the FIFO. The arbiter uses a finite state machine to transmit packets (figure 4(c)) in-order, consuming one clock cycle for a valid packet while skipping invalid packets.

[5]Even so, a superscalar mapper could be considered for a more powerful core. To achieve this, modifications are necessary for both the fabric and allocator, including duplicating communication channels and SEs. Extra arbiters must be deployed to manage contention, e.g., when multiple packets are sent to the same security engine.



(a) Coupling ISAX interface into µcore pipeline (blue lines: data forwarding to handle data hazards) (b) Connecting the message queues

Fig. 6: Custom instructions are integrated into the μcore's pipeline at the MA stage. *(gray lines: control paths; MSQ_Ctrl: Message Queue Controller; IFU: Instruction Fetch Unit; Mini-D: Mini-Decoder):* For the ISAX interface (IFC), a full-duplex microarchitecture handles requests and responses independently, with a mini-decoder and multiplexer. In the request channel, a mini-decoder directs an ISAX request to the message queues' MSQ_Ctrl or status registers (via an APB bridge). In the response channel, a multiplexer selects returned data, routing it to the commit stage and the EX stage using forwarding logic for handling data hazards.

Instruction	Target Queue	Descriptions
`count rd, rs1`	Input/ Output	Count number of packets buffered in message queue rs1, returning it to rd.
`top rd, rs1`	Input	Return bitfields [rs1+63:rs1] of the first element to rd.
`pop rd, rs1`	Input	Remove the first element and return its bitfields [rs1+63:rs1] to rd.
`recent rd, rs1`	Input	Return bitfields [rs1+63:rs1] of the most recently removed element to rd.
`push rs1`	Output	Push packet rs1 for transmission.

TABLE I: Main control instructions for the message queues.

used to generate the *current target*, buffered in CT_reg until it is moved to the PT_reg after packet transmission. The CT_reg value sets the relevant target bit in the SE's AE_Bitmap (figure 5 ⓑ). The AE_Bitmaps returned by all SEs are combined using OR gates to form the decision, selectively broadcasting to the analysis engines.

Fabric network. The fabric network features a half-duplex multicast (1-to-N) channel and a full-duplex routing (N-to-N) channel. The multicast channel selectively broadcasts packets, while the routing channel allows the checkers to transmit packets among themselves.

The multicast channel uses multiplexers to direct packets from the event filter to the message queues in the analysis engines. All multiplexers are controlled by the allocator, regulating transmission and masking of each packet. The routing channel uses a Manhattan grid [31]–[34] Network-on-Chip (NoC) mesh. Each router has five bi-directional ports, connected to other routers located to its north, south, east, and west, as well as to an analysis engine.

D. ISA and Programming Model

The analysis engines run guardian kernels concurrently on μcores, finding vulnerabilities. To handle the frequent hardware-software interactions, we use a FIFO-based programming model implemented into RISC-V Rocket cores via custom instructions. As these instructions take up a large fraction of total μcore cycles, we integrate FIFO-management instructions using a new tight-coupling arrangement; Rocket's existing ISAX interface, which executes custom instructions post-commit, caused too many data hazards. We also redesign our instruction set to better support high-throughput programming design patterns that minimize hazards further.

ISA. We connect the message queues to the fabric network in the mapper, allowing the μcore to receive packets via an input queue and send packets through an output queue, in order to support pipelined parallelism strategies such as used in the shadow stack [21]. Our

Main core	
Core	4-Width, out-of-order SonicBOOM [26], @3.2GHz
Pipeline	128-Entry ROB, 96-Entry IQ, 32-entry LDQ/STQ, 128 Int/FP Phy Registers, 2 Int ALUs, 1 FP/Multi/Div ALU, 2 MEM, 1 Jump Unit, 1 CSR Unit
Branch Predictor	TAGE algorithm, 256-entry BTB, 32-entry RAS, 6 TAGE table with 2 - 64 bits history
Memory	
L1 ICache	32KB, 8-way, 8 MSHRs
L1 DCache	32KB, 8-way, 8 MSHRs
L2 Cache	512KB, 8-way, 12 MSHRs
LLC	4MB, 8-way, 8 MSHRs
Memory	16 GB DDR3 @1066MHz, max 32 requests
FireGuard and Interconnects	
Event Filter	4-width, 16-entry FIFO
Mapper	4 SEs, 8-entry CDC, fabric @1.6GHz
Analysis Engine	In-order Rocket μcore [35], 5-stage pipeline, @1.6GHz, 32-entry message queues, no FPU
L1 Cache	4KB, 2-way for both I- and D-Cache
Interconnect	Memory bus @ 1GHz, others @ 3.2GHz

TABLE II: Hardware configurations evaluated.

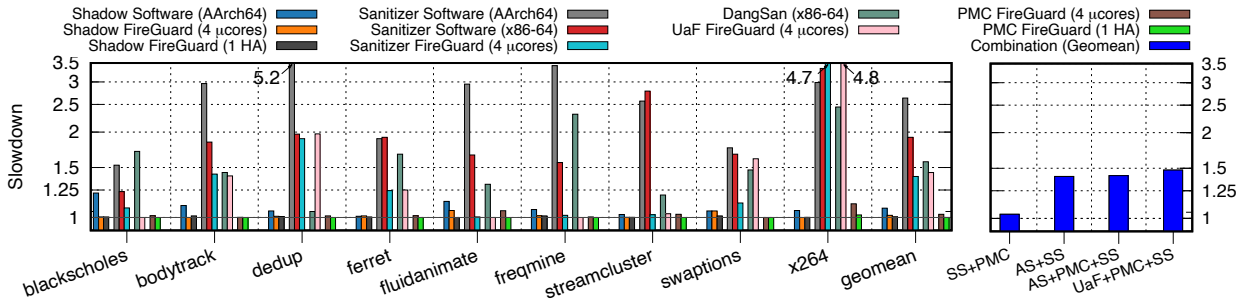
queue controller, with status registers and software drivers, allows guardian kernels to manage the message queues using ISAX [35] custom instructions (see table I). `Top`, `push` and `pop` were inherited from the original design [21]; `count` was added to aid in loop unrolling, and `recent` to allow accessing extra information about an element already processed: for example, the PC is only needed on a detected error in AddressSanitizer, and is discarded otherwise.

Microarchitecture Support. RISC-V Rocket runs custom instructions post-commit [35]. The routing to the ISAX peripheral blocks the core for at least 3 cycles for each instruction and can extend up to 13 cycles in the presence of data hazards and contention [36]. This causes data hazards and large slowdowns when such instructions are as commonly used as our queue operations. We redesigned Rocket's interface to move it into the MA stage[6] of the pipeline (figure 6), multiplexing between the ISAX unit and the load-store unit.

IV. Evaluation

Experimental Setup To comprehensively evaluate the feasibility of FireGuard, we built FireGuard into RISC-V cores. BOOM main cores were augmented with data-forwarding channels, filters, and mappers; Rocket μcores were configured as security engines, running different safeguards, i.e., kernels. We implemented the microarchitecture with

[6]The MA stage was chosen as it is the first stage in the pipeline that is non-speculative, simplifying the implementation of the state-destructive `pop` but otherwise minimizing hazards, such that only one bubble is required for an instruction immediately following the custom instruction that uses its output.



(a) FireGuard v.s. software techniques (b) Combining safeguards

Fig. 7: Performance results for FireGuard *(4 µcores or 1 HA for each kernel)* running Parsec with different/combined safeguards *(AS: AddressSanitizer; SS: Shadow Stack; in (b), SS is implemented as a HA when three guardian kernels are deployed).*

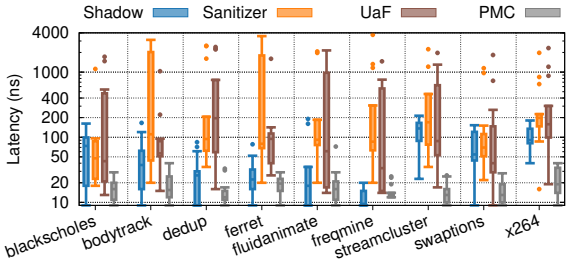


Fig. 8: Detection latency while using 4 µcores *(unit: ns).*

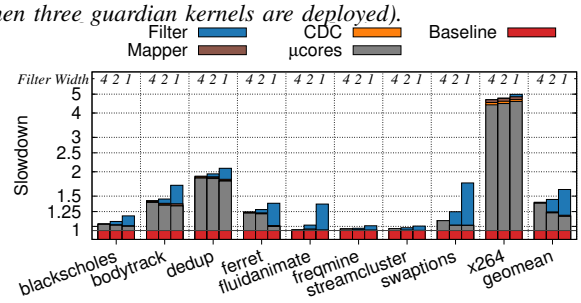


Fig. 9: Cumulative bottlenecks v.s. filter widths (on 4-µcore Sanitizer), measured by the proportion of time queues are full.

Chisel (v3.4) and synthesized the RTL using Vivado toolchains (v2021.2). The generated netlist was deployed on Virtex UltraScale+ FPGAs using FireSim [27], [37], emulating the setup in table II.

We booted Linux (with kernel v5.7.0) and executed Parsec [38], running the simmedium dataset with kernels: Custom Performance Counter with bounds check (PMC) [21], AddressSanitizer [29], Use-after-Free detector (UaF)[7] and shadow stack [28], [39].

A. Performance Overhead

Figure 7(a) shows the slowdown experienced by the main core while executing kernels in FireGuard, in comparison to software-based counterparts (implemented via LLVM). In all instances, FireGuard is configured with four µcores. Hardware Accelerators (HAs) within FireGuard are also presented for PMC and shadow stack.

Using 4 µcores is sufficient to run a PMC with overheads of 2.5% geomean, shadow stack at 2.1%, AddressSanitizer at 39%, or UaF at 42%. By replacing the µcores with a single HA, the overhead of PMC and shadow stack can be reduced to 0%. Software techniques give 7.9% overhead for shadow stack, and 163.5% for AArch64 and 91.5% for x86-64 architectures on AddressSanitizer[8]. FireGuard consistently outperforms software techniques, with the exception of x264 with AddressSanitizer, and Dedup on UaF. The former is due to the exceptionally high volume of load and store instructions executed in x264, where four µcores fail to keep up to such a degree that sharing compute resources with the compute core, as in software schemes, is favorable (though FireGuard can do better with more µcores, see section IV-D). The latter is caused by DangSan [40] being more suited to dedup's allocation patterns than our UaF detector's underlying Minesweeper [30], where DangSan itself on FireGuard would give false negatives, as it relies on zeroing pointers, which may occur only after the UaF access if offloaded to a µcore.

Combining kernels. Figure 7(b) reports the slowdown experienced by the main core while executing multiple kernels simultaneously. The kernel that incurs the most slowdown dominates the performance, but slowdowns are not multiplied. This is benefited from the parallel execution of the kernels. When one kernel halts the execution of the main core, decreasing the number of instructions, other kernels can persist in their function with reduced computational overhead.

B. Detection Latency

To examine the detection latency, we inject erroneous input at various locations in the core, e.g., the jump unit, the LDQ and the STQ, etc, simulating e.g., jump to a hijacked PC and access a freed memory address. For each workload, 50–100 attacks are generated, and the detection latencies are shown in figure 8. PMC experiences the lowest detection latency, consistently detecting attacks under 50 *ns*. Due to its complex block parallelism model [21], the shadow stack sees marginally higher latencies than PMC. In the worst-case scenario (x264), a latency of 220ns is observed. For AddressSanitizer, though the median is always <200ns, the tail exceeds 2,000ns. This occurs when TLB and cache misses co-occur on many accesses in the same queue: our accurate modeling of TLB misses in FireSim causes worst-case delays to be higher than in the original work [21].

C. Analysis of Microarchitecture Bottlenecks

Figure 9 decomposes overheads in a 4-µcore FireGuard when running AddressSanitizer using varying event-filter widths (1, 2, or 4), highlighting the accumulated bottlenecks stemming from different elements. A 4-width event filter is as wide as the core's commit, so can always keep up. As the width decreases, overhead increases significantly, from 16% at 2-width to 34% at 1-width. The geomean overhead from the filter jumps from 16% (2-width) to 34% (1-width).

D. Analysis of Scalability

Figure 10 shows the slowdown with varying µcore count.

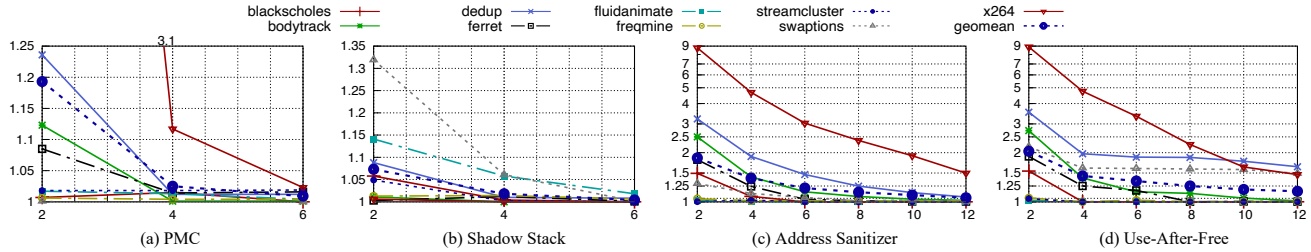
PMC. Figure 10(a) shows that using two µcores to perform PMC leads to a 20% geomean slowdown. Four µcores reduce this to 2%; yet, x264 still has a 17% slowdown. When scaling the µcores number to six, all workloads can be run with an overhead of less than 5%.

Shadow stack. Figure 10(b) shows that using two µcores for executing shadow stack leads to 7.3% slowdown (geomean), which is comparable to the performance of LLVM's software mechanism. With four µcores, the geomean overhead is reduced to 2.1%, decreasing to 0.4% when using six µcores. With six µcores, all workloads can be executed with <1% slowdown, except fluidanimate (1.8%).

AddressSanitizer. Figure 10(c) is heavier than (a) or (b). Using two µcores results in 86% overhead, still outperforming software (section IV-A). However, three workloads (bodytrack, dedup, x264)

[7]Our design takes MineSweeper [30] and uses analysis engines to find loads and stores to quarantined regions, to find as well as prevent bugs.

[8]AddressSanitizer is not supported for RISC-V in either GCC or LLVM, so the comparison focuses on AArch64 and x86-64. For shadow stack, support is absent for both RISC-V and x86-64, so we compare against AArch64.



(a) PMC (b) Shadow Stack (c) Address Sanitizer (d) Use-After-Free

Fig. 10: Slowdown when using varying numbers of μcores for guardian kernels *(x-axis: number of μcores; y-axis: slowdown).*

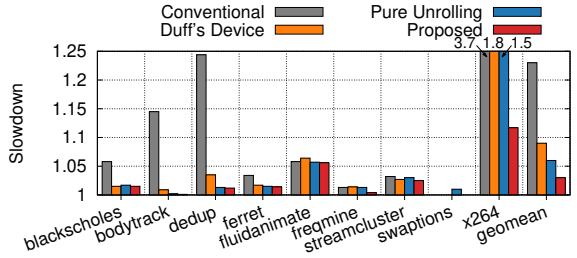


Fig. 11: Slowdown of programming models (4-μcore PMC).

exhibit over 100% overhead. This is due to the extremely high volume of load and store instructions that necessitate analysis. Even with 12 µcores, x264 continues to endure a 58.9% slowdown, while all other workloads see negligible overhead (less than 1%).

UaF. Figure 10(d) is the heaviest workload. As well as monitoring all loads and stores, the extra work to quarantine and release memory allocations only when safe [30] causes some overheads that do not parallelize away; for example, dedup sees flat overheads due to extra memory-allocation work. The geomean reaches 1.16× by 12 µcores, lower than the 1.6× that is achieved by software mechanisms [40].

E. Analysis of Programming Models

Figure 11 shows the performance overhead when executing PMC using different programming models (section III-D) with 4 μcores. By being aware of the remaining hazards in the μcores, we can improve performance significantly. A conventional single-iteration loop suffers from frequent data hazards due to both queue `count` checks and `pop` instructions. Duff's device [41] reduces hazards on size checks, made possible by the queue `count` instruction, and pure unrolling helps further, by removing hazards for `pop` instructions if the queue is relatively full. A hybrid strategy, using unrolling when possible and Duff's device otherwise, is uniformly best.

F. Hardware Overhead

We study hardware overhead of the microarchitecture by performing a physical implementation of a 4-µcore FireGuard at the post-layout stage using Synopsys 14*nm* Generic PDKs. The RTL is synthesized with Design Compiler (v2022.12), and the netlist is placed and routed via IC Compiler 2 (v2022.12).

The area of the SoC is 2.91mm^2, where the BOOM is 1.107mm^2 and each Rocket is 0.061mm^2. The total area of FireGuard's transport mechanisms is 0.043mm^2 (3.88% and 1.48% of the BOOM and the SoC): the filter occupies 0.032mm^2 and the mapper 0.011mm^2. Building a 4-µcore FireGuard upon a BOOM thus needs 0.287mm^2, i.e., 25.9% and 9.86% of the BOOM and the SoC.

Intuitively, this observation diverges from the goal of low-overhead design. But, this is only because the prototype BOOM cores are small compared with commercial designs, yet the Rocket cores are very large, with closed-source in-order 64-bit cores available using half the resources [17]. To provide more comprehensive insights, we present a detailed analysis below on real-world SoCs.

G. Feasibility Analysis

We compare *performance-area trade-offs* between BOOM and the OoO processors in commercial SoCs [42], [43], [45]–[47], [47], [47], [48], and examine the overhead associated while constructing

Processors (Performance Core only)				
Core (SoC)	**BOOM [26]**	**FireStorm [42] (M1-Pro)**	**Cortex-A76 [43] (Kirin-960)**	**AlderLake-S [44] (i7-12700F)**
Peak Freq.	3.2GHz	3.2GHz	2.8GHz	4.9GHz
Tech.	14nm	5nm	7nm	10nm
Area	1.11mm^2	2.53mm^2	1.23mm^2	7.30mm^2
@14nm	1.11mm^2	22.55mm^2	3.61mm^2	22.63mm^2
IPC	1.3	3.79	2.07	2.83
Normalized Throughput	**1**	**2.92**	**1.27**	**3.35**
FireGuard Elements				
Filter Width	4-way	8-way	4-way	6-way
#µcores	4	12	5	13
Overhead (% / Core)	**0.29**mm^2 **(25.9%)**	**0.81**mm^2 **(3.6%)**	**0.35**mm^2 **(9.6%)**	**0.85**mm^2 **(3.8%)**
An Independent Kernel for All Cores				
Overhead (% / SoC)	**0.29**mm^2 **(9.86%)**	**6.10**mm^2 **(0.47%)**	**1.23**mm^2 **(0.57%)**	**6.67**mm^2 **(0.99%)**

TABLE III: Feasibility of FireGuard in commercial SoCs.

FireGuard upon these processors and SoCs. To do so, we first estimate the core sizes from die shots of each system [44]–[46], then normalize the area using 14 *nm* technology based on the density difference [49]. We then scale up the number of μcores to match the increased IPC and clock relative to BOOM (calculated using IPC × peak frequency), where the IPC is measured through single-thread Parsec execution.

CPU-level overhead. Table III's upper portion examines throughput and area between BOOM and performance cores in the SoCs; the middle portion estimates the overhead to support a kernel, while attaining the performance in section IV-A.

Modern OoO cores are significantly larger than BOOM. To achieve a linear increase in processor throughput, a superlinear increase in hardware overhead is required. FireStorm is 2.92× faster than BOOM, but each core consumes 20.3× the area. While building FireGuard with a bigger core would require extra µcores to keep up with the execution, the increase is only linear. Building FireGuard upon FireStorm, Cortex-A76, and AlderLake-S requires 12, 5, and 13 µcores respectively, giving 3.6%, 9.6%, and 3.8% overhead/core.

SoC-level overhead. To ensure global security, FireGuard elements must be equipped for all processors in the SoC. Hence, we apply the same analysis on all core types in the SoCs and report the overhead.

Table III's bottom portion indicates that integrating a kernel in M1-Pro, Kirin-980, and i7-12700F leads to an overhead of less than 1%. This negligible impact makes implementing several kernels per-core practical. Moreover, the energy overhead would be even lower, since the majority of FireGuard operates within a low-frequency domain.

V. Conclusion

We have presented FireGuard, a microarchitecture for fine-grained instruction analysis. To make the design practical for deployment, we presented a buffer-free data forwarding channel, a superscalar event filter and a broadcast-free mapper. Feasibility analysis shows FireGuard can be integrated into modern SoCs with less than 1% increase in area. In summary, comprehensive in-core analysis is practical and efficient to build into real cores and SoCs.

VI. Acknowledgement

We appreciate the anonymous reviewers for their helpful feedback. This work is supported by the National Key Research and Development Program (Grant No. 2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086, 62204036) and the Basic Research Program of Jiangsu (Grants No. BK20243042).

References

[1] A. Burns and R. I. Davis, "A survey of research into mixed criticality systems," vol. 50, no. 6. ACM New York, NY, USA, 2017, pp. 1–37.

[2] J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe, "The cheri capability model: Revisiting risc in an age of risk," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 457–468, 2014.

[3] Zecar, in *https://zecar.com/reviews/*, 2023.

[4] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *20th USENIX security symposium (USENIX Security 11)*, 2011.

[5] T. Wehbe, V. Mooney, and D. Keezer, "Hardware-based run-time code integrity in embedded devices," *Cryptography*, vol. 2, no. 3, p. 20, 2018.

[6] A. P. Felt, E. Chin, S. Hanna, D. Song, and D. Wagner, "Android permissions demystified," in *Proceedings of the 18th ACM conference on Computer and communications security*, 2011, pp. 627–638.

[7] S. Bannister, "Memory tagging extension," 2019.

[8] X. Vera, "Inside tiger lake: Intel's next generation mobile client cpu," in *Hot Chips Symposium*, 2020.

[9] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, "Ensemble learning for low-level hardware-supported malware detection," in *Research in Attacks, Intrusions, and Defenses: 18th International Symposium, RAID 2015, Kyoto, Japan, November 2-4, 2015. Proceedings*. Springer, 2015, pp. 3–25.

[10] M. Milenković, A. Milenković, and E. Jovanov, "Hardware support for code integrity in embedded processors," in *Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*, 2005, pp. 55–65.

[11] T. Nyman, G. Dessouky, S. Zeitouni, A. Lehikoinen, A. Paverd, N. Asokan, and A.-R. Sadeghi, "Hardscope: Thwarting dop with hardware-assisted run-time scope enforcement," 2017.

[12] D. Kuzhiyelil, P. Zieris, M. Kadar, S. Tverdyshev, and G. Fohler, "Towards transparent control-flow integrity in safety-critical systems," in *International Conference on Information Security*. Springer, 2020, pp. 290–311.

[13] J. Devietti, C. Blundell, M. M. Martin, and S. Zdancewic, "Hardbound: Architectural support for spatial safety of the c programming language," vol. 42, no. 2. ACM New York, NY, USA, 2008, pp. 103–114.

[14] S. M. A. Zeinolabedin, J. Partzsch, and C. Mayr, "Real-time hardware implementation of arm coresight trace decoder," *IEEE Design & Test*, vol. 38, no. 1, pp. 69–77, 2020.

[15] J. Zhang, B. Qi, Z. Qin, and G. Qu, "Hcic: Hardware-assisted control-flow integrity checking." IEEE, 2018.

[16] M. Ozsoy, C. Donovick, I. Gorelik, N. Abu-Ghazaleh, and D. Ponomarev, "Malware-aware processors: A framework for efficient online malware detection," in *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2015, pp. 651–661.

[17] ARM BTI, in *hhttps://developer.arm.com/documentation/100748/latest/Security-features-supported-in-Arm-Compiler-for-Embedded/*, 2023.

[18] Intel LAM, https://lpc.events/event/11/contributions/LAM-LPC-2021.pdf, 2021.

[19] Intel CET, 2023.

[20] N. Christoulakis, G. Christou, E. Athanasopoulos, and S. Ioannidis, "Hcfi: Hardware-enforced control-flow integrity," in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, 2016, pp. 38–49.

[21] S. Ainsworth and T. M. Jones, "The guardian council: Parallel programmable hardware security," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1277–1293.

[22] S. Fytraki, E. Vlachos, O. Kocberber, B. Falsafi, and B. Grot, "Fade: A programmable filtering accelerator for instruction-grain monitoring," in *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2014, pp. 108–119.

[23] D. Y. Deng, D. Lo, G. Malysa, S. Schneider, and G. E. Suh, "Flexible and efficient instruction-grained run-time monitoring using on-chip reconfigurable fabric," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2010, pp. 137–148.

[24] S. Chen, M. Kozuch, T. Strigkos, B. Falsafi, P. B. Gibbons, T. C. Mowry, V. Ramachandran, O. Ruwase, M. Ryan, and E. Vlachos, "Flexible hardware acceleration for instruction-grain program monitoring," *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, pp. 377–388, 2008.

[25] X. Ge, W. Cui, and T. Jaeger, "Griffin: Guarding control flows using intel processor trace," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 585–598, 2017.

[26] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, 2020, pp. 1–7.

[27] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra *et al.*, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 29–42.

[28] M. Abadi, M. Budiu, U. Erlingsson, and J. Ligatti, "Control-flow integrity principles, implementations, and applications," *ACM Transactions on Information and System Security (TISSEC)*, vol. 13, no. 1, pp. 1–40, 2009.

[29] K. Serebryany, D. Bruening, A. Potapenko, and D. Vyukov, "Address-Sanitizer: A fast address sanity checker," in *2012 USENIX annual technical conference (USENIX ATC 12)*, 2012, pp. 309–318.

[30] M. Erdős, S. Ainsworth, and T. M. Jones, "Minesweeper: a "clean sweep" for drop-in use-after-free prevention," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 212–225.

[31] A. Ankur, A. Pandya, A. Abu, and L. Young-Uhg, "Noc architecture design methodology," *Journal of the Korea Institute of Information and Communication Engineering*, vol. 10, no. 1, pp. 57–64, 2006.

[32] M. Schoeberl, S. Abbaspour, B. Akesson, N. Audsley, R. Capasso, J. Garside, K. Goossens, S. Goossens, S. Hansen, R. Heckmann *et al.*, "T-crest: Time-predictable multi-core architecture for embedded systems," *Journal of Systems Architecture*, vol. 61, no. 9, pp. 449–471, 2015.

[33] Z. Jiang, N. C. Audsley, and P. Dong, "Bluevisor: A scalable real-time hardware hypervisor for many-core embedded systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 75–84.

[34] Z. Jiang, K. Yang, N. Audsley, N. Fisher, W. Shi, and Z. Dong, "Bluescale: a scalable memory architecture for predictable real-time computing on highly integrated socs," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1261–1266.

[35] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[36] P. Davide, "Design and programming of a coprocessor for a risc-v architecture," Master's thesis, Politecnico di Torino, 2017.

[37] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[38] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[39] LLVM, 2023.

[40] E. Van Der Kouwe, V. Nigade, and C. Giuffrida, "Dangsan: Scalable use-after-free detection," in *Proceedings of the Twelfth European Conference on Computer Systems*, 2017, pp. 405–419.

[41] T. Duff, "Duff's device," 1988.

[42] D. Johnson, "Firestorm pipeline overview," in *https://dougallj.github.io/applecpu/firestorm.html*, 2022.

[43] ARM, "Cortex-a76," in *https://developer.arm.com/Processors/Cortex-A76*, 2018.

[44] Intel, "Alder lake: Microarchitecture," in *https://en.wikichip.org/wiki/intel/microarchitectures/alder_lake*, 2022.

[45] "Apple m1 pro die shot," in *https://www.anandtech.com/show/17019/apple-announced-m1-pro-m1-max-giant-new-socs-with-allout-performance*, 2021.

[46] "Kirin 980 die shot," in *https://www.anandtech.com/show/13564/chiprebel-releases-kirin-980-die-shot-cortex-a76s-mali-g76-in-view*, 2018.

[47] E. Rotem, A. Yoaz, L. Rappoport, S. J. Robinson, J. Y. Mandelblat, A. Gihon, E. Weissmann, R. Chabukswar *et al.*, "Intel alder lake cpu architectures," *IEEE Micro*, vol. 42, no. 3, pp. 13–19, 2022.

[48] Intel, "Alder lake: Specification," in *https://www.intel.co.uk/content/www/uk/en/products/platforms/details/alder-lake-s.html*, 2022.

[49] K. Kumawat, "7nm vs 10nm vs 14nm: Fabrication process," in *https://www.techcenturion.com/7nm-10nm-14nm-fabrication*, 2019.

AutoSkewBMT: Autonomously Synthesizing Optimized Integrity Authentication Mechanism for DNN Accelerators

Rakin Muhammad Shadab*, Sanjay Gandham†, Mingjie Lin‡
Department of ECE
University of Central Florida
Email: *ra397343@ucf.edu, †sanjay.gandham@ucf.edu, ‡Mingjie.Lin@ucf.edu

***Abstract*—As domain-specific accelerators for deep neural network (DNN) inference gain popularity due to their performance and flexibility advantages over general-purpose systems, the security of accelerator data in memory has emerged as a significant concern. However, the overhead associated with standard memory security measures, such as encryption and integrity authentication, presents a major challenge for accelerators, particularly given the high throughput demands of typical DNN applications [1].**

In this work, we present AutoSkewBMT, a security framework that autonomously generates optimized integrity system configurations to enhance the Bonsai Merkle Tree (BMT)-based integrity authentication workflow for DNN accelerators. The framework leverages a novel and efficient design space generation algorithm to optimally skew the BMT for specific workloads. Configurations generated by AutoSkewBMT outperform recent state-of-the-art solutions by up to 32% on general DNN workloads.

***Index Terms*—DNN, accelerator, BMT, framework**

I. Introduction

The age of Artificial Intelligence (AI) is upon us as the ever-growing, multi-faceted applications of machine learning and AI have brought about unforeseen levels of automation and advancements in different domains. In order to facilitate such massive growth, the hardware to accelerate machine learning and deep learning workflow has also gone through rapid development in the recent years. State-of-the-art, compute-focused Graphics Processing Units (GPUs) such as Nvidia's H100 graphics card can drive large-scale Deep Neural Network (DNN) models in servers and consumer-targeted spaces [2]. However, it might be impractical to use such massively sized GPU or GPU-like accelerators for all use-cases. For specialized applications that require domain-specific acceleration (including embedded and edge AI), it would be more imperative to use FPGA-driven DNN accelerators as they would typically perform better than GPUs in real-time inference, while also being a better fit for low-power, resource-constrained systems [3].

However, these custom DNN accelerators are susceptible to the same memory security exploits that typically affect the general-purpose computers [1], [4]. The common adversaries such as the confidentiality and integrity-focused threats can steal important information from the memory through a shared memory bus [5], [6]. The state-of-the-art techniques to mitigate these vulnerabilities such as AES encryption and BMT-based authentication (Fig. 1), provide strong security assurance but also incur high performance overhead [7]. Therefore, it is important to find the best balance between performance and security features, especially for accelerators as they need very high throughput for fast inference.

The contributions in this paper are mentioned below:

- This work proposes a new tool chain, AutoSkewBMT, for optimizing the integrity verification process in a secure DNN accelerator. Generally the integrity check is performed through an integrity tree such as BMT-based authentication [8], [9]. We strive to reduce the overhead of such authentication with autonomous generation of the optimal configurations for such an integrity protection system for specific DNN workloads.
- As the accelerators operate with a fixed data access pattern, it is useful to skew the BMT to take advantage of the more-frequently accessed nodes. However, the skewing of BMT can be tricky especially for larger trees as the overhead might vary drastically for different skew patterns. AutoSkewBMT generates an optimally skewed BMT for an accelerator workload through the use of an efficient and novel design space generation & exploration algorithm based on Huffman encoding [10].
- AutoSkewBMT tool includes a counter access pattern generator that creates the metadata access patterns directly from the DNN model attributes and the accelerator parameters.

We limit ourselves to only BMT (excluding other integrity trees such as tree of counters (ToC) or parallel Merkle tree) for this work due to its extensive use in most of the contemporary literature [7], [11].

II. Background & Motivation

The memory-based vulnerabilities in a computing system can largely be divided into confidentiality and integrity-based attacks [11], [12]. The confidentiality-based threats attempt to directly access the user data over the memory bus [7]. The integrity-based threats, on the other hand, can be one of the three attacks – splicing, spoofing or memory replay – that overwrites an existing data block with another block of redundant data (Fig. 2) with spatial or temporal locality in the memory [6], [9]. The confidentiality of the memory contents can be protected by encrypting the entire memory with a state-of-the-art

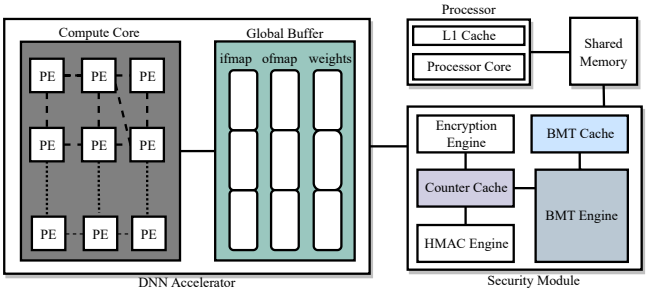


Fig. 1. A secure DNN accelerator with confidentiality and integrity protection.

encryption technique, such as the AES CTR-mode encryption [5], [8].

The AES CTR-mode encryption breaks down encryption metadata i.e., encryption counters into multiple smaller counters to generate a one-time pad (OTP) for the encryption of the data [11]. These counters or the encryption metadata are protected from the integrity threats by an integrity tree (Fig. 2). One of the most popular choice for the integrity tree is Bonsai Merkle Tree (BMT) as it has a low memory overhead and high flexibility of implementation [7], [9],

[13]. In such cases, the encrypted data can be integrity-protected through the use of a separate hash-based Message Authentication Code (HMAC) scheme [7], [11]. All of these security checks are performed within a trusted execution environment (TEE) outside the memory [9], [14].

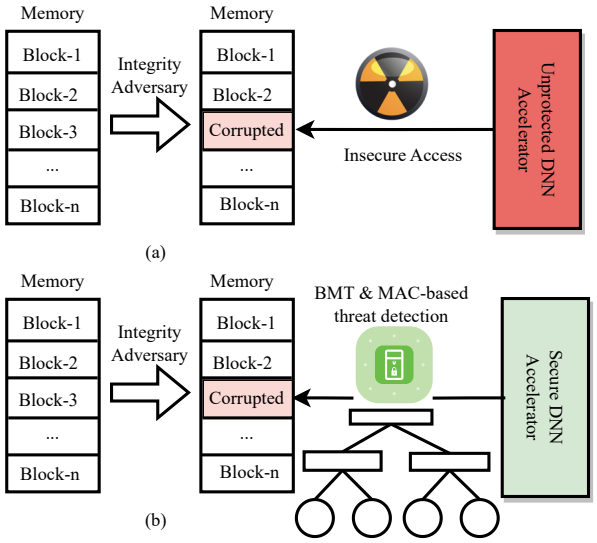


Fig. 2. (a) An unprotected DNN accelerator affected by integrity adversary, & (b) the same attack is detected by BMT & MAC in a secure accelerator.

TABLE I
COMPARISON BETWEEN RECENT SECURE DNN ACCELERATORS

	Confidentiality, Integrity	TEE	Tree/VN
Chiron [15]	✓	3rd-party	Tree
Occlumency [16]	✓	3rd-party	Tree
GuardNN [4]	✓	Custom	VN
TNPU [17]	✓	Custom	VN
Securator [1]	✓	Custom	VN

Since the DNN accelerators don't have enough on-chip memory to store the entire feature maps or weights, they must use the main memory (typically shared with the processor) for the data and instructions and as such, the same security threats also apply to the them [1]. Therefore, they need similar TEE-based protection as the general-purpose systems, either from a third-party implementation [15], [16], [18] or a custom TEE [4], [17], [19]. However, the third-party TEE solutions (such as Intel SGX) are typically associated with high overhead and low flexibility/customization [1], [19]. As a result, more and more recent secure accelerators such as Securator [1], GuardNN [4] & TNPU [17] use their own, custom hardware-assisted TEEs. Since the accelerators are typically more bottlenecked by memory bandwidth than processors, these custom TEEs generally exclude the direct use of BMT or other integrity trees as they can incur high overhead for specific access patterns. Instead, they use alternate techniques such as Version Number (VN) to preserve integrity [1], [17]. But they either still need third-party integrity tree or additional TEE and software support from the processor to protect the VN (i.e., GuardNN & TNPU) and/or mandate close integration between the accelerator architecture and security module design, even embedding some security functionalities in the accelerator's compute core (i.e., Securator). We strive to address this problem by allowing for a flexible, accelerator-independent security module (as seen in Fig. 1) and introducing a tool that can determine an optimized BMT subsystem configuration for a given accelerator to ensure high memory bandwidth.

In the threat model for this work, we assume the DNN accelerator, host processor & the security module to reside within a TEE. As such, they are considered secure. The memory is outside the TEE and therefore, is susceptible to the confidentiality & integrity attacks. We only consider the security of the accelerator data in this work and don't consider the use of non-volatile memory (NVM). Our target platform is FPGA due to its flexibility of re-configuration.

III. OVERHEAD IN BMT-BASED PROTECTION

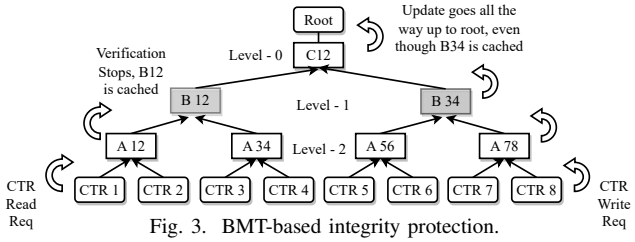


Fig. 3. BMT-based integrity protection.

The security module to prevent memory confidentiality and integrity-based attacks, as seen in Fig. 1, usually consists of AES encryption engine, a HMAC engine, a counter cache and a BMT engine + its associated BMT cache. Every memory access from the DNN accelerator must go through the security module for the security purposes. A BMT is constructed from the encryption counters where each of the counter values are hashed and then the results are truncated and concatenated to form a parent node. This process is repeated in the higher levels until the tree reaches only a single parent – root node. The number of children nodes for each parent node is determined by the so called '-arity' of the tree. For example, every parent node in a binary BMT (as shown in Fig. 3) has two children. The entire BMT except the root is stored in the memory (root is stored in a secure register).

Both the encryption and HMAC engines rely on the encryption metadata such as counters for the encryption and HMAC generation. A secure, on-chip counter cache is accessed to read/write a relevant counter value (Fig. 1). If the request to the counter cache is hit, then BMT authentication of the counter is skipped as the contents in the counter cache is considered secure. Otherwise, the counter is read from the memory, authenticated through BMT engine and then stored in the counter cache. In such cases, the counter request is forwarded to BMT engine that fetches the counter from the memory and verifies its integrity by re-generating its parent through hashing. If the parent node is not cached in the BMT cache and needs a memory access, then the parent needs to be verified as well. This triggers a chain verification process that only stops either at the root node or at a cached node (Fig. 3). We assume a dedicated BMT cache for each level of the BMT to enhance performance and reduce recursiveness in the BMT algorithms [7], [20]. Once the integrity is verified, the specific counter is then stored in the counter cache and can be used for future operations without re-verification unless it is evicted and read from the memory again. If the counter request is a write request, then the counter is incremented (and authenticated through BMT first, if missed in the cache) and its corresponding BMT chain also needs to be updated. This update can be done immediately/eagerly or in a relaxed manner [7], [9], but we will only consider the eager update

process in this work due to its widespread usage. The eager-update technique always updates the entire chain regardless of the caching status of the BMT nodes (Fig. 3). The BMT authentication and update process, while thwarting the previously mentioned integrity attacks, also incurs significant overhead.

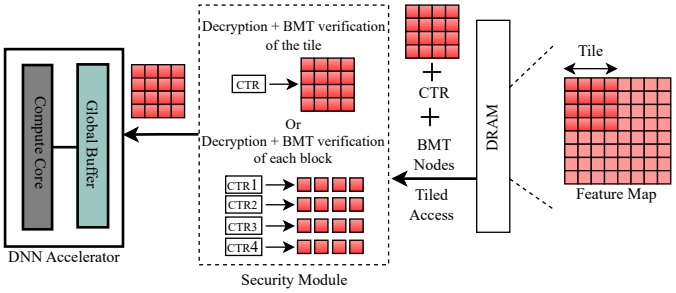


Fig. 4. Tile-level vs block-level integrity authentication.

DNN accelerators use tiling technique to access feature maps and weights of the DNN model from the memory. While utilizing tiled accesses, the accelerator only fetches a section of the feature map in its global buffer, known as a tile. This is done due to the local buffer size constraints on the accelerator. An integrity-protected accelerator (such as an accelarator with BMT protection) with tiled accesses can use encryption counters to protect the entire tile (tile-level integrity protection) or they can be used to encrypt and protect individual data blocks (block-level integrity protection). The tile-level integrity protection can lower the integrity overhead since only one counter protects the entire tile. In such cases, the counter is only updated once all the data blocks within the tile are written or updated. During tiled data read, only one counter needs to be verified by the BMT since each counter would protect an entire tile (Fig. 4). However, this type of protection is not crash-consistent and cannot provide security guarantee for every block in the tile during a system crash/power failure if the system memory is Non-Volatile Memory (NVM). On the other hand, the block-level integrity protection is the same type of protection used in general computing systems. While this technique can cause additional overhead due to the update/authentication of individual blocks, it is flexible and can be used with any type of memory.

Since the DNN accelerators operate on a fixed and predictable data access pattern, the BMT-authentication and update process can benefit from making the verification and update path of some of the most commonly accessed counters shorter compared to other, less frequently-accessed counters. This is known as the skewing of the BMT (Fig. 5). Let's assume that each of the counters except $CTR4$ and $CTR8$ have an access frequency of 2 (i.e, they are accessed twice) and $CTR4$ and $CTR8$ are more frequently accessed with an access frequency of 8. If the accesses are write accesses, then the balanced BMT (Fig. 3) will need a hash operation at each level, totalling 4 hashes (including root) for a write to a counter. Therefore, the total number of hash operations for the balanced BMT = 6*4*2 + 2*4*8 = 112. On the other hand, the same scenario for the skewed BMT in Fig. 5 requires (4*5*2 + 2*4*2 + 2*2*8) = 88 hash operations.

Even though FAST [21] already used skewed BMT for processors, it didn't use a combined counter access pattern generator in the tool chain and instead collected the counter access traces for each application by separately simulating them. It also didn't use a BMT cache and relied on the use of direct BMT node access from the memory. Moreover, the memory access patterns and the granularity of

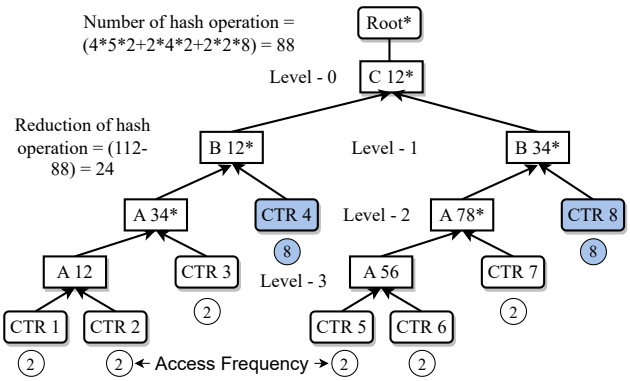


Fig. 5. An example of a skewed BMT with the same counters as in Fig. 3. Here, CTR4 and CTR8 are the most frequently accessed and placed closer to the root. The * sign indicates a change in the hash value compared to the balanced BMT shown in Fig. 3.

integrity authentication in DNN accelerators depend not only on the workload but also on the accelerator design features such as tiling (as shown in Fig. 4). FAST algorithm doesn't account for these scenarios as it is only targeted for general-purpose applications.

In the case of a DNN accelerator, the parameters are fixed at the system initialization, the memory accesses are also fixed for a given DNN model. Therefore, the skewed BMT configuration will also be constant for that specific accelerator workload. As such, we propose a security framework, AutoSkewBMT, that determines a skewed BMT configuration for a set of accelerator parameters and a fixed DNN model. The counter access pattern is also generated automatically from the DNN model attributes and accelerator parameters, eliminating the need to determine it separately by running inference on an accelerator. If DNN model/workload changes, the skewed BMT will be re-skewed through the tool chain for the new workload instead of dynamic, run-time reformation.

IV. AutoSkewBMT: Methodology

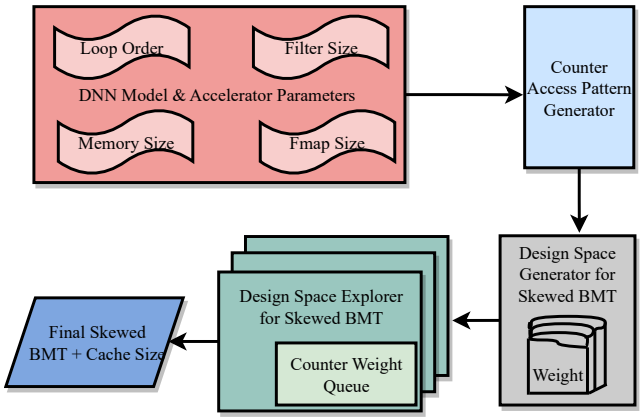


Fig. 6. AutoSkewBMT framework.

The AutoSkewBMT framework consists of three components - counter access pattern generator, design space generator and design space explorer (Fig. 6). The DNN model parameters such as the input and output feature map size, filter size, memory size and accelerator parameters such as global buffer size and loop order are provided as the input to the framework. These attributes are then used to generate the data access pattern for the accelerator data and then consequently, the resultant counter access pattern to encrypt/decrypt the data block.

The counter accesses are then fed into a design space generator that generates different weights for every counter, based on the frequency of read and write accesses. These weights are then passed over to the design space explorer to determine an optimized skewed BMT structure by placing the most frequently accessed counters closest to the root node. It also fixes an optimal BMT cache size at each tree level of the skewed BMT based on the number of nodes at each level. The output of our tool is a skewed BMT configuration optimized for a fixed accelerator design and a specific DNN workload.

Algorithm 1: Counter Access Pattern Generation

Input: Memory Size S, input size I_S, output size O_S, filter size F_S, loop order R, global buffer size B_S
Output: List of Counter Accesses, L_C
Parameters: Feature map dimension: x, y, z, tile dimension: t_x, t_y, t_z, filter co-efficient: n, initial memory address $base_{addr}$, current data address $addr$, initial counter address $base_{ctr}$, counter address ctr_{addr}, list index i

while $Access_{req} < S$ **do**
  Calculate t_x, t_y, t_z;
  Calculate $addr$ = $base_{addr} + x + (y * t_x) + (z * t_x * t_y) + (n * t_x * t_y * t_z)$;
  Update ctr_{addr} = $base_{ctr}$ + ($addr$/2^{12});
  Append $L_C[i]$ = ctr_{addr};
  Increment i;
  Increment $Access_{req}$;
  Update x, y, z, n, t_x, t_y, t_z;
end

Algorithm 2: Design Space Generation & Exploration

Input: List of Counter Accesses with the number of read and write request, L_C, size of the protected memory S, base address of the counter range, $base_{ctr}$
Output: Shape of the optimized skewed tree and the size of the BMT cache
Parameters: Number of children nodes for each parent or the -arity of the BMT, K, an empty weight queue, Q, a list for the weights of every access, W, list index, j, number of read accesses, ctr_{rd}, number of write accesses, ctr_{wr}, skewing index, p, skewing threshold, T, number of BMT levels, M, tile update/verification weight, $T_{u/a}$

for $L_C[i]$ *in* L_C **do**
  j = ctr_{addr} - $base_{ctr}$;
  $W[j]$ = $W[j]$ + (ctr_{rd} + (M+1) * ctr_{wr}) * $T_{u/a}$;
  Push $W[j]$ into Q;
end
while Q *is not empty &* p ¡ T **do**
  Pop K entries from Q;
  Update the new parent of the popped entries to be N;
  Push N with the popped entries;
  increment p;
end
Append the remaining nodes;
Fix cache size at each BMT level = total cache size * % of total nodes at the level;

The counter access pattern generator calculates the current data address, $addr$ from the accelerator parameters (Alg. 1). The accelerator typically accesses a feature map in a group of tiles. As the dimensions of the feature maps might vary for different layers of a DNN model, the tile dimensions would also change. As such, these dimensions will be different in accordance with the current layer of the DNN and the model itself. Once $addr$ is generated, the counter address, ctr_{addr} is calculated from the $addr$. The base address of the counter, ctr_{base} typically starts after the data and the HMAC address range in the memory. The counter address, ctr_{addr} is later appended into a list L_C that also includes the information on the request type (read/write). This list is forwarded to the design explorer tool at the end of the access pattern generation (Fig. 6).

The design space generator tool calculates a weight for each counter request based on the information provided by the access pattern generator. These weights are based on the worst-case scenario for a read or write operation. In the worst-case, a counter read request will result in a BMT verification chain that goes all the way up to the root since none of the nodes will be present in the BMT cache. However, once the nodes are read, the verification of all the nodes in the chain can be done concurrently using parallel hash engines, as seen in [14] & [20]. On the other hand, a counter write will always go through the entire update chain (Fig. 3) as we consider an eagerly updated tree. We assume a BMT with 5 levels (M = 5) for the context of this work, and as such, the counter write operation will require M + 1 = 6 consecutive, non-overlapped hash operations to update a node at each level, including the sixth level (root). However, the algorithm is valid for a BMT with any number of tree levels as M can be any valid integer number. In other words, it is scalable with the size of the tree.

We assign a weight of 1 to each read and a weight of M + 1 (6 in this case) to each write to a counter (Alg. 2). The total weight is then calculated by summing these two weights and multiplied with a weight of tile update/authentication granularity ($T_{u/a}$). $T_{u/a}$ is calculated by the ratio of total number of verification/updates required in a tile to total number of uncached counter accesses in a tile. For block-level integrity protection, this weight is simply going to be 1. The counters with the highest total weight will then be put closest to the root node in the final skewed BMT. These weights are used by the design space explorer tool to generate the skewed BMT. The weights are stored in a priority queue (called weight queue) and their parents are updated accordingly to skew the verification/update chain (Alg. 2). Based on a skewing threshold, the number of counter nodes being pushed up is decided and all the remaining counters are placed at the bottom of the tree while the rest of the nodes are appended. The size of the cache at each level of the BMT is then fixed, based on the total percentage of nodes at that level. The algorithm for design space generation & exploration (Alg. 2) is inspired by popular Huffman encoding [10], similar to FAST [21].

The AutoSkewBMT toolchain relies on memory access pattern and parameters like feature map size and the number of accesses in a tile. But it doesn't have any dependency on the actual type of computation in the each layer of the DNN model (i.e., convolution or matrix-matrix multiplication) or the computation techniques in the accelerator. Therefore, it is highly flexible and can work with any type of DNN models and accelerators if the aforementioned parameters are known.

Our proposed work can be used for both tile-level and block-level integrity granularity. However, for the scope and evaluation of this work, we only use block-level granularity.

V. Consistency of Security Guarantee with AutoSkewBMT

The skewed BMT generated by AutoSkewBMT framework follows the same authentication and update algorithm as a regular BMT. The only difference is that the values of the parents for the nodes that are moved to higher levels will be different than the original BMT as the accompanying nodes under the same parent will be different from the original orientation (as seen in Fig. 5). However, this change in the hash values within an authentication/update chain doesn't violate the security assurance as the root will also be updated accordingly and the entire authentication/update chain will maintain the consistency with the new values. In other words, the security guarantee in the skewed BMT will remain the same as the regular BMT.

VI. Evaluation

We compare three different accelerator systems – a system with no security protection (baseline), one with both encryption-based confidentiality protection and a balanced BMT-based integrity protection and finally the system with encryption and skewed BMT (configured by AutoSkewBMT) – against each other. We also include two other state-of-the-art solutions – GuadNN [4] & TNPU [17] – in the mix for the performance analysis. We consider the use of a 5-level balanced BMT that is skewed for different access patterns by the AutoSkewBMT tool. Both the counter cache and the BMT caches are direct-mapped caches. We utilize a state-of-the-art BMT engine architecture (similar to HERMES [14]). The configuration of the security module is elaborated on Tab. II.

TABLE II
Security Module Configuration

Encryption, MAC, Counter cache & BMT engine details	
Protected memory size	128MB
Encryption	AES-128
MAC	HMAC (SHA-256 Hash)
Data granularity	64B
HMAC block size	64B
BMT parameters	5-level, 8-ary
Counter cache size	32KB
BMT cache size	40.3KB (32KB+4KB+4KB+128B+128B)

The security blocks are synthesized for Xilinx U200 FPGA platform with Vivado tool(Tab. III). The BMT engine designed for the integrity authentication with the resultant skewed BMT configuration from AutoSkewBMT incurs only a 3% overhead for the Look-Up Table (LUT) usage and a negligible overhead of flip-flop usage over the BMT engine for the regular, balanced BMT. Our energy analysis shows less than 2% on-chip power increase for the system configured by the AutoSkewBMT over the balanced BMT. We also synthesized the security modules for ASIC using Synopsys DC and found similar 3% overhead for the skewed BMT engine.

We compare the performance of all three of our systems with TNPU [17] & GuardNN [4]. The optimized and shortened BMT authentication chain for most frequently accessed counters allows the AutoSkewBMT system to exhibit up to 7% better performance compared to TNPU and 8% better performance over the balanced BMT in widely used DNN workloads (Fig. 7). TNPU depends on a third-party integrity tree (such as Intel SGX) support for the protection of its integrity metadata [17] and exhibits some overhead because of it. Our system also outperforms GuardNN by up to 32% margin as GuardNN suffers from additional overhead caused by a software-based scheduler and processor TEE for its integrity mechanism [4].

TABLE III
Hardware synthesis results

	LUT	Flip-flop	BRAM
Encryption Engine	36997(3.1%)	13038(0.6%)	0(0.0%)
HMAC Engine	6205(0.5%)	6133(0.3%)	0(0.0%)
Counter Cache	3391(0.3%)	4019(0.2%)	8.5(0.0%)
BMT Engine	137650 (11.7%)	158300 (6.7%)	14.5 (0.7%)
Skewed BMT Engine	172540 (14.7%)	164700 (6.9%)	14.5 (0.7%)
BMT Cache (Total)	48912 (4.1%)	40447 (1.71%)	128 (5.9%)

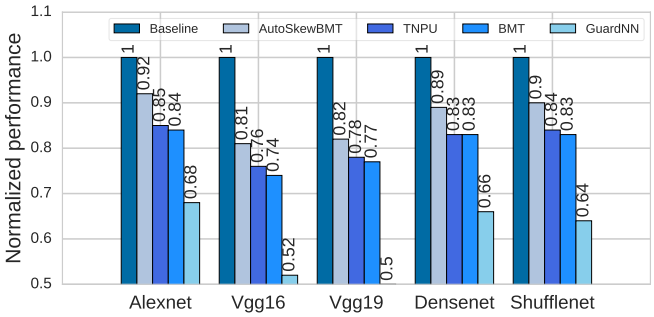


Fig. 7. Normalized performance comparison.

The distribution in performance is supported by Tab. IV. As the re-organization of the counters within the skewed BMT is heavily influenced by the number of writes (as tree updates due to counter write is the critical bottleneck, as discussed in the previous section), the resultant lower number of hash operations would directly coincide with the performance boost. As a consequence, we see the best performance improvement for AutoSkewBMT in Alexnet (Fig. 7) as AutoSkewBMT optimization results in 23% reduction of the total number of hash operations for that workload (Tab. IV).

TABLE IV
Number of hash operations for counter write

	Alexnet	Vgg16	Vgg19	Densenet	Shufflenet
BMT	65352	4005648	4503744	140928	207366
Skewed BMT	50376	3456734	4018253	121762	193397
Reduction	**23%**	**13.7%**	**10.8%**	**13.6%**	**6.7%**

Even though the total number of writes to different counters is only about 20% of the total counter operations for all workloads (Tab. V), the writes should have a high impact on the system performance even for balanced BMT as every data write i.e., counter write must initiate an update to the tree in an eager-update system (even for a write hit in the counter cache, as every write must be persisted to the tree). The update of the tree due to a counter write can take at least six

times higher latency than a verification of the said counter during a read (as mentioned before). It is because the hashing of all the nodes can be performed in parallel during the read operation after the nodes are read, while the hashing during the update operation for an eagerly-updated tree needs to be performed in sequence and cannot be overlapped [7], [20]. This difference can be even larger for BMT cache hits since the verification always stops at a cached node, while the update is always propagated to the root for eager update (as seen in Fig. 3). Since both the Vgg models use a significantly higher number of counters and read/write operations compared to other models, all the secure systems exhibit higher overheads for these models compared to other workloads. On the other hand, Densenet and Shufflenet have a higher overhead for all the systems compared to the Alexnet even though they have relatively lower number of total counter operations (Tab. V). However, both of them also have considerably higher number of counter write operations compared to Alexnet (Tab. V), which explains the additional overhead.

TABLE V
COUNTER ACCESS STATISTICS

	No. of Counters	Counter Read	Counter Write
Alexnet	700	45052	10892
Vgg16	6291	2153945	667608
Vgg19	7573	2867225	750624
Densenet	237	3612	23488
Shufflenet	320	3686	34561

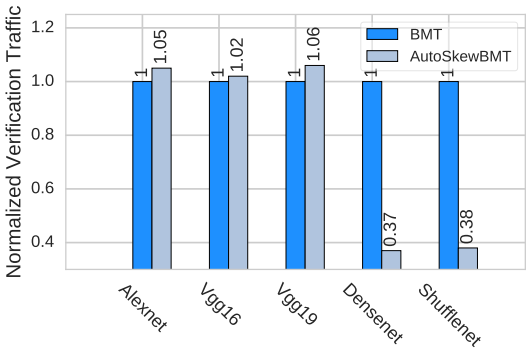


Fig. 8. Normalized worst-case BMT verification.

We again initiate the workloads, but this time without the BMT caches to determine the total number of worst-case BMT node verification operations due to counter read requests. In the absence of BMT cache, every single BMT node in a verification chain needs to be read from the memory and verified individually. This process will always reach the root in this case. As AutoSkewBMT puts significantly more weights on the counters with higher number of writes compared to the number of reads (Alg. 2), some counters that are more frequently accessed for writes would get prioritized for skewing over the counters with a high number of reads but no writes as these counters with read-heavy accesses might be pushed down to lower levels of the tree. This would nominally increase the number of node verification from some workloads (Fig. 8) as the counters at the bottom-most level would now have to traverse one extra tree level (as seen with CTR1, CTR2, CTR5 and CTR6 in Fig. 5). However, both Densenet and Shufflenet generate read accesses to only a few out of all the counters. This scenario allows these counters to have a large weight as they also have high number of writes and as such, they are moved up the tree level during the skewed BMT generation. As a result, we see a drastic reduction of the percentage of total node verification for Densenet and Shufflenet in this simulated worst-case scenario (Fig. 8). The latency breakdown in Fig. 9 shows the largest BMT latency reduction for Alexnet (as it reduces the hashing by 23% (Tab. IV)), but all the benchmarks benefit from AutoSkewBMT. The latency breakdown in Fig. 9 shows the largest BMT latency reduction for Alexnet (as it reduces the hashing by 23% (Tab. IV)) and results in best performance gain in Fig. 7), but all the benchmarks benefit from AutoSkewBMT. Even though both Densenet and Shufflenet have a lot of write operations (Tab. V), they see less latency reduction as the percentage of reduced hash operations for them is relatively lower (Tab. IV).

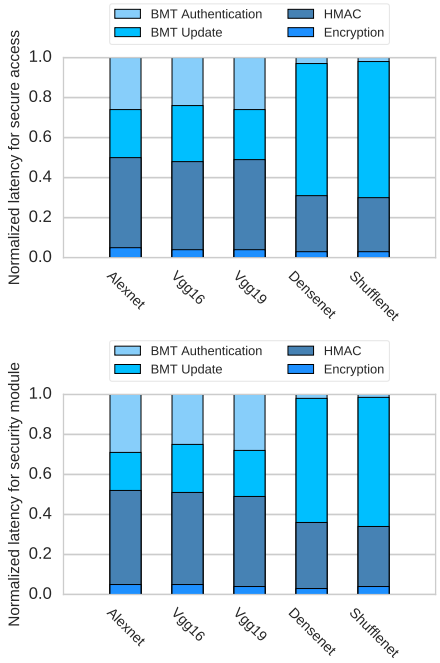


Fig. 9. Normalized latency for different security during memory access, for a secure-memory system with regular BMT (top) and the system with AutoSkewBMT-generated BMT configurations (bottom).

VII. DISCUSSION

AutoSkewBMT can generate an optimally skewed BMT configuration for a particular DNN workload and accelerator before the system is initialized. It is flexible, scalable and can be used with any DNN workload. However, currently there is no run-time tree regeneration support. Every time a new model is used, the toolchain needs to be compiled again with the new parameters to create a new version of the tree. Future research might look into making the run-time tree generation process feasible with low overhead.

VIII. CONCLUSION

We propose AutoSkewBMT, a tool chain to autonomously generate the configurations for an optimized metadata integrity authentication system for FPGA-based DNN accelerators. Our tool generates an optimal integrity tree from the design parameters for an accelerator and the feature map details of a given DNN model. We observe up to 7% and 32% respective performance improvement for our work over two state-of-the-art secure accelerator setups.

REFERENCES

[1] N. Shrivastava and S. R. Sarangi, "Securator: A fast and secure neural processing unit," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 1127–1139.

[2] A. C. Elster and T. A. Haugdahl, "Nvidia hopper gpu and grace cpu highlights," *Computing in Science & Engineering*, vol. 24, no. 2, pp. 95–100, 2022.

[3] P. Dhilleswararao, S. Boppu, M. S. Manikandan, and L. R. Cenkeramaddi, "Efficient hardware architectures for accelerating deep neural networks: Survey," *IEEE Access*, 2022.

[4] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Guardnn: secure accelerator architecture for privacy-preserving deep learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 349–354.

[5] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, "Triad-NVM: Persistency for integrity-protected and encrypted non-volatile memories," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 104–115.

[6] R. Elbaz, D. Champagne, C. Gebotys, R. B. Lee, N. Potlapally, and L. Torres, "Hardware mechanisms for memory authentication: A survey of existing techniques and engines," *Transactions on Computational Science IV*, pp. 1–22, 2009.

[7] Y. Zou, K. A. Zubair, M. Alwadi, R. M. Shadab, S. Gandham, A. Awad, and M. Lin, "ARES: Persistently secure non-volatile memory with processor-transparent and hardware-friendly integrity verification and metadata recovery," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 21, no. 1, pp. 1–32, 2022.

[8] A. Freij, H. Zhou, and Y. Solihin, "Bonsai Merkle forests: Efficiently achieving crash consistency in secure persistent memory," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1227–1240.

[9] R. M. Shadab, Y. Zou, S. Gandham, A. Awad, and M. Lin, "HMT: A hardware-centric hybrid bonsai merkle tree algorithm for high-performance authentication," *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 4, pp. 1–28, 2023.

[10] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[11] A. Awad, S. Suboh, M. Ye, K. A. Zubair, and M. Al-Wadi, "Persistently-secure processors: Challenges and opportunities for securing non-volatile memories," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 610–614.

[12] M. Henson and S. Taylor, "Memory encryption: A survey of existing techniques," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, pp. 1–26, 2014.

[13] R. M. Shadab, Y. Zou, S. Gandham, and M. Lin, "OMT: A run-time adaptive architectural framework for bonsai merkle tree-based secure authentication with embedded heterogeneous memory," in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2023, pp. 191–202.

[14] Y. Zou, A. Awad, and M. Lin, "Hermes: Hardware-efficient speculative dataflow architecture for bonsai Merkle tree-based memory authentication," in *2021 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2021, pp. 203–213.

[15] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *arXiv preprint arXiv:1803.05961*, 2018.

[16] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang, and J. Song, "Occlumency: Privacy-preserving remote deep-learning inference using sgx," in *The 25th Annual International Conference on Mobile Computing and Networking*, 2019, pp. 1–17.

[17] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, "Tnpu: Supporting trusted execution with tree-less integrity protection for neural processing unit," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2022, pp. 229–243.

[18] H. C. Tanuwidjaja, R. Choi, S. Baek, and K. Kim, "Privacy-preserving deep learning on machine learning as a service—a comprehensive survey," *IEEE Access*, vol. 8, pp. 167 425–167 447, 2020.

[19] W. Hua, M. Umar, Z. Zhang, and G. E. Suh, "Mgx: Near-zero overhead memory protection for data-intensive accelerators," in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, 2022, pp. 726–741.

[20] R. M. Shadab, Y. Zou, S. Gandham, A. Awad, and M. Lin, "A secure computing system with hardware-efficient lazy bonsai merkle tree for fpga-attached embedded memory," *IEEE Transactions on Dependable and Secure Computing*, 2023.

[21] Y. Zou and M. Lin, "Fast: A frequency-aware skewed Merkle tree for FPGA-secured embedded systems," in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2019, pp. 326–331.

.

Rank-based Multi-objective Approximate Logic Synthesis via Monte Carlo Tree Search

Yuyang Ye[1], Xiangfei Hu[2], Yuchen Liu[2], Peng Xu[1], Yu Gong[3], Tinghuan Chen[4], Hao Yan[2], Bei Yu[1], Longxing Shi[2]
[1]CUHK [2]Southeast University [3]Nanjing University of Aeronautics and Astronautics [4]CUHK-Shenzhen

***Abstract*—Approximate Logic Synthesis (ALS) is an automated technique designed for error-tolerant applications, optimizing delay, area, and power under specified error constraints. However, existing methods typically focus on either delay reduction or area minimization, often leading to local optima in multi-objective optimization. This paper proposes a rank-based multi-objective ALS framework using Monte Carlo Tree Search (MCTS). It develops non-dominated circuit ranking, to guide MCTS in exploring local approximate changes (LACs) across the entire circuit and generate approximate circuit sets with great optimization potential. Additionally, a Rank-Transformer model is introduced to predict path-domain ranks, enhancing the application of high-quality LACs within circuit paths. Experimental results show that our framework achieves faster and more efficient optimization in delay and area simultaneously compared to state-of-the-art methods.**

I. Introduction

As transistor technology advances into the nanoscale, optimizing circuit timing and area has become critical. With the growing demand for error-tolerant applications such as image processing and machine learning, approximate computing has emerged as an innovative circuit design paradigm [1]. By carefully managing introduced errors, it enables significant reductions in circuit delay and area with minimal impact on functionality.

To automatically generate approximate circuits that meet user-defined error constraints, Approximate Logic Synthesis (ALS) techniques have been developed. These methods optimize circuits by applying Local Approximate Changes (LACs) on circuits, which modify local circuit structures. Based on their optimization targets, ALS methods are categorized into two types: (1) **Delay-driven methods:** These methods iteratively apply LACs to critical paths, reducing the propagation delay of specific gates and improving overall circuit timing. For instance, HEDALS [1] and TCAD24 [2] use critical error graphs and reinforcement learning to accelerate this process with minimal error. DCGWO [3] reduces both critical path depth and area while enhancing gate drive strength, further decreasing delay. (2) **Area-driven methods:** These methods, including [4]–[9], treat all circuit gates in circuits as LAC candidates, prioritizing them that remove the most gates. Through multiple iterations, they identify LACs that maximize area reduction.

However, existing chips commonly pursue higher frequency and lower area usage. This requires designers to optimize both delay and area simultaneously. The LACs used in delay-driven methods are overly localized, as they are confined to critical paths and adjacent paths of them. Meanwhile, the LACs used in area-driven methods are too global, as they are scattered across the entire circuit. As a result, both methods fail to adequately balance the global and local search for LACs, leading to insufficient multi-objective optimization.

When dealing with a multi-objective ALS problem with an extremely large solution space, Evolutionary Algorithms [10] and Bayesian Optimization [11] struggle to find the true Pareto frontier within an acceptable cost. In this case, LAMOO [12] recommends using Monte Carlo Tree Search (MCTS) to assist in learning space partitioning. By sampling within the finely partitioned solution space, LAMOO can quickly find optimal solutions. The method relies on accurate performance modeling for effective partitioning and searching. However, it is challenging to obtain accurate models for large circuits, which is considered in our work. The inaccuracy easily causes conventional MCTS to sample in incorrect solution spaces, leading to local optima.

To address the aforementioned challenges, we propose a rank-based multi-objective Approximate Logic Synthesis (ALS) framework leveraging Monte Carlo Tree Search (MCTS). This framework integrates domain-specific knowledge in ALS to customize and implement MCTS, achieving an effective balance between global and local search for Logic Approximation Candidates (LACs). By incorporating circuit-domain and path-domain ranking mechanisms, the framework guides MCTS exploration, enabling efficient simultaneous optimization of delay and area without relying on precise modeling. Our main contributions are summarized as follows:

- We propose a multi-objective ALS framework based on tailored MCTS. It can optimize delay and area under various error constraints.
- We design a non-dominated circuit ranking method to split LAC solution space and to guide MCTS's global exploration. This method identifies LAC-induced approximate circuits with high optimization potential of delay and area for subsequent sampling.
- We develop a Rank-Transformer to predict path-domain LAC ranks, steering MCTS toward high-quality local LACs on critical paths, facilitating more effective delay and area optimization.
- On open-source circuits using the TSMC 28nm library, our framework achieves superior delay reduction and area savings under different error constraints, with some speedup compared to state-of-the-art methods.

II. Preliminaries

A. Error Metrics

Our framework supports two commonly used error metrics: normalized mean error distance (NMED) and error rate (ER). For a circuit with respectively I and O numbers of PIs and POs, assume that the probability of input vector I_i occurring is P_i ($1 \leq i \leq 2^I$). In this case, NMED and ER can be defined as follows:

Mean error distance is the mean difference between approximate circuit output value V_i^{app} and accurate circuit output value V_i^{acc}. NMED is the mean error distance normalized by the maximum output value, defined in Equation (1).

$$\text{NMED} = \sum_{i=1}^{2^I} \frac{P_i \times |V_i^{\text{app}} - V_i^{\text{acc}}|}{2^O - 1}. \tag{1}$$

ER is the total probability of incorrect outputs generated by the input patterns of an approximate circuit. It is calculated by Equation (2), where O_i^{app} and O_i^{acc} are output vectors of the approximate circuit and accurate circuit for input vector I_i.

$$\text{ER} = \sum_{i=1}^{2^I} \left[P_i \times (O_i^{\text{app}} \neq O_i^{\text{acc}})\right]. \tag{2}$$

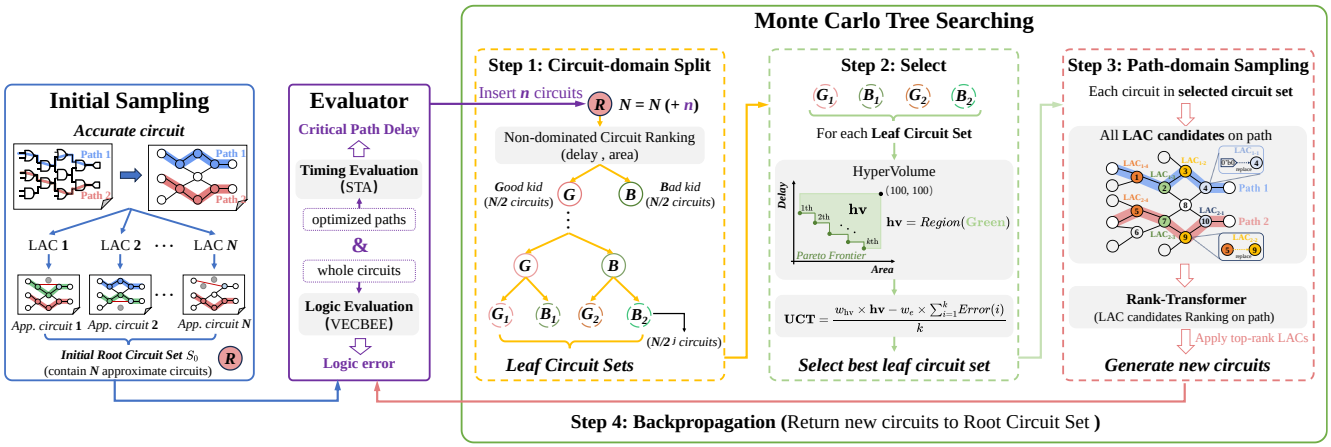


Fig. 1 The overall flow of Rank-based Multi-objective Approximate Logic Synthesis framework via Monte Carlo Tree Search.

B. Timing and Logic Error Evaluator

Timing Evaluation. We use the critical path delay, which determines the minimum specified clock period, to measure the timing performances of circuits. The critical path delay of the approximate circuit is obtained through Static Timing Analysis using Synopsys PrimeTime [13].

Logic Error Evaluation. Instead of the traditional full-input pattern traversal method, we use VECBEE [8] based on Monte Carlo simulation to evaluate the logic error of the approximate circuit. This method can accelerate the capture of signal changes caused by LACs, achieving fast error evaluation with nearly no deviation.

C. Problem Formulation

Each LAC used in our work is gate-based, defined as follows:

Definition 1 (Gate-based LAC)**.** *The LAC that substitutes a gate in the circuit with another gate or a constant logic value '0' or '1'. The replaced gate is called the target gate, and the gate (or value) replacing it is called the switch gate.*

Based on **Definition 1**, the rank-based ALS problem can be formulated as follows:

Problem 1 (Rank-based ALS)**.** *Given a post-synthesis accurate circuit with timing and logic information, select and apply proper LACs based on the rank of the explored gate-based LACs to deeply optimize its delay and area under the error constraint.*

III. Proposed Framework

The overall flow of our framework is shown in Fig. 1. The framework uses MCTS to deeply explore the optimization potential of delay and area. Each node in the search tree represents a circuit set, containing multiple approximate circuits. The MCTS receives initial circuits and iteratively performs four steps: *Split*, *Selection*, *Sampling* and *Backpropagation*. In each iteration, *Split* dynamically modifies the tree structure from the root circuit set. *Selection* identifies a suitable leaf circuit set, while *Sampling* uses gate-based LACs to perform further optimization on the circuits within the selected leaf circuit set. The new circuits are *backpropagated* to the root circuit set after timing and logic evaluation. This iterative process ensures efficient exploration and optimization across the approximate circuit space.

A. Initial Sampling

The accurate circuit is represented by a directed acyclic graph (DAG). In this DAG, nodes are gates and edges are connection relationships between gates. To generate circuits for MCTS optimization, we applied N different LAC samplings separately to the accurate circuit.

Within each LAC sampling, the target gate is selected from the critical paths. Then, the switch gate is filtered from gates with shorter arrival time in the path. To limit the introduced error, the gate having the highest **output similarity** with the target gate is preferred. Assume that $O_{I_i}^{\text{tar}}$ and $O_{I_i}^{\text{swi}}$ are respectively output signal of the target gate and the switch gate candidate under input vector I_i, the output similarity SIM can be calculated in Equation (3):

$$\text{SIM} = \sum_{i=1}^{N_I} \left[\frac{(O_{I_i}^{\text{tar}} == O_{I_i}^{\text{swi}})}{N_I} \right], \tag{3}$$

where N_I is the number of input vectors. After evaluation, the generated circuits without error violations are inserted into the root circuit set for MCTS.

B. Circuit-domain Rank-based Splitting

The split step is designed to achieve the partitioning of the current root circuit set $S_t : \{c_1^t, c_2^t, \ldots, c_n^t\}$ and reconstruct a performance-driven search tree. To cover the solution space of **Problem 1**, an exponential number of circuits are required. After circuit-domain splitting, the vast solution space (i.e., the root circuit set S_t) is partitioned and performance-driven promising circuit sets (i.e., leaf circuit sets) are generated. In leaf circuit sets, the timing and area performance of circuits are better and the optimization potential is greater. As capturing an accurate performance model and optimization gradient is challenging, we propose a non-dominated circuit ranking to sort the circuits in S_t. This ranking process consists of two parts: Pareto level ranking, and crowding distance ranking within one specific Pareto level.

Pareto Level Ranking. In Pareto level ranking, we first establish the Pareto dominance between circuits in the circuit set. If c_i^t is not worse than c_j^t in both delay and area, and strictly better in at least one, then c_i^t dominates c_j^t. Circuits not dominated by any others are assigned to the 0-th Pareto level (i.e., their Pareto-level PL $= 0$). Subsequently, circuits dominated only by circuits in previous Pareto levels are assigned to the next level. This process repeats until all circuits are assigned. Taking Fig. 2 as an example, approximate circuit A is not dominated by any other circuit, thus its Pareto-level $\text{PL}_A = 0$. Since B and C are dominated only by circuits in the 0-th Pareto level, they are assigned to the 1-th Pareto level. Additionally, D and E are dominated only by circuits in the 0-th and 1-th Pareto level, thus they are assigned to the 2-th Pareto level. Circuits with lower

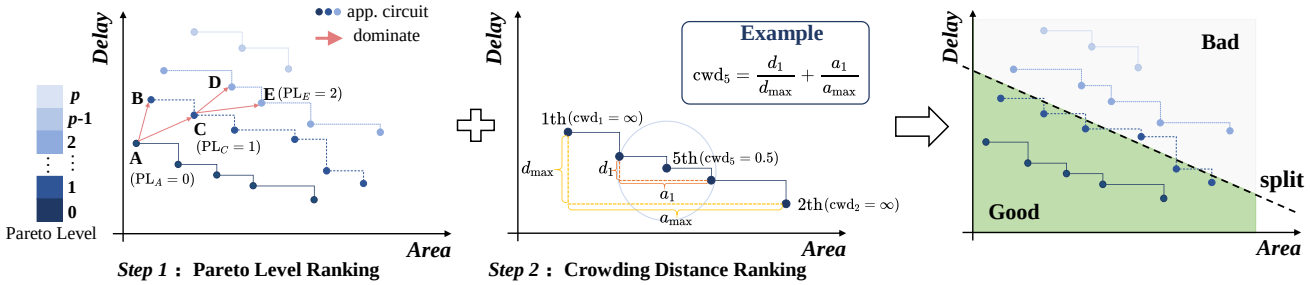


Fig. 2 The illustration of non-dominated circuit ranking. Step 1: Coarse ranking based on Pareto Level. Step 2: Precise ranking based on crowding distance within each Pareto Level.

Pareto-level PL indicate better timing and area performance.

Crowding Distance Ranking. Within each Pareto level, we perform further ranking based on the crowding distance. For circuits with higher crowding distances, conducting further sampling on them can lead to more efficient exploration. It is because their gradients are less likely to overlap in the solution space. Therefore, we sort the circuits within the same Pareto level in descending order of crowding distance. As shown in the Fig. 2, in the i-th Pareto level, the largest delay difference and area difference between circuits are $d_{\max}$ and $a_{\max}$, respectively. For circuit c_j^i in this Pareto level, the adjacent circuits are c_{j-1}^i and c_{j+1}^i. Assume that the delay difference and area difference between c_{j-1}^i and c_{j+1}^i are d_j and a_j, respectively. In this case, the crowding distance of c_j^i can be calculated as follow:

$$\mathrm{cwd}(c_j^t) = \frac{d_j}{d_{\max}} + \frac{a_j}{a_{\max}}. \tag{4}$$

Since the circuits at both ends of the Pareto level have no adjacent circuits, their crowding distances are set to infinity ($+\infty$). In the case of identical crowding distances, we consider circuits with a smaller delay to be more worthy of further sampling. The overall rank is referred to as the circuit-domain ranking and used to replace gradient prediction for space partitioning.

Splitting based on Circuit-domain Ranking. The MCTS partitions S_t and reconstructs the search tree based on the circuit-domain ranking. At the beginning of the iteration, the search tree will be reset to the root circuit set S_t. The non-dominated circuit ranking is performed on S_t to get the circuit rank. Circuits in the top 50% rank with better performance and greater optimization potential are labeled as positive while the others are negative. Subsequently, two leaf circuit sets are expanded from S_t, including a good kid set that receives the positive circuits and a bad kid set that receives the negative circuits. The generated good kid set also undergoes the aforementioned split and expansion. This process continues until the number of circuits inside all newly generated leaf circuit sets is less than a user-specified value. In the final rounds of the process, the newly generated bad kid set also experiences split and expansion to ensure the diversity of the partitioned spaces.

C. Selection

The Selection step is an extension of the circuit-domain rank. It incorporates accumulated logic errors from the iterative optimization process to guide a more precise search. Specifically, it filters the best circuit set based on the upper confidence bound for trees (UCT) from the leaf circuit sets of the search tree generated after splitting. Since ALS requires sufficient optimization of the objectives with minimal error, UCT is computed based on the HyperVolume $\mathbf{hv}$ and the average error of k Pareto front circuits within the circuit set.

HyperVolume $\mathbf{hv}$ can provide an effective evaluation of the convergence, uniformity, and spread of samples when the true Pareto frontier cannot be determined [12]. Therefore, it is highly suitable for our optimization problem. Higher $\mathbf{hv}$ characterizes better optimization quality of the Pareto front circuits within the circuit set. Assuming the x-coordinate of the objective space represents the ratio of i-th approximate circuit area to accurate circuit area $Ratio_a^{(i)}$, and the y-coordinate represents the corresponding delay ratio $Ratio_d^{(i)}$. In this case, the reference point is (1, 1). $\mathbf{hv}$ is the area enclosed by k Pareto front circuits and the reference point, as defined in Equation (5). Note that $Ratio_a^{(0)} = 1$. For one leaf circuit set, it is computed as:

$$\mathbf{hv} = \sum_{i=1}^{k} \left[(1 - Ratio_d^{(i)}) \times (Ratio_a^{(i-1)} - Ratio_a^{(i)}) \right]. \tag{5}$$

The average error is introduced to ensure the approximated circuits are under the error constraint. Based on HyperVolume $\mathbf{hv}$ and the average error, UCT of one leaf circuit set is computed as:

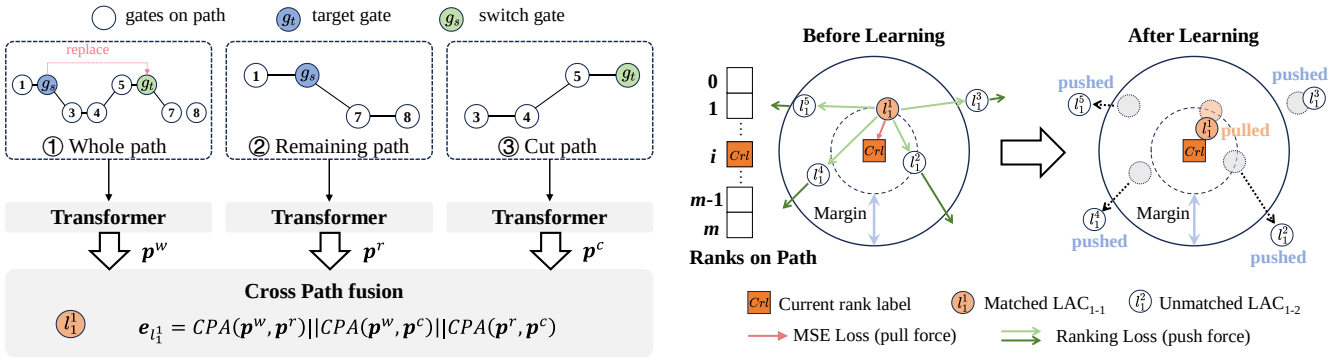
$$\mathbf{UCT} = \frac{w_{hv} \times \mathbf{hv} - w_e \times \sum_{i=1}^{k} Error(i)}{k}, \tag{6}$$

where w_{hv} and w_e are weights of HyperVolume and average error, respectively. The leaf circuit set with the highest UCT is selected for further sampling optimization.

D. Path-domain Rank-based Sampling

The sampling is performed on all circuits inside the selected leaf circuit set. In our MCTS framework, we maintain and apply a set of timing-area-reducing LAC $\mathcal{L}_M : \{l_M^1, l_M^2, \ldots, l_M^n\}$ for each circuit to generate new approximate circuits. To ensure that applying $\mathcal{L}_M$ can bring both stable delay and area reduction, each LAC in $\mathcal{L}_M$ is extracted from critical paths of the circuit. In a given critical path with $K+1$ gates, we consider all gates except those directly connected to PI as target gate candidates. For each candidate, we select a gate in its preceding path with the highest similarity SIM as the switch gate, thereby forming a gate-based LAC candidate l_C. This approach yields a critical path LAC set $\mathcal{L}_C : \{l_c^1, l_c^2, \ldots, l_c^K\}$. To select high-quality l_c from $\mathcal{L}_C$ to insert into $\mathcal{L}_M$. Thus, the LAC candidates in $\mathcal{L}_C$ need to be ranked based on their induced delay and area improvements. As ranking through simulation incurs high time costs, we design and implement a Rank-Transformer to enable fast rank predictions on paths. Its detailed information includes feature engineering, embedding & fusion and rank prediction.

Feature Engineering. As shown in Fig. 3a, for each l_c, we create three paths based on its target gate g_t and switch gate g_s, including: (1) the whole path, which retains the complete information of the path before approximation; (2) the cut path, which contains information of removed gates; (3) the remaining path, which includes information of



(a) Path embedding and fusion.

(b) Training of rank prediction.

Fig. 3 The illustration of the Rank-Transformer. In (b), matched LAC refers to LAC whose rank within a path matches the true rank label, while unmatched LAC refers to LAC with mismatched rank. The pull is driven by MSE Loss, and the push is induced by Ranking Loss.

the path after approximation. These three paths can effectively capture the impact of approximation on the entire critical path, enabling the rank-transformer to comprehensively learn path-domain information.

Additionally, three parameters influencing the optimization brought by l_c are also provided and regarded as the feature of l_c, including delay reward Reward_D, area reward Reward_A and similarity SIM. SIM is already defined in Section III-A and is computed based on Equation (3). Reward_D refers to the difference between the maximum delays of g_t and the g_s. Reward_A refers to the total area of gates inside the difference set between the maximum fanout free cones (MFFCs) of g_t and g_s. These two rewards are defined as follows:

$$\text{Reward}: \begin{cases} \text{Reward}_D = \text{Delay}(g_t) - \text{Delay}(g_s) \\ \text{Reward}_A = \text{Area}(MFFC_{g_t} - MFFC_{g_s}) \end{cases}, \quad (7)$$

.

Embedding & Fusion. The embedding result of each LAC candidate $\boldsymbol{e}_{l_c}$ is generated based on fusion embedding results of the whole path, the cut path and the remaining path. Transformer helps us to embed each path independently and generates the whole path embedding result $\boldsymbol{p}_{l_c}^w$, the cut path embedding result $\boldsymbol{p}_{l_c}^c$ and the remaining path embedding result $\boldsymbol{p}_{l_c}^r$. As shown in Fig. 3a, a cross-path attention mechanism (CPA) proposed in [14] is then employed to achieve path fusion. In our work, the self-attention in Transformer helps to capture internal-path relationships and the cross-path attention mechanism facilitates information exchange among different paths. Finally, the final embedding result of LAC candidate $\boldsymbol{e}_{l_c}$ can be computed by contacting the three path embedding results as:

$$\boldsymbol{e}_{l_c} = \text{CPA}(\boldsymbol{p}_{l_c}^w, \boldsymbol{p}_{l_c}^c) \| \text{CPA}(\boldsymbol{p}_{l_c}^w, \boldsymbol{p}_{l_c}^r) \| \text{CPA}(\boldsymbol{p}_{l_c}^c, \boldsymbol{p}_{l_c}^r) \quad (8)$$

Rank Prediction. Based on the final embedding result of LAC candidate $\boldsymbol{e}_{l_c}$, we can predict the rank of it using the Softplus function as:

$$r_c = \text{Softplus}(\boldsymbol{e}_{l_c}). \quad (9)$$

Inspired by Ranknet [15], we design a loss function $\mathcal{L}$ in this work to improve the accuracy of predicted LAC ranks on the path. $\mathcal{L}$ is defined as follows:

$$\mathcal{L} = \mathcal{L}_{MSE} + \mathcal{L}_{rank}, \quad (10)$$

where $\mathcal{L}_{MSE}$ and $\mathcal{L}_{rank}$ are MSE loss and ranking loss respectively. For a given LAC candidate l_c, assume the true rank label of it is $\hat{r}_c$. In this case, the MSE loss is the mean squared difference between the predicted ranks and the true rank labels:

$$\mathcal{L}_{MSE} = \frac{1}{K} \sum_{i=1}^{K} (r_c - \hat{r}_c)^2. \quad (11)$$

As shown in Fig. 3b, for a given true rank label, the MSE loss $\mathcal{L}_{MSE}$ pulls the predicted rank of matched LAC (i.e., LAC whose true rank match the current true rank label) closer to it. Since relying solely on the pull force of the MSE Loss for ranking can lead to the homogenization of predicted ranks, the ranking loss $\mathcal{L}_{rank}$ is introduced for correction. $\mathcal{L}_{rank}$ is a customized pair-wise loss built upon the comparison between the predicted ranks of l_c and l_j:

$$\mathcal{L}_{rank} = \frac{2}{K^2 - 1} \sum_{i=1}^{K} \max(0, R_c^i), \quad (12)$$

$$R_c^i = \text{Margin} - \text{sign}[(r_c - r_i) \cdot (\hat{r}_c - \hat{r}_i)], \quad (13)$$

where Margin is a hyper-parameter. Fig. 3b illustrates that the ranking loss $\mathcal{L}_{rank}$ generates a push force between the predicted ranks of matched LAC and unmatched LACs (i.e., LACs whose true ranks do not match the current true label rank), pushing the latter at least by a Margin difference away from the former. Therefore, the predicted ranks of LAC candidates in $\mathcal{L}_C$ can maintain certain differences, effectively preventing ranking homogenization.

By applying the above rank prediction mechanism to the path, the rank of LAC candidates in $\mathcal{L}_C$ (i.e., the **path-domain rank**) are effectively obtained. LAC candidates with top ranks are selected into $\mathcal{L}_M$ and applied. In the later iterations of MCTS, the number of critical paths in the circuit may become extremely large. When several rounds of search fail to effectively reduce the delay, a set of area-reducing LAC $\mathcal{L}_A : \{l_A^1, l_A^2, \ldots, l_A^n\}$ for each circuit is maintained to replace $\mathcal{L}_M$. Then, LACs in $\mathcal{L}_A$ are chosen by a greedy algorithm.

E. Backpropagation

As the length of LACs sequences applied to the accurate circuit increases with iterations, the performances of the newly generated circuits, including their delay reductions, area reductions and introduced logic errors, can effectively represent the impact brought by current optimization process. Therefore, these newly generated circuits are backpropagated to the root circuit set S_t after evaluation, expanding S_t to S_{t+1}. Note that circuits with error violations are discarded.

TABLE I Statistics of the benchmarks used in our experiment. The units of delay and area are respectively ps and μm^2.

Random / Control				Arithmetic			
Circuit	#Gate	Delay	Area	Circuit	#Gate	Delay	Area
c880	322	185.34	177.67	c6288	1641	847.79	687.08
c1908	366	235.14	223.34	adder	1639	1394.7	495.78
c2670	922	218.40	288.71	barshift	2933	262.52	1806.6
c3540	667	293.09	459.42	max	2940	2799.8	954.03
c5315	2595	122.25	1129.6	mult	26429	4117.5	31635.6
c7552	1576	282.13	939.33	sine	11560	3234.4	7173.9
cavlc	573	186.35	450.31	sqrt	13542	67929.3	6262.1
priority	2336	1126.8	1423.3	square	14696	8211.1	7752.8

IV. Experimental Results

A. Experiment Setup

Our framework is implemented in C++ and the Rank-Transformer is implemented in Python using the PyTorch library. Our work is tested on the Linux machine with 32 cores and 4 NVIDIA Tesla V100 GPUs. The LAC candidates used for the Rank-Transformer training are derived from TSMC 28nm standard arithmetic circuits, including different bit-width adders, multipliers, and divisors.

Our framework is applied to circuits selected from ISCAS'85 [16] and EPFL [17] to demonstrate its effectiveness and generalization capability. These circuits are pre-synthesized to gate-netlists with TSMC 28nm technology using Synopsys Design Compiler [18]. Their statistics are listed in TABLE I. Among them, the random/control circuits are optimized under ER constraints, while the arithmetic circuits are optimized under NMED constraints. For each generated approximate circuit, we use Synopsys PrimeTime [13] to extract its critical paths and report the delay. Meanwhile, the circuit area is obtained using ABC [19]. In terms of logic simulation, we randomly generate 100,000 input vectors for VECBEE [8] to ensure both accuracy and efficiency of the logic error estimation.

Important parameters of our framework are listed as follows. The upper bound of MCTS iterations is 26. For $Split$ step, the minimum number of circuits required for the leaf circuit set to undergo further splitting is set to 20. For UCT in $Selection$ step, the weight of HyperVolume w_{hv} = 0.7, while the weight of average error w_e = 0.3. These two weights are determined based on the optimization results of multiple tests. The Margin used in the pair-wise loss $\mathcal{L}_{rank}$ of the Rank-Transformer is 20.

Since our framework enables fast multi-objective optimization, three indicators are used to evaluate its performance, including **delay ratio** (the critical path delay of the best approximate circuit over the accurate one), **area ratio** (the area of the best approximate circuit over the accurate one) and **runtime**. Considering the randomness of logic simulation, our framework performs five optimization runs for each test circuit. The average values of the above three indicators are used as the final evaluation criteria.

B. Multi-objective Optimization Results

We compare our framework with the sate-of-the-art delay-driven and area-driven works under ER and NMED constraints. For delay-driven works, we choose three methods, including: HEDALS [1] using critical error graph, TCAD24 [2] utilizing reinforcement learning, and DCGWO [3] optimizing both critical path depth and area. For area-driven works, VECBEE-SASIMI [8], which combines a greedy algorithm with efficient error estimation, is selected.

Results under ER constraint. We compare the multi-objective optimization performance of our framework with other works under a 3% ER constraint. According to the results presented in TABLE II, our framework obtains greater delay reduction in the majority of cases and reduces more area across all cases. On average, it achieves a 29.84% delay reduction while saving 23.24% in area.

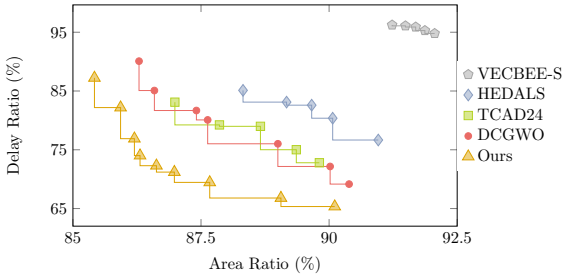


Fig. 4 Multi-objective optimization results across all works on 128-bit multiplier under 1.96% NMED constraint.

Furthermore, we present the optimization results of all works under different ER constraints (1% - 5%), including the average delay ratio trends shown in Fig. 5a and the average area ratio trends illustrated in Fig. 5c. The results indicate that our framework consistently achieves higher levels of optimizations in both delay and area while meeting various error rate requirements.

Results under NMED constraint. We compare the multi-objective optimization performance of our framework with other works under a 1.96% NMED constraint. The results presented in TABLE II indicate that our framework obtains greater delay and area reductions in the majority of cases. On average, our framework achieves a 25.51% delay reduction while saving 15.93% in area, both outperforming other works. The variations in delay and area reductions achieved by all works under different NMED constraints (0.48% - 2.44%) are shown in Fig. 5b and Fig. 5d, respectively. The results also demonstrate that our framework consistently achieves deeper optimizations in both delay and area while meeting diverse error distance requirements.

Additionally, to determine whether the approximate circuits generated by our framework are fully dominant in the objective function space, we evaluate the optimization results of all works on 128-bit multiplier under the 1.96% NMED constraint. We retain all approximate circuits in the 0-th Pareto level of the root circuit set to construct the Pareto frontier of our framework. Based on the results in Fig. 4, the Pareto front circuits generated by our framework still remain on the Pareto frontier among the circuits produced by all works, providing strong evidence of the superiority of our work in multi-objective optimization.

C. Runtime Comparison

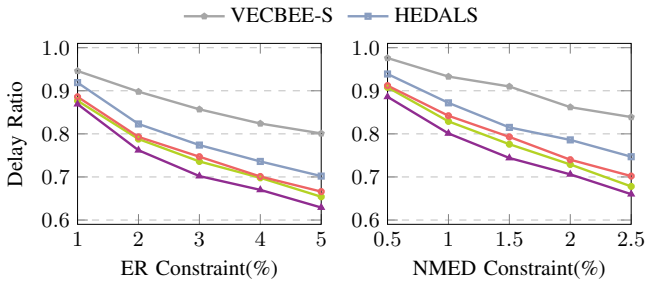
The overall runtimes for the circuit optimization flow under NMED constraint across different methods are listed in TABLE IV. Note that the overall runtime accounts for both the time consumed in approximate optimization and all evaluation processes. The comparison results demonstrate that our framework achieves an average speedup of 1.19× to 4.16× compared to TCAD24 [2]. As shown in Fig. 6, for both DCGWO [3] and our framework, which utilize Synopsys PrimeTime [13] for accurate timing analysis, the primary time consumption in each iteration is attributed to timing analysis and logic simulation. Benefiting from the rank-based partition of the approximate circuit set, the application of multiple LACs to promising circuits in each iteration, and the inherent parallelism of MCTS, our framework achieves rapid search and convergence speeds. Therefore, our framework can achieve fast and efficient optimization.

TABLE II Comparison of multi-objective optimization performance between our framework and other works under the 3% ER constraints.

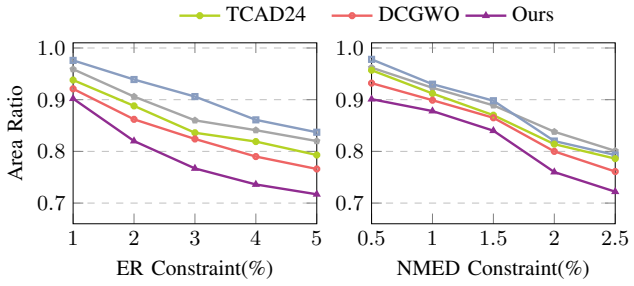
Circuit	VECBEE-S [8]		HEDALS [1]		TCAD24 [2]		DCGWO [3]		Ours	
	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio
c880	92.16%	86.75%	93.02%	89.22%	83.21%	84.98%	85.21%	82.76%	**77.09%**	**67.54%**
c1908	84.21%	63.37%	**46.12%**	59.36%	48.15%	62.34%	43.97%	57.02%	49.89%	**45.72%**
c2670	79.14%	69.78%	79.64%	94.17%	75.39%	61.28%	76.92%	60.33%	**74.11%**	**56.96%**
c3540	97.93%	94.72%	89.97%	92.46%	84.32%	90.55%	90.61%	87.14%	**75.06%**	**85.18%**
c5315	93.57%	96.68%	94.24%	97.81%	88.55%	90.29%	89.72%	91.12%	**87.06%**	**89.87%**
c7552	91.79%	95.66%	78.53%	99.72%	77.58%	95.34%	79.88%	94.29%	**71.43%**	**91.06%**
cavlc	93.20%	83.78%	96.83%	92.85%	94.28%	85.38%	**92.07%**	89.62%	94.35%	**81.18%**
priority	53.17%	97.16%	47.96%	98.77%	37.28%	98.54%	39.12%	97.22%	**32.25%**	**96.57%**
Average	85.65%	85.99%	77.41%	90.55%	73.60%	83.59%	74.69%	82.44%	**70.16%**	**76.76%**

TABLE III Comparison of multi-objective optimization performance between our framework and other works under 1.96% $NMED$ constraints.

Circuit	VECBEE-S [8]		HEDALS [1]		TCAD24 [2]		DCGWO [3]		Ours	
	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio	delay ratio	area ratio
c6288	97.33%	90.81%	73.28%	89.90%	74.38%	91.25%	76.92%	89.01%	**71.95%**	**87.72%**
adder	82.62%	96.42%	78.14%	94.02%	66.88%	92.96%	74.23%	93.69%	**59.23%**	**92.50%**
barshift	90.01%	83.87%	87.46%	89.98%	83.12%	80.55%	82.78%	83.26%	**82.16%**	**75.62%**
max	92.55%	86.44%	81.96%	93.24%	75.99%	85.64%	76.81%	91.80%	**74.86%**	**82.33%**
mult	95.89%	91.79%	82.59%	89.66%	78.99%	88.66%	80.08%	87.63%	**71.20%**	**86.98%**
sine	94.10%	90.28%	91.11%	93.48%	86.14%	88.56%	**82.10%**	89.78%	87.60%	**86.25%**
sqrt	83.23%	91.07%	75.03%	92.10%	75.66%	90.54%	79.16%	**86.29%**	**71.21%**	89.32%
square	92.53%	80.81%	82.13%	76.27%	79.36%	77.54%	82.58%	72.23%	**77.72%**	**71.86%**
Average	91.03%	88.94%	81.46%	89.83%	77.56%	86.95%	79.33%	86.46%	**74.49%**	**84.07%**



(a) Delay Ratios under ER (b) Delay Ratios under NMED

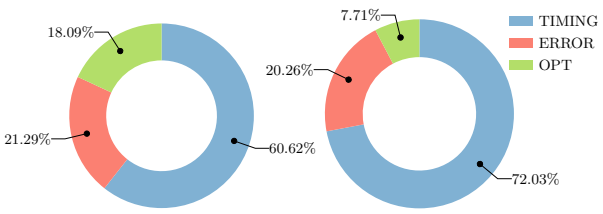


(c) Area Ratios under ER (d) Area Ratios under NMED

Fig. 5 Aveage delay ratios and average area ratios achieved by our framework and others under different ER and NMED constraints.

TABLE IV Overall Runtime (min.) Comparison.

Circuit	VECBEE-S	HEDALS	TCAD24	DCGWO	Ours
c6288	58.60	34.18	22.95	28.05	**20.32**
adder	22.63	18.17	17.68	15.92	**10.09**
barshift	37.89	30.11	22.69	21.26	**17.70**
max	34.68	39.44	23.31	27.13	**19.22**
mult	293.01	187.82	57.83	64.12	**40.38**
sine	71.88	51.63	51.24	42.35	**39.76**
sqrt	551.95	268.8	105.87	132.67	**98.95**
square	57.22	37.13	23.74	**20.95**	24.78
Average	140.98	83.41	40.66	44.06	**33.90**



(a) DCGWO [3] (b) Our framework

Fig. 6 Runtime breakdown in one iteration.

V. Conclusion

This paper presents a rank-based multi-objective Approximate Logic Synthesis (ALS) framework implemented using Monte Carlo Tree Search (MCTS). The core approach focuses on: (1) utilizing non-dominated circuit ranking to guide MCTS in globally identifying approximate circuits with high optimization potential for further exploration, and (2) leveraging the Rank-Transformer to predict path-domain rankings of local approximate changes (LACs) to select high-quality LACs within critical paths. These selected LACs enable effective optimization of both delay and area. Experimental results demonstrate that the proposed framework efficiently optimizes delay and area under various error constraints with low runtime costs.

Acknowledgment

The project is supported in part by the National Key R&D Program of China (No. 2023YFB4402900), the National Natural Science Foundation of China under Grant 62304197, Research Grants Council of Hong Kong SAR (No. RFS2425-4S02, No. CUHK14211824 and No. CUHK14210723), and the MIND project (MINDXZ202404).

REFERENCES

[1] C. Meng, Z. Zhou, Y. Yao, S. Huang, Y. Chen, and W. Qian, "Hedals: Highly efficient delay-driven approximate logic synthesis," *IEEE TCAD*, vol. 42, no. 11, pp. 3491–3504, 2023.

[2] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Timing-driven technology mapping approximation based on reinforcement learning," *IEEE TCAD*, 2024.

[3] X. Hu, Y. Ye, T. Chen, H. Yan, and B. Yu, "Timing-driven approximate logic synthesis based on double-chase grey wolf optimizer," 2024. [Online]. Available: https://arxiv.org/abs/2411.10990

[4] S. Venkataramani, A. Sabne, V. Kozhikkottu, K. Roy, and A. Raghunathan, "Salsa: Systematic logic synthesis of approximate circuits," in *Proc. DAC*, 2012, pp. 796–801.

[5] Y. Wu and W. Qian, "An efficient method for multi-level approximate logic synthesis under error rate constraint," in *Proc. DAC*, 2016, pp. 1–6.

[6] S. Hashemi, H. Tann, and S. Reda, "Blasys: Approximate logic synthesis using boolean matrix factorization," in *Proc. DAC*, 2018, pp. 1–6.

[7] C. Meng, W. Qian, and A. Mishchenko, "Alsrac: Approximate logic synthesis by resubstitution with approximate care set," in *Proc. DAC*. IEEE, 2020, pp. 1–6.

[8] S. Su, C. Meng, F. Yang, X. Shen, L. Ni, W. Wu, Z. Wu, J. Zhao, and W. Qian, "VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE TCAD*, vol. 41, no. 11, pp. 5085–5099, 2022.

[9] J. Ma, S. Hashemi, and S. Reda, "Approximate logic synthesis using boolean matrix factorization," *IEEE TCAD*, vol. 41, no. 1, pp. 15–28, 2021.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[11] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9851–9864, 2020.

[12] Y. Zhao, L. Wang, K. Yang, T. Zhang, T. Guo, and Y. Tian, "Multi-objective optimization by learning space partitions," *arXiv preprint arXiv:2110.03173*, 2021.

[13] Synopsys, "Primetime user guide," https://www.synopsys.com/cgi-bin/imp/pdfdla/pdfr1.cgi?file=primetime-wp.pdf, 2023.

[14] S. Yan, X. Xiong, A. Arnab, Z. Lu, M. Zhang, C. Sun, and C. Schmid, "Multiview transformers for video recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 3333–3343.

[15] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 89–96.

[16] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE MDTC*, vol. 16, no. 3, pp. 72–80, 1999.

[17] EPFL, "The epfl combinational benchmark suite," https://www.epfl.ch/labs/lsi/page-102566-en-html/benchmarks/, 2019.

[18] Synopsys, "Design compiler user guide," https://www.synopsys.com/zh-cn/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html, 2023.

[19] A. Mishchenko *et al.*, "ABC: A system for sequential synthesis and verification," 2022.

Truly Pre-Routing Timing Prediction via Considering Power Delivery Network

Yuyang Ye[1], Mingwei He[2], Lizheng Ren[2], Jianwang Zhai[3,†], Tinghuan Chen[4], Jun Yang[2], Longxing Shi[2]
[1]CUHK [2]Southeast University [3]Beijing University of Posts and Telecommunications [4]CUHK, Shenzhen

***Abstract*—Fast and accurate pre-routing timing prediction is essential in the chip design flow. However, existing machine learning (ML)-assisted pre-routing timing methods often overlook the impact of power delivery networks (PDNs), which contribute to IR drop and routing congestion. This limitation can make these methods less practical for real-world circuit design flows. To address this, we propose two specialized encoders—an IR drop-aware encoder and a routing congestion-aware encoder—that effectively capture PDN effects through multimodal fusion of netlist, layout, and PDN data. To mitigate the challenges of imbalanced multimodal fusion, we further develop a Pareto optimization approach to ensure balanced utilization of all modalities, enhancing timing prediction accuracy. Comprehensive experiments on large-scale open-source designs using TSMC's 16nm technology node validate the superiority of our model over state-of-the-art pre-routing timing prediction methods.**

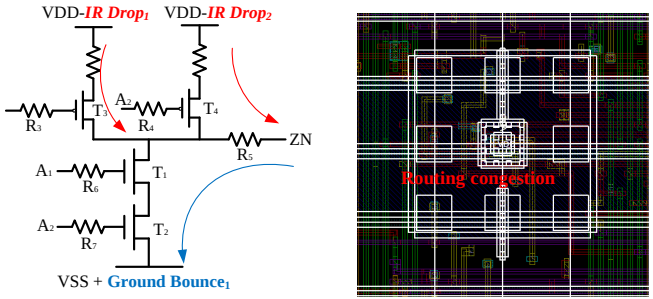
I. Introduction

Static Timing Analysis (STA) engines are widely used throughout physical design stages to guide timing optimization. However, during the placement stage, the absence of wire parasitic information leads to inaccurate timing estimates, resulting in suboptimal placement quality and requiring multiple design iterations. These iterations are often time-consuming due to the lengthy routing stages. Therefore, an accurate and efficient pre-routing timing prediction model based on the placement solution is crucial for enabling timing-driven placement and avoiding redundant routing [1].

Recent research has leveraged machine learning (ML) techniques to improve pre-routing timing prediction by incorporating the impact of routing on timing results [2]–[6]. Works like [5] and [6] focus on learning bimodal information from netlists and layouts using traditional multimodal fusion methods. Compared to unimodal approaches [2]–[4], incorporating multimodal information significantly improves the accuracy of pre-routing timing predictions.

However, these studies do not account for the impact of power delivery networks (PDNs) on timing performance. PDNs are essential for supplying stable voltage to chip components and maintaining signal integrity. As technology scales down, increased parasitic resistance in PDN interconnects can lead to IR drop issues, causing timing degradation. As shown in Fig. 1a, IR drop, including drop on the VDD rail and bounce on the VSS rail of a NAND cell, slows the switching speed and increases propagation delay. In traditional IC design flows, PDN design is done before placement. As shown in Fig. 1b, the white areas represent the PDN in the layout. A well-designed PDN requires dedicated metal layers that consume routing resources, increasing routing congestion. This reduces available space for signal routing, potentially leading to longer wire lengths and introducing additional parasitic resistance and capacitance, further impacting timing. Therefore, PDN effects, including IR drop and routing congestion, must be considered for achieving accurate pre-routing timing prediction models.

PDNs, netlists, and layouts represent distinct modalities in circuit design. However, the information in PDNs is more domain-specific and limited compared to netlists and layouts, creating an imbalance in modality representation. Traditional multimodal learning techniques, such as those in [5] and [6], assume a balanced relationship among modalities, which is not the case here. This imbalance presents challenges in training effective timing models, a problem we term imbalanced multimodal learning. A critical question arises: **How can we effectively integrate all circuit modalities (netlist, layout, and PDN) for pre-routing timing prediction?**



(a) Cell circuits. (b) Routed layout.

Fig. 1 PDN causes different IR drops and routing congestion results. IR drops influence cell delays and routing congestion results influence routing solutions.

In this work, we propose a novel pre-routing timing prediction framework that effectively balances the multimodal fusion of netlist, layout, and PDN data. To fully capture circuit information from all modalities, we develop two specialized encoders: IR drop-aware encoder integrates layout-based and PDN-based knowledge on one netlist-based model; Routing congestion-aware encoder integrates netlist-based and PDN-based knowledge on one layout-based model. To ensure balanced learning across all circuit modalities during timing model training, we introduce a Pareto multimodal fusion algorithm. This algorithm uses IR drop and routing congestion predictions as innocent assistance to improve the accuracy of pre-routing timing predictions. We highlight our contributions:

- We introduce the first multimodal fusion framework combining netlist, layout, and PDN information for pre-routing timing prediction.
- We develop an IR drop-aware encoder and a routing congestion-aware encoder, enabling accurate modeling of PDN impacts.
- We design the Pareto multimodal fusion algorithm to address the imbalanced multimodal learning problem during timing model training, thereby improving timing prediction accuracy with the assistance of IR drop and routing congestion predictions.
- We conduct experiments and ablation studies on large-scale open-source designs in the advanced TSMC 16nm technology.

II. Preliminaries

A. Power Network Design vs. PDN Graph

As shown in Fig. 2, we model a complex power delivery network as a PDN graph $\mathbb{G}_d = (\mathcal{V}_d, \mathcal{E}_d, \mathcal{P}_d)$, where each node $v_d \in \mathcal{V}_d$

†Corresponding author.

represents a PDN node, and each edge $e_d \in \mathcal{E}_d$ represents a power wire resistance or a power via resistance between different PDN nodes. The PDN path p_d in path set $\mathcal{P}_d$ is defined as:

Definition 1 (PDN path)**.** *The path from the power source to the target cell consists of all nodes, power vias, and power wires visited.*

In our work, PDN paths are selected based on the total resistance of power vias and power wires. For each cell, the top 5 paths with the smallest total resistance from each power source are identified using depth-first search.

B. Problem Formulation

Problem 1 (Pre-routing Timing Prediction Considering PDN)**.** *Given the pre-routing layout, netlist, and power delivery network of a circuit, our goal is to make a truly accurate and efficient estimation of the sign-off global timing metrics, i.e., endpoint arrival time.*

C. Multimodal Pre-routing Timing Prediction

As illustrated in Fig. 3a, we briefly introduce the overall flow of past mutimodal pre-routing timing prediction works [5], [6]. In [5], the authors introduce a novel endpoint embedding framework that combines graph and convolutional neural networks to extract netlist and layout information, respectively. Meanwhile, [6] presents LayNet, a multimodal neural network designed for pre-routing congestion prediction, which integrates both layout and netlist data through a novel heterogeneous message-passing approach.

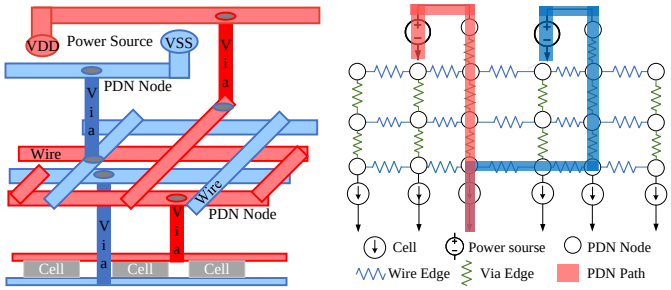
However, these approaches do not address the effects of PDNs, including IR drop and routing congestion, which are critical factors in accurate timing prediction. In contrast, this work aims to fill this gap by introducing a multimodal fusion framework that considers PDN effects alongside netlist and layout information. Our approach models timing performance by integrating data from three key aspects:

- **Netlist:** Netlist information is essential for delay computation, as it captures the connections between cells and wires, which form the basis for timing analysis.
- **Layout:** Placement and layout information play a crucial role in determining the routing solution, impacting timing performance.
- **PDN:** The PDN information contributes to model PDN effects, including IR drop and routing congestion results.

III. Proposed Framework

A. Overall Flow

As illustrated in Fig. 3b, we first briefly introduce the overall flow of our pre-routing timing prediction framework. The proposed framework can be divided into two parts: Part 1 learns the information in netlist, layout and PDN modality data through two specialized encoders: the IR drop-aware encoder (Section III-C) and the routing congestion-aware encoder (Section III-D); and Part 2 achieves balanced multimodal fusion based on Pareto method to improve timing prediction accuracy (Section III-E). In Part 1, the IR drop-aware encoder achieves a netlist-based model, while the routing-congestion encoder achieves a layout-based model. They learn netlist, layout, and PDN modality data. As shown in Fig. 3b, they can output embedding results, including $\boldsymbol{e}^{ir}$ and $\boldsymbol{e}^{co}$, which are used in IR drop and routing congestion prediction, respectively. By combining them ($\boldsymbol{e}^{ir}||\boldsymbol{e}^{co}$), it is used in pre-routing timing prediction. In Part 2, we develop a Pareto-based multimodal fusion method that ensures balanced integration of all modalities, improving the accuracy of the timing predictions. During model training, the method computes the optimal gradients for the IR drop-aware encoder (i.e., $\mathbf{p}^{ir}(\theta^{ir})$) and the routing congestion-aware encoder (i.e., $\mathbf{p}^{co}(\theta^{co})$). As shown in Fig. 3b, these gradients are used to update the encoder parameters θ^{ir} and θ^{co}, allowing the framework to integrate the IR drop and routing congestion predictions to enhance the pre-routing timing prediction. By fully leveraging and balancing the information from netlist, layout, and PDN modalities, our framework achieves highly accurate pre-routing timing predictions.



(a) Power delivery network (PDN). (b) PDN grpah.

Fig. 2 PDN is represented with PDN graph in our work. PDN node is translated to the node in graph, and PDN wire and via are translated to edge in graph. PDN path is the most important supplying path from the power source to the target cell.

As shown in Fig. 3a, traditional timing-driven multimodal learning methods often rely on a single multimodal joint timing loss, which optimizes the netlist and layout feature encoders. However, the PDN modality, being more domain-specific, can cause imbalanced learning when fused with netlist and layout data, leading to suboptimal results. To resolve this, we introduce IR drop loss and routing congestion loss in the respective encoders to balance modality contributions and improve prediction accuracy. In our work, the final loss functions are:

$$\mathcal{L} = \mathcal{L}_t + \mathcal{L}_{ir} + \mathcal{L}_{co}, \tag{1}$$

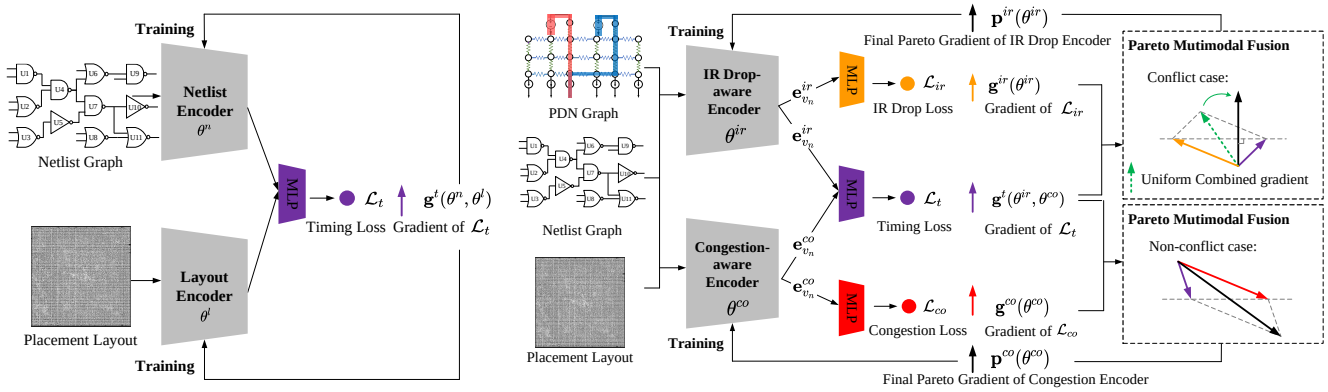
where $\mathcal{L}_t$ is the multimodal joint timing loss, $\mathcal{L}_{ir}$ is the IR drop loss and $\mathcal{L}_{co}$ is the routing congestion loss. They are cross-entropy loss functions. The details of achieving and updating IR drop-aware and routing congestion-aware encoders are discussed as follows.

B. Data Representation

PDN Modality Data. The PDN graph $\mathbb{G}_d$ is represented with node feature matrix $\boldsymbol{X}^D$: $\{\boldsymbol{x}^D_{v_d}, v_d \in \mathcal{V}_d\}$. The details of features in feature vector $\boldsymbol{x}^D_{v_d}$ includes: (1) Located metal layer: the number of the located metal layer. (2) Connected power wire resistance: total resistance of power wires connected with the target power node. (3) Connected power via resistance: total resistance of power vias connected with the target power node.

Netlist Modality Data. We transfer the circuit netlist to a graph $\mathbb{G}_n = (\mathcal{V}_n, \mathcal{E}_n)$ consisting of a node set ($\mathcal{V}_n$), an edge set ($\mathcal{E}_n$). Nodes are cells and Edges are wires in circuits. The details of features in node feature vector $\boldsymbol{x}^N_{v_n}$ for cell node include: (1) Cell type: e.g., NAND, NOR, embedded as a one-hot vector; (2) Pin capacitance: extracted from the timing library; (3) Cell drive strength: extracted from the cell type name of circuits. (4) Cell supply voltage: extracted from the design requirement. The details of features in edge feature vector $\boldsymbol{x}^N_{e_n}$ for wire include: (1) Placement wire delay: extracted from timing report of circuit netlists after placement. The wire delay is calculated based on Elmore delay and the Manhattan distance between the positions of a wire's drive pin and its sink pin.

Layout Modality Data. We divide the overall placement layout into 512×512 bins. For each bin b_l on layout, the detailed considered layout features $\boldsymbol{X}^L$: $\{\boldsymbol{x}^L_{b_l}, b_l \in 512 \times 512 \text{ bins}\}$ should be closely correlated with the final routing congestion and solution, which



(a) Past bi-modal pre-routing timing prediction work.

(b) Our multi-modal pre-routing timing prediction work.

Fig. 3 Comparsion between our multimodal flow and conventional bimodal flow.

include: (1) The macro cell regions; (2) Rectangular uniform wire density (RUDY); (3) Gate density.

Labels. In our work, we use three distinct types of labels: (1) IR Drop Labels: The static IR drop values for each cell instance are extracted from ANSYS RedHawk [7], an industry-standard IR drop analysis tool. (2) Routing Congestion Labels: The golden congestion results of layouts are generated by IC Compiler II [8]. (3) Timing Labels: These labels represent the output pin arrival times for each cell after routing in the circuit. The golden IR drop-aware timing results are produced by the static timing analysis engine, PrimeTime [9]. The IR drop is incorporated into the timing analysis using the command read_sdf Modified_IR.sdf, where Modified_IR.sdf is the file generated by RedHawk after performing IR drop analysis.

C. IR Drop-aware Encoder

As shown in Fig. 4a, our IR drop-aware encoder learns timing-driven information for each cell v_n in the netlist through three distinct embedding strategies: (1) Netlist-only embedding: This embedding captures basic timing information from the circuit netlist by aggregating features of cells and wires. (2) Netlist-PDN embedding: This embedding gathers PDN information from the PDN paths associated with each cell in the netlist. (3) Netlist-layout embedding: This embedding integrates layout location data for each cell in the netlist through a novel message-passing paradigm applied to the layout edges. By combining these three embeddings, our model effectively incorporates a variety of timing-related information and output the IR drop-aware embedding result $\boldsymbol{e}^{ir}_{v_n}$, enhancing the accuracy of IR drop prediction and pre-routing timing prediction.

Netlist-only Embedding. Given the input netlist cell features $\{\boldsymbol{x}^N_{v_n}, v_n \in \mathcal{V}_n\}$ and edge features $\{\boldsymbol{x}^N_{e_n}, e_n \in \mathcal{E}_n\}$, graph learning can help achieve netlist feature embedding and output $\{\boldsymbol{e}^N_{v_n}, v_n \in \mathcal{V}_n\}$. For cell v_n, the $\boldsymbol{e}^N_{v_n}$ is generated via:

$$\boldsymbol{e}^N_{v_n} = \text{GNN}(\{\boldsymbol{x}^N_{v_n}, v_n \in \mathcal{V}_n). \tag{2}$$

Here, GNN proposed in the past timing prediction work [10] is used in our work to learn cell features and edge features jointly.

Netlist-PDN Embedding. For each cell v_n in the circuit netlist, we use a depth-first search (DFS) method to select PDN paths (shown in Section II-A) and generate the PDN path set $\mathcal{P}^d_{v_n}$. Using PDN node features $\{\boldsymbol{x}^D_{v_d}, v_d \in \mathcal{V}_d\}$, we leverage Transformers to embed PDN features path by path, learning information from the PDN paths. For a given PDN path p^d, the $\boldsymbol{t}_{p^d}$ is generated via:

$$\boldsymbol{t}_{p^d} = \text{Transformer}(\{\boldsymbol{x}^D_{v_d}, v_d \in \mathcal{N}_{p^d}\}), \tag{3}$$

where $\mathcal{N}_{p^d}$ is the node set on PDN path p^d. In real chip design, the cell's IR drop is caused jointly by different PDN paths. Thus, after encoding all paths in the path set $\{p^d \in \mathcal{P}^d_{v_n}\}$, we need to combine these path embeddings $\{\boldsymbol{t}_{p^d}, p^d \in \mathcal{P}^d_{v_n}\}$ together. Different PDN paths have different contributions to the IR drop of the target cell. Thus, we adopt a PDN path attention mechanism α_{p^d} to learn the different weights:

$$\alpha_{p^d} = \frac{\exp(\text{LeakyReLU}(\boldsymbol{W}_1 \cdot \boldsymbol{t}_{p^d}))}{\sum_{p^d_k \in \mathcal{P}^d_{v_n}} \exp(\text{LeakyReLU}(\boldsymbol{W}_1 \cdot \boldsymbol{t}_{p^d_k}))}, \tag{4}$$

where $\boldsymbol{W}_1$ is the trainable weight vector. Based on attention mechanisms, the final PDN path embedding result and netlist-PDN embedding result of cell v_n should be generated via:

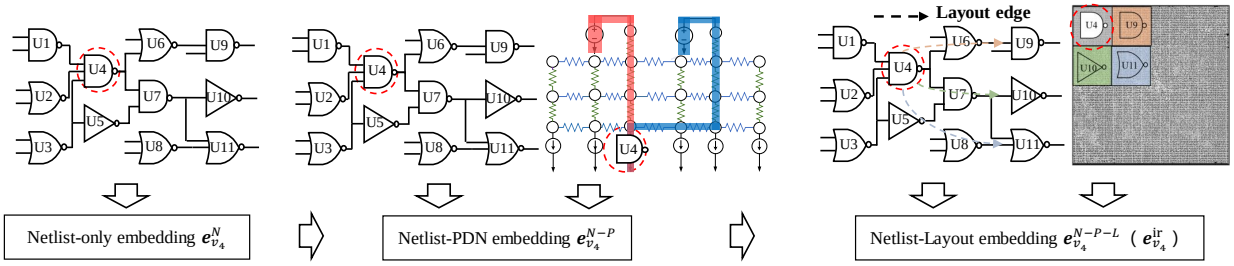
$$\boldsymbol{e}^P_{v_n} = \sigma(\sum_{p^d \in \mathcal{P}^d_{v_n}} \alpha_{p^d} \cdot \boldsymbol{t}_{p^d}), \quad \boldsymbol{e}^{N-P}_{v_n} = \boldsymbol{e}^P_{v_n} \| \boldsymbol{e}^N_{v_n}. \tag{5}$$

Netlist-Layout Embedding. Unlike traditional pre-routing timing prediction methods, our approach constructs layout edges for cells in the netlist based on their placement information. As shown in the right part of Fig. 4a, we provide an example of a layout edge. For cell U4, its layout neighbors U9, U10, and U11 are placed nearby, though not adjacent in the netlist. These layout neighborhoods are connected by layout edges, which are crucial for timing and IR drop prediction but challenging for conventional graph learning methods. We refer to these cells as layout neighborhoods of cell U4, connected by layout edges. To better capture this information, we introduce a layout neighbor attention mechanism ($\alpha^{N-L}_{v_n}$) that embeds cell information through layout edges. The layout attention coefficients $\alpha^{N-L}_{v_n}$ are computed as follows:

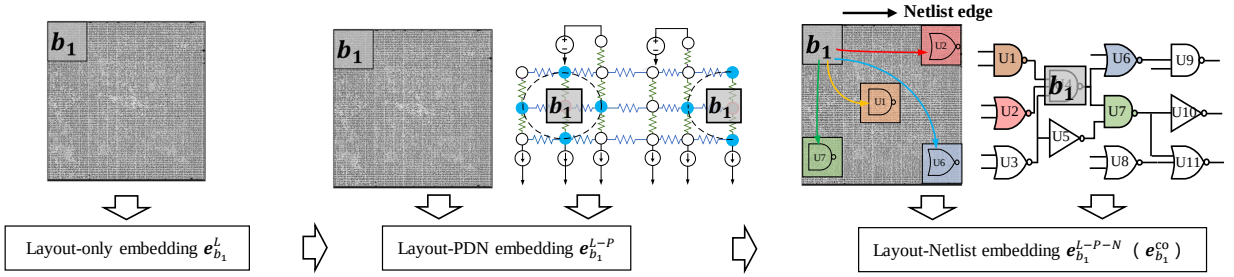
$$\alpha^{N-L}_{v_n} = \frac{\exp(\text{LeakyReLU}(\boldsymbol{W}_2 \cdot \boldsymbol{e}^{N-P}_{v_n}))}{\sum_{v_j \in \mathcal{N}^L_{v_n}} \exp(\text{LeakyReLU}(\boldsymbol{W}_2 \cdot \boldsymbol{e}^{N-P}_{v_j}))}, \tag{6}$$

where v_n is the target cell and the v_j is its layout neighbor belongs to layout neighborhood set N_{v_n}, which is connected with target cell v_n through layout edge. $\boldsymbol{W}_2$ is the trainable weight vector. The negative input slope of LeakyReLU nonlinear function is set as 0.2. Based on attention mechanisms, the final IR drop-aware embedding of cell v_n should be generated via:

$$\boldsymbol{e}^{ir}_{v_n} = \boldsymbol{e}^{N-P-L}_{v_n} = \sigma(\sum_{v \in \mathcal{N}_{v_n}} \alpha^{N-L}_v \cdot \boldsymbol{e}^{N-P}_v). \tag{7}$$



(a) The illustration of IR drop-aware encoder.



(b) The illustration of routing congestion-aware encoder.

Fig. 4 The information in the netlist graph, PDN graph, and layout image is learned through two encoders: IR drop-aware encoder and Routing congestion-aware encoder.

D. Routing Congestion-aware Encoder

As shown in Fig. 4b, in our routing congestion-aware encoder, for each bin b_l in the layout, we learn timing-driven information through three types of embeddings: (1) Layout-only embedding: Captures basic physical information from the circuit layout, important for modeling routing demands and solutions. (2) Layout-PDN embedding: Incorporates PDN information to predict routing congestion for each bin. (3) Layout-netlist embedding: Integrates netlist adjacent information for each bin in the layout.

Layout-only embedding. On layouts, We start by encoding layout features grid by grid independently. For the trade-off between efficiency and effectiveness, the ResNet [11] and ASPP [12] are used to extract and compress input layout features, respectively. Thus, the layout-only embedding result for each bin $\boldsymbol{e}^L_{b_l}$ can be computed as:

$$\boldsymbol{e}^L_{b_l} = \text{ResNet-ASPP}(\boldsymbol{x}^L_{b_l}). \tag{8}$$

Layout-PDN embedding. For each bin b_l, we generate a set of bin-wise PDN nodes based on their locations. As shown in the middle part of Fig. 4b, for bin b_l in the layout, two PDN nodes are located in the region, forming the bin-wise PDN node set. Using the PDN node features $\{\boldsymbol{x}^D_{v_d}, v_d \in \mathcal{V}_d\}$, we apply a graph learning model (GraphSAGE [13]) to achieve local PDN feature embeddings and learn information from the PDN graph. For each PDN node v_d, the embedding $\boldsymbol{e}_{v_d}$ is generated as follows:

$$\boldsymbol{e}_{v_d} = \text{GraphSage}(\{\boldsymbol{x}^D_{v_d}, v_d \in \mathcal{N}^D_{b_l}\}), \tag{9}$$

where $\mathcal{N}^D_{b_l}$ is the PDN node set for the layout bin b_l. We then compute the layout-PDN embedding for bin b_l by averaging pooling the local feature embeddings of all PDN nodes in $\mathcal{N}^D_{b_l}$, and combining it with the layout-only embedding $\boldsymbol{e}^L_{b_l}$:

$$\boldsymbol{e}^{L-P}_{b_l} = \text{AVE}\ (e_{v_d}, v_d \in \mathcal{N}^D_{b_l}) \parallel \boldsymbol{e}^L_{b_l}. \tag{10}$$

Layout-Netlist embedding. Building on the netlist and layout, Lay-Net [6] integrates netlist-based message passing into a layout-based model to improve congestion prediction performance. As shown in the right part of Fig. 4b, netlist connection information can be modeled on the layout using netlist edges. For example, in bin b_1, where cell U4 is located, information from other bins containing neighboring cells of U4 can be captured through Lay-Net through netlist edge. Thus, the routing congestion-aware embedding $\boldsymbol{e}^{co}_{b_l}$ is computed as:

$$\boldsymbol{e}^{co}_{b_l} = \boldsymbol{e}^{L-P-N}_{b_l} = \text{Lay-Net}(\boldsymbol{e}^{L-P}_{b_l}). \tag{11}$$

Cell-wise Masking. Inspired by [5], gate-wise masking $\boldsymbol{M}_{v_n}$ can help us to get the routing congestion-aware embedding result for each gate $\{\boldsymbol{e}^{co}_{v_n}, v_n \in \mathcal{V}_N\}$ as:

$$\boldsymbol{e}^{co}_{v_n} = \boldsymbol{M}_{v_n} \boldsymbol{e}^{co}_{b_l}. \tag{12}$$

IR drop prediction and routing congestion prediction are achieved based on the IR drop-aware embedding result $\boldsymbol{e}^{ir}_{v_n}$ and routing congestion-aware embedding result $\boldsymbol{e}^{co}_{v_n}$. Importantly, timing prediction is achieved through combining $\boldsymbol{e}^{ir}_{v_n}$ and $\boldsymbol{e}^{co}_{v_n}$.

E. Pareto Multimodal Fusion

As shown in Equation (1), timing loss, IR drop loss, and routing congestion loss are closely related in multimodal cases. However, their gradients may conflict during model training, as illustrated in Fig. 3b. This raises the challenge of effectively integrating these gradients. To address this, we introduce Pareto integration into our multimodal fusion framework, which resolves conflicts among the timing, IR drop, and routing congestion gradients. The Pareto optimization problem during gradient integration of training is formulated as follows:

$$\begin{aligned} &\min_{\alpha^t, \alpha^{ir} \in \mathcal{R}} \left\| \alpha^t \mathbf{g}^t(\theta^{ir}) + \alpha^{ir} \mathbf{g}^{ir}(\theta^{ir}) \right\|^2 \\ &\text{s.t.} \quad \alpha^t, \alpha^{ir} \geq 0, \alpha^t + \alpha^{ir} = 1, \end{aligned} \tag{13}$$

$$\begin{aligned} &\min_{\beta^t, \beta^{co} \in \mathcal{R}} \left\| \beta^t \mathbf{g}^t(\theta^{co}) + \beta^{co} \mathbf{g}^{co}(\theta^{co}) \right\|^2 \\ &\text{s.t.} \quad \beta^t, \beta^{co} \geq 0, \beta^t + \beta^{co} = 1, \end{aligned} \tag{14}$$

where $\|*\|$ denotes the L_2-norm. For the IR drop-aware encoder parameters θ^{ir}, $\mathbf{g}^t(\theta^{ir})$ and $\mathbf{g}^{ir}(\theta^{ir})$ denote the gradients of the multimodal joint timing loss $\mathcal{L}_t$ and the IR drop loss $\mathcal{L}_{ir}$, respectively. Similarly, for the routing congestion-aware encoder parameters θ^{co}, $\mathbf{g}^t(\theta^{co})$ and $\mathbf{g}^{co}(\theta^{co})$ represent the gradients of $\mathcal{L}_t$ and the routing congestion loss $\mathcal{L}_{co}$, respectively. The weights α^t, α^{ir}, β^t, and β^{co} are the final gradient weights to be determined.

This optimization problem involves finding the minimum norm in the convex hull of the gradient vectors ($\mathbf{g}^t$, $\mathbf{g}^{ir}$, and $\mathbf{g}^{co}$). The final Pareto solution provides a common descent direction that benefits all learning objectives. To illustrate solving the Pareto optimization problem in Equation (13), its analytical solution can be derived as:

$$\begin{cases} \alpha^t = 1, \alpha^{ir} = 0 & \cos\beta \geq \frac{\|\mathbf{g}^t(\theta^{ir})\|}{\|\mathbf{g}^{ir}(\theta^{ir})\|} \\ \alpha^t = \frac{\left(\mathbf{g}^{ir}(\theta^{ir}) - \mathbf{g}^t(\theta^{ir})\right)^\top \mathbf{g}^{ir}(\theta^{ir})}{\left\|\mathbf{g}^t(\theta^{ir}) - \mathbf{g}^{ir}(\theta^{ir})\right\|^2}, \alpha^{ir} = 1 - \alpha^t & \text{otherwise,} \end{cases} \tag{15}$$

where β is the angle between $\mathbf{g}^t(\theta^{ir})$ and $\mathbf{g}^{ir}(\theta^{ir})$. Then, we have the final gradient after integration:

$$\mathbf{p}^{ir}(\theta^{ir}) = 2\alpha^t \mathbf{g}^t(\theta^{ir}) + 2\alpha^{ir}\mathbf{g}^{ir}(\theta^{ir}). \tag{16}$$

Here we use 2α as the gradient weight to keep the same gradient weight summation with a uniform baseline (i.e., $2\alpha_t^{ir} + 2\alpha^{ir} = 2$). When using stochastic gradient descent based on the integrated gradient, the parameter of the IR drop-aware encoder is updated as:

$$\theta^{ir} := \theta^{ir} - \varepsilon \mathbf{p}^{ir}(\theta^{ir}) + \varepsilon\zeta, \tag{17}$$

where ζ is the stochastic gradient descent noise term and it satisfies:

$$\zeta \sim \mathcal{N}\left(0, \frac{\left(2\alpha^t\right)^2 C^t + \left(2\alpha^{ir}\right)^2 C^{ir}}{|S|}\right), \tag{18}$$

where $\frac{C^t}{|S|}$ and $\frac{C^{ir}}{|S|}$ represent batch sampling covariance. Studies [14], [15] show that the strength of noise in stochastic gradient descent influences the minima found, with larger noise often leading to flatter minima that generalize better. For both the IR drop-aware and routing congestion-aware encoders, the gradients of the IR drop loss and routing congestion loss typically have larger magnitudes and higher batch sampling covariance than the timing loss (i.e., $\mathbf{g}^{ir}(\theta^{ir}) \gg \mathbf{g}^t(\theta^{ir})$ and $\mathbf{g}^{co}(\theta^{co}) \gg \mathbf{g}^t(\theta^{co})$). Analytical Pareto methods assign greater weight to the timing loss gradient, which can weaken model generalization in multimodal learning. To address this, our method considers both the direction and magnitude during gradient integration. It ensures the final gradient aligns with all learning objectives while enhancing the strength of noise to improve generalization. We handle conflict and non-conflict cases separately:

Non-conflict case. We first consider the case where $\cos\beta > 0$. In this scenario, the cosine similarity between $\mathbf{g}^t(\theta^{ir})$ and $\mathbf{g}^{ir}(\theta^{ir})$ is positive. Since the gradient vectors' convex combination is common to all learning objectives, we assign $2\alpha_t^{ir} = 2\alpha^{ir} = 1$ instead of using the analytic Pareto solution for gradient integration. This choice helps to enhance the stochastic gradient descent noise term. With this setting, the final gradient can be formulated as:

$$\mathbf{p}^{ir}(\theta^{ir}) = \mathbf{g}^t(\theta^{ir}) + \mathbf{g}^{ir}(\theta^{ir}). \tag{19}$$

Conflict case. In the case where $\cos\beta < 0$, it is crucial to determine a common direction for all losses while enhancing the strength of the stochastic gradient descent noise term during gradient integration. First, we compute the analytic solution of Equation (15), which provides a non-conflicting direction. To further enhance the noise term strength, we increase the magnitude of the final gradient as:

$$\mathbf{p}^{ir}(\theta^{ir}) = \frac{2\alpha^t \mathbf{g}^t(\theta^{ir}) + 2\alpha^{ir}\mathbf{g}^{ir}(\theta^{ir})}{\|2\alpha^t \mathbf{g}^t(\theta^{ir}) + 2\alpha^{ir}\mathbf{g}^{ir}(\theta^{ir})\|} \cdot \left\|\mathbf{g}^t(\theta^{ir}) + \mathbf{g}^{ir}(\theta^{ir})\right\|. \tag{20}$$

In this case, the noise term satisfies:

$$\xi \sim \mathcal{N}\left(0, \lambda^2 \cdot \frac{\left(2\alpha^t\right)^2 C^t + \left(2\alpha^{ir}\right)^2 C^{ir}}{|S|}\right), \tag{21}$$

where $\lambda = \left\|\mathbf{g}^t(\theta^{ir}) + \mathbf{g}^{ir}(\theta^{ir})\right\| / \left\|2\alpha^t \mathbf{g}^t(\theta^{ir}) + 2\alpha^{ir}\mathbf{g}^{ir}(\theta^{ir})\right\|$. Thus, our noise term ξ has a larger strength than the noise term ζ of the analytical Pareto solution shown in Equation (18). Accordingly, model generalization is improved.

TABLE I Benchmark statistics.

Circuit	#gates	#wires	#EPs	#PNs	#PVs+#PWs	Δt
DMX	11616	11671	4671	6394	11706	12.2%
GFX	11004	11156	49403	99900	251314	13.4%
AC97	4954	5046	2262	3418	5435	8.5%
VGA	32727	32863	17749	37844	48510	16.5%
NOVA	105756	107641	30409	43912	94740	15.6%
Tot.Train	166057	168377	104494	191468	471705	66.2%
TOP	3943	4121	1694	2698	5163	9.2%
ECG	39992	40941	7691	11804	29095	14.2%
ETH	25327	25450	11681	9578	26314	11.9%
USB	6666	7000	1799	2298	5700	14.4%
TATE	153192	154720	31480	61666	114319	18.2%
Tot.Test	229120	232232	54345	88044	180591	67.9%

IV. Experimental Results

Our framework is implemented in Python using the PyTorch library, and in C++. The multimodal timing model is trained on a Linux machine with 32 cores and 4 NVIDIA Tesla V100 GPUs, completing training in approximately 4.5 hours using parallel training on 4 GPUs. The process uses 128GB of memory, with embedding dimensions for both IR drop-aware and routing congestion-aware encoders set to 128. The model is trained for 200 epochs with a learning rate of 0.001 and a batch size of 1024.

We train and evaluate our timing model using various open-source designs [16], and the framework can be applied to unseen designs without retraining. The benchmark circuits include designs with varying netlist and PDN scales. These open-source designs are synthesized using Synopsys Design Compiler [17] under the TSMC 16nm technology node and go through power planning, placement, and routing using Synopsys IC Compiler II [8]. The VDD options include $1.0V$, $0.9V$ and $0.8V$. Detailed information is shown in TABLE I, where #EPs represents the number of endpoints, #PNs represents the number of power nodes, #PVs+#PWs represents the total number of power vias and wires. Δt represents the percentage of timing degradation induced by IR drop.

A. Timing Prediction Accuracy

We compare our framework with previous state-of-the-art (SOTA) works [2]–[6] for pre-routing timing prediction on cell output pin arrive time. Unlike these methods, which struggle with PDN effects such as IR drop and routing congestion, our framework effectively addresses these challenges. In "Direct Fusion", multimodality data is learned directly without Pareto fusion and the timing model is trained based on timing loss. In "Analytical Pareto", multimodality data is learned with Pareto fusion using the analytical solutions to integrated gradients of timing, IR drop and routing congestion loss. The accuracy of our framework and other SOTA models is shown in TABLE II, evaluated using the R^2 score (higher is better) and maximum absolute error ($MAXE$, lower is better).

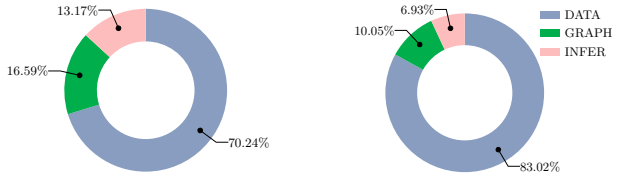
Our results demonstrate that our model accurately predicts true post-routing timing results while accounting for PDN effects. For training designs, the average R^2 and $MAXE$ scores are 0.96/5.30ps.

TABLE II Timing prediction accuracy. The unit of "$MAXE$" is ps.

Cir.	DAC19 [2]		DAC22-He [3]		DAC22-guo [4]		DAC23 [5]		Lay-Net [6]		Direct Fusion		Analytical Pareto		Ours	
	R^2	$MAXE$	R^2	$MAXE$	R^2	$MAXE$	R^2	$MAXE$	R^2	$MAXE$	R^2	$MAXE$	R^2	$MAXE$	R^2	$MAXE$
DMX	0.67	26.33	0.71	22.89	0.80	28.36	0.86	22.45	0.87	26.21	0.85	27.33	0.88	12.38	**0.96**	**5.26**
GFX	0.71	31.20	0.76	29.34	0.79	25.24	0.82	21.38	0.84	17.39	0.86	13.54	0.90	11.28	**0.93**	**4.52**
AC97	0.75	49.29	0.81	32.68	0.83	30.99	0.87	28.34	0.88	21.22	0.86	25.66	0.92	13.51	**0.94**	**3.06**
VGA	0.77	75.13	0.83	42.97	0.84	39.68	0.85	34.27	0.84	27.54	0.84	31.06	0.91	11.99	**0.97**	**6.59**
NOVA	0.71	93.14	0.81	71.54	0.82	38.17	0.85	56.28	0.87	37.28	0.88	32.59	0.93	18.46	**0.98**	**7.09**
Ave.	0.72	55.02	0.78	39.88	0.82	39.38	0.85	32.54	0.86	25.93	0.86	26.04	0.91	13.52	**0.96**	**5.30**
TOP	0.54	36.19	0.63	24.93	0.69	23.97	0.83	18.59	0.82	19.27	0.84	17.86	0.86	15.87	**0.96**	**3.08**
ECG	0.49	53.25	0.57	46.27	0.61	38.65	0.77	31.24	0.80	33.65	0.82	28.39	0.87	16.52	**0.93**	**4.53**
ETH	0.51	63.72	0.59	52.19	0.64	45.27	0.79	27.29	0.78	31.20	0.72	33.96	0.84	9.86	**0.91**	**3.67**
USB	0.38	46.86	0.56	50.28	0.63	41.38	0.76	32.05	0.80	24.36	0.73	33.81	0.86	12.56	**0.92**	**3.09**
TATE	0.33	113.64	0.62	99.37	0.65	95.68	0.80	78.69	0.81	69.59	0.75	81.22	0.83	22.53	**0.95**	**9.32**
Ave.	0.45	62.73	0.59	54.61	0.64	48.99	0.79	37.57	0.80	35.61	0.77	39.05	0.85	15.45	**0.93**	**4.74**

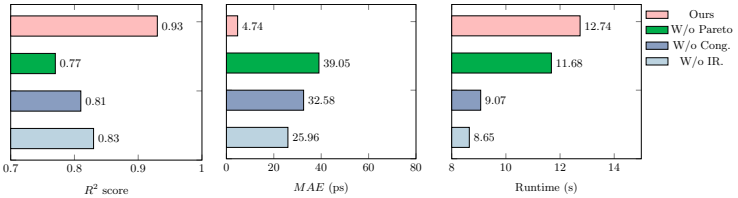
TABLE III Runtime Comparison. The unit is "s".

Cir.	Commerial Flow (20 threads)				Ours	
	Route [8]	IR [7]	STA [9]	Total	Total	Speedup
DMX	20329	77	18	20424	12.56	1626×
GFX	7782	67	16	7865	11.22	701×
AC97	618	72	8	698	4.27	164×
VGA	32727	88	16	32831	9.94	3303×
NOVA	75381	103	45	75529	25.89	2917×
TOP	298	43	9	350	6.17	57×
ECG	35689	72	17	35778	13.65	2621×
ETH	24659	97	12	24768	7.54	3285×
USB	518	67	9	594	5.12	116×
TATE	113192	218	68	113478	31.06	3653×
Ave.	31119	90	22	31231	12.74	1844×



(a) On USB (small circuit). (b) On TATE (large circuit).

Fig. 5 Runtime breakdown of our work.



(a) (b) (c)

Fig. 6 Comparison among different schemes by (a) R^2 score, (b) $MAXE$, and (c) runtime.

For unseen testing designs, these scores are 0.93/4.74ps. After analyzing the results, we summarize our findings below:

- **Highly accurate:** Our framework achieves accurate pre-routing timing prediction for both seen and unseen circuits, generalizing well across designs of varying functions and scales without retraining. This means our model is practical in IC design flow.
- **Triple Modality Advantage:** By fully and effectively integrating netlist, layout, and PDN modality data, our framework outperforms prior single-modal models [2]–[4] and bi-modal approaches [5], [6].
- **Improved Generalization:** Compared with direct fusion and analytical Pareto fusion, our modified Pareto fusion significantly enhances generalization on unseen designs.

B. Runtime Comparison

We demonstrate the speedup of our model compared to an industry-leading commercial tool. The IR drop analysis and routing procedures in modern VLSI design flows are extremely time-consuming due to their high complexity. In contrast, machine learning models offer overwhelming efficiency advantages. The runtime results of our work, shown in TABLE III, include the model inference, graph construction and data preprocessing stage. On average, our proposed flow runs 1844× faster than the commercial tool for evaluating sign-off timing metrics. As illustrated in Fig. 5, the majority of the runtime in our framework is attributed to the data preprocessing stage. It means that there is still a large alteration potential.

C. Ablation Study

This section presents ablation studies. We compare the following schemes: (1) **W/o Pareto**: Multi-modality data fusion is performed without our proposed Pareto training method (Section III-E), relying solely on a single timing loss during training. (2) **W/o Cong.**: The routing congestion-aware encoder (Section III-D) is excluded from the framework. (3) **W/o IR**: The IR drop-aware encoder (Section III-C) is excluded from the framework.

As demonstrated in Fig. 6, our work is compared with other schemes by R^2 score (Fig. 6a), MAE (Fig. 6b) and runtime (Fig. 6c). The results highlight that the inclusion of multimodal data from netlist, layout, and PDN, processed through the IR drop-aware encoder and the routing congestion-aware encoder, is critical for accurate pre-routing timing prediction. Moreover, removing the Pareto multimodal fusion approach significantly degrades model accuracy, as the reliance on a single timing loss is insufficient to capture complex dependencies. In conclusion, the ablation study demonstrates the importance and necessity of the IR drop-aware encoder, routing congestion-aware encoder, and Pareto multimodal fusion method.

V. Conclusion

This work introduces a comprehensive pre-routing timing prediction framework that effectively integrates netlist, layout, and PDN modality data. The proposed IR drop-aware and routing congestion-aware encoders enable detailed modeling of timing information and PDN impacts. The Pareto multimodal fusion algorithm addresses imbalances among circuit modalities during the training of our multimodal timing model, ensuring meaningful contributions from each modality data. Experiments on advanced TSMC 16nm designs validate the framework, demonstrating significant improvements in timing prediction accuracy and efficiency.

Acknowledgment

This work is supported by the National Key R&D Program of China (2022YFB2901100), the National Natural Science Foundation of China (No. 62404021), and the Beijing Natural Science Foundation (No. 4244107, QY24216, QY24204).

REFERENCES

[1] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer, 2011, vol. 312.

[2] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. DAC*, 2019, pp. 1–6.

[3] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead rc network," in *Proc. DAC*, 2022, pp. 1213–1218.

[4] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proc. DAC*, 2022, pp. 1207–1212.

[5] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-Tolerant timing prediction via multimodal fusion," in *ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[6] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu, and M. Wong, "Lay-net: Grafting netlist knowledge on layout-based congestion prediction," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[7] Ansys, "RedHawk User Guide," https://www.ansys.com/products/semiconductors/ansys-redhawk-sc, 2024.

[8] Synopsys, "IC Compiler II User Guide," https://www.synopsys.com/implementation-and-signoff/physical-implementation/ic-compiler.html, 2023.

[9] ——, "PrimeTime User Guide," https://www.synopsys.com/cgi-bin/imp/pdfdla/pdfr1.cgi?file=primetime-wp.pdf, 2023.

[10] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 547–552.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[12] R. Azad, M. Asadi-Aghbolaghi, M. Fathy, and S. Escalera, "Attention deeplabv3+: Multi-level context attention mechanism for skin lesion segmentation," in *European Conference on Computer Vision (ECCV)*. Springer, 2020, pp. 251–266.

[13] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[14] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, "On large-batch training for deep learning: Generalization gap and sharp minima," *arXiv preprint arXiv:1609.04836*, 2016.

[15] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro, "Exploring generalization in deep learning," *Advances in neural information processing systems*, vol. 30, 2017.

[16] OpenCores, http:///opencores.org/, 2023.

[17] Synopsys, "Design Compiler User Guide," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html, 2024.

FactorHD: A Hyperdimensional Computing Model for Multi-Object Multi-Class Representation and Factorization

Yifei Zhou[1], Xuchu Huang[1], Chenyu Ni[1], Min Zhou[1], Zheyu Yan[1,2,*], Xunzhao Yin[1,2,*], Cheng Zhuo[1,2,*]
[1]Zhejiang University, Hangzhou, China; [2]Key Laboratory of Collaborative Sensing and Autonomous Unmanned Systems of Zhejiang Province, China; *Corresponding authors, email: {zyan2, czhuo, xzyin1}@zju.edu.cn

***Abstract*—Neuro-symbolic artificial intelligence (neuro-symbolic AI) excels in logical analysis and reasoning. Hyperdimensional Computing (HDC), a promising brain-inspired computational model, is integral to neuro-symbolic AI. Various HDC models have been proposed to represent class-instance and class-class relations, but when representing the more complex class-subclass relation, where multiple objects associate different levels of classes and subclasses, they face challenges for factorization, a crucial task for neuro-symbolic AI systems. In this article, we propose FactorHD, a novel HDC model capable of representing and factorizing the complex class-subclass relation efficiently. FactorHD features a symbolic encoding method that embeds an extra memorization clause, preserving more information for multiple objects. In addition, it employs an efficient factorization algorithm that selectively eliminates redundant classes by identifying the memorization clause of the target class. Such model significantly enhances computing efficiency and accuracy in representing and factorizing multiple objects with class-subclass relation, overcoming limitations of existing HDC models such as "superposition catastrophe" and "the problem of 2". Evaluations show that FactorHD achieves approximately $5667\times$ speedup at a representation size of 10^9 compared to existing HDC models. When integrated with the ResNet-18 neural network, FactorHD achieves 92.48% factorization accuracy on the Cifar-10 dataset.**

I. Introduction

Neuro-symbolic artificial intelligence (neuro-symbolic AI) combines symbolic methods with modeling techniques to analyze and reason about problems logically [1]. Unlike traditional neural structures that rely on repeated training and weighted connections between neurons, neuro-symbolic AI employs explicit representations that are comprehensible to human brains [2], leverages hyperdimensional computing (HDC) as the computational paradigm, allowing for a deeper understanding of the modeled world. HDC is a promising computational model that offers high computation efficiency and noise tolerance [3–6]. Inspired by the information processing patterns of the human brain, HDC uses hyperdimensional vectors (HVs) to represent and operate data, and supports complete sets of algebraic computation. Existing HDC models accommodate two primary representation relations, using binding-bundling form of representations (similar to disjunctive normal form) to represent objects with class-instance relation [7, 8] and class-class relation [9], respectively. However, the development of neuro-symbolic AI demands HDC to represent more complex relations such as the class-subclass relation, where multiple objects associate different levels of classes and subclasses [10], as shown in Fig. 1(a). The class-subclass relation exhibits a hierarchical structure known as 'class-instance','type-token' or 'is-a' and is related to categorization or taxonomy hierarchy.

When representing the class-subclass relation, prior HDC models inevitably face the challenges for factorization, which plays a central role in the reasoning of HDC, as shown in Fig. 1(c). These challenges include "superposition catastrophe" [11], where the subclass items of multiple objects simultaneously become mixed and indistinguishable, and "the problem of 2" [12], which induces information loss when several identical objects are represented simultaneously. Another challenge faced by HDC models is the inefficiency in factorization as the size of class-subclass hierarchy for representation (i.e., the number of class and subclass items) scales up. Existing HDC models [13, 14] often rely on numerous iterations of unbinding operations and similarity measurements to reach high factorization accuracy, resulting in significant time costs and large number of computations. Moreover, even when only a subset of subclasses are of interest, current HDC models still require complete factorization of the target object HVs [15]. These limitations hinder the neuro-symbolic application in a broader range of scenarios.

To address above challenges, we propose FactorHD, which consists of a symbolic encoding that can represent multiple objects with multiple levels of subclasses, and a factorization algorithm that can efficiently factorize encoded object representations for the selected subset of subclass items. The main contributions of our work can be summarized as follows:

- We introduce FactorHD, a neuro-symbolic HDC model that supports the representation and factorization of multiple objects with class-subclass relation hierarchy. Compared to prior binding-bundling form of representations, FactorHD employs a novel symbolic encoding method that embeds an extra bundling clause to connect items and a redundant class label. Such bundling-binding-bundling form of representation preserves more information for multiple objects with multiple class and subclass levels, which can still be factorized without causing "superposition catastrophe" and "the problem of 2".
- We developed a factorization algorithm for FactorHD that offers superior computational efficiency and a reduced number of computations while maintaining high factorization accuracy. In contrast, prior algorithms require

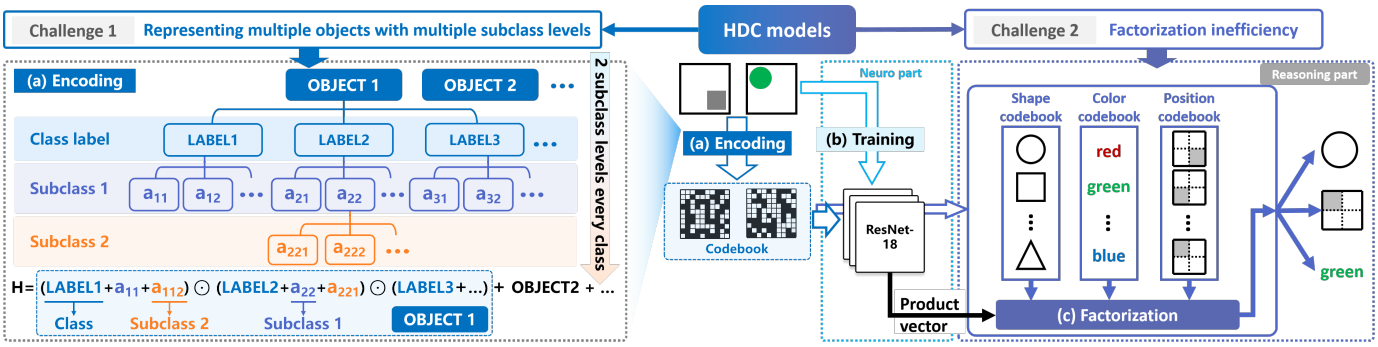


Fig. 1. (a) An example of class-subclass relation with 3 classes (labels) and 2 subclass levels in hierarchy. FactorHD represents this relation by bundling subclass levels within the same class and binding different classes together. The encoding process generates HV codebooks. Existing HDC models struggle with such complex representations. (b) The training processes involve the implementation of ResNet-18 network as the neuro part of neuro-symbolic AI model. (c) The factorization process given an object query. The HDC challenges involve representation and factorization.

iterative exploration of the entire representation hierarchy. Additionally, our proposed algorithm allows for partial factorization of specific subclasses of interest, eliminating the need for a complete and costly factorization process.

- Evaluation results demonstrate that FactorHD can factorize significantly larger representations, with faster speeds and reduced number of computations. The factorization time complexity of FactorHD is approximately $O(N_M)$, where N_M represents the number of subclass items within each class. Moreover, FactorHD shows lower declining rate in factorization accuracy as the problem size grows, exhibiting high accuracy and better scalability compared to other designs. Integrated with the ResNet-18 neural network as the neuro part, FactorHD also shows high accuracy on training and factorizing practical datasets such as RAVEN [16], Cifar-10 and Cifar-100.

II. Background and Motivation

In this section, we provide an overview of HDC models, then discuss the challenges of representation and factorization for existing HDC designs.

A. Hyperdimensional Computing Basics

HDC was originally proposed in cognitive neuroscience for symbolic reasoning [17], based on the observation that the human brain operates on high-dimensional data [3]. HDC designs are significantly powerful in reasoning and association of the abstract information, while they are weak in features extraction from complex data, e.g., image/video [18]. Symbolic representations in HDC are based on the quasi-orthogonality characteristic of HVs, implying that randomly generated HVs are nearly orthogonal and dissimilar to each other [19].

HDC measures the similarity between HVs during inference for reasoning [19] and factorization [7, 13]. Here we assume that $\vec{V_1}$, $\vec{V_2}$ are two randomly generated quasi-orthogonal HVs with $\vec{V_1}$,$\vec{V_2}$ $\in \{-1,+1\}^D$, where D is the dimensionality of HVs. Common similarity metrics include cosine similarity [20, 21], Hamming distance [22–24] and dot product [25, 26]. A similarity metric close to 0 implies that the HVs are quasi-orthogonal. In this design, we use dot product similarity (denoted as $sim(.,.)$) to recognize target HVs, and the similarity is computed by $sim(\vec{V_1}, \vec{V_2}) = \vec{V_1} \cdot \vec{V_2}/D$.

HDC also supports complete sets of operators for computation, mainly includes bundling ($+$) that acts as memorization during HV addition, binding ($\odot$) that associates multiple HVs, unbinding that disassociate HVs, and permutation (ρ) that preserves the position information within a sequence [27].

Bundling ($+$) of two HVs $\vec{V_1}$ and $\vec{V_2}$ is performed by component-wise addition. The bundled HVs preserves high similarity to its component HVs, i.e., $sim(\vec{V_1}+\vec{V_2}, \vec{V_1}) \gg 0$. Hence, we can identify the components by similarity measurement. Note that in this design, we restrict and clip the component values of bundling results of single object to the range of $\{-1, 0, 1\}$. However, when bundling HVs of different objects, we retain the results in Z^D as prior designs [7].

Binding ($\odot$) of two HVs $\vec{V_1}$ and $\vec{V_2}$ is performed by component-wise multiplication. The resultant HV is dissimilar to any of its constituent vectors, i.e., $sim(\vec{V_1} \odot \vec{V_2}, \vec{V_1}) \approx 0$.

Unbinding of HVs is achieved through the same arithmetic operation used for binding when the operation is self-inverse. In this case, the binding of two identical HVs results in a constant value of 1, i.e., $\vec{V_1} \odot \vec{V_1} = 1$.

B. Challenges for Representation and Factorization

Existing HDC models [7–9, 13, 28, 29] accommodate two primary representation relations, using binding-bundling form to represent objects with class-instance relation [8] and class-class relation [9]. Consequently, these models can be categorized into two types based on their representation structures: the class-instance model (C-I model) and the class-class model (C-C model). Within C-I model [7, 8, 29], objects are represented by binding class and instance (i.e., subclass) HVs. Bundling operations are used to combine different classes. For example, to represent a brown dog, the object is encoded as: $object = animal \odot dog + color \odot red$, where $animal$ and $color$ are class HVs, while dog and red are instance HVs. To partially factorize the object, i.e., the $color$ of the object, the unbinding process applies: $color \odot object = red + noise$. As a result, the C-I model is effective in factorization and obtaining partial information. Within C-C model [9, 13, 28], binding connects different classes while bundling connects multiple objects. For instance, the previous example combining another white cat can be represented as: $objects = dog \odot red + cat \odot$

$white$. Therefore, the C-C model is capable of representing multiple objects simultaneously [19].

However, when representing the more complex class-subclass relations, where multiple objects associate different levels of classes and subclasses (e.g., animals-dogs-spaniels-Fido) [10], existing HDC models all inevitably face challenges for factorization. The C-I model face the challenges of "superposition catastrophe" [19] for multiple objects representation in the class-subclass relations, and "the problem of 2" [12] when representing several identical objects. Even though the C-C model can avoid the above problems when representing multiple objects, they still face inefficiency in factorization as the size of representation scales up. Given F class or subclass codebooks $A_i = \{a_{i1}, a_{i2}, \ldots, a_{iM}\}$ with M holographic bipolar item HVs ($a_{ij} \in \{-1, +1\}^D, i = 1, 2, \ldots, F$, $j = 1, 2, \ldots, M$), the goal is to unbind the representation product vector H (e.g., $H = a_{11} \odot a_{23} \odot a_{32}$) to identify subclass items, i.e., a_{11}, a_{23} and a_{32}. Without loss of generality, when we factorize a_{11} from H, we must identify and eliminate term $a_{23} \odot a_{32}$, necessitating exploration of all item vector combinations and similarity measurements. If $F = 3$ and $M = 256$, factorizing a specific item HV of an encoded object HV may require $M^F = 16,777,216$ similarity measurements. As the size of representation for factorization increases, this number of repeated unbinding operation grows rapidly. Moreover, in many scenarios where only a subset of class and subclass items are of interest, current C-C models still needs to factorize all composing item vectors, resulting in computation wastes.

To address the factorization challenge, various models have been proposed for the C-C model. The nonlinear dynamical resonator network [14] provides a neural solution to the factorization problem by iteratively searching for the F terms across the set of possible solutions, assuming that each codebook contains a finite set of M code vectors. Another approach [13] designed a novel in-memory stochastic factorizer (IMC factorizer) to handle larger problem sizes (i.e., size of representation hierarchy) with high accuracy. At $D = 256$, $F = 3$ and $M = 256$, the resonator network [14] becomes incapable of factorizing the representation vectors, while the IMC factorizer [13] achieves an accuracy of 99.71% with an average 3312 iterations. Nevertheless, C-C models still exhibit relatively low computing efficiency when factorizing larger problems.

III. FactorHD for Multi-Object Multi-Class Representation and Factorization

In this section, we introduce FactorHD, a novel neuro-symbolic HDC model that provides enhanced symbolic representation and factorization capability with complex hierarchy. We discuss the encoding and factorization method for 3 different symbolic representations of FactorHD. We finally investigate the selection of optimal TH values for factorizing representations with multiple objects.

A. FactorHD Encoding for Representation

Here we introduce the proposed symbolic encoding method for FactorHD representation with class-subclass relation hierarchy. Fig. 1(a) shows an example of the proposed class-subclass relation hierarchy with two subclass levels represented in bundling-binding-bundling form. In practical scenarios, fixed labels are assigned to each class to distinguish different classes. We use $LABELi$ to denote the HVs of the top class a_i, as the top level class is often used as the label in the traditional class-instance classification. The class-subclass relation of different levels in a class is known in advance, such as "dog is animal", and notably, our focus is on identifying whether an object contains an item of a specific class or subclass, rather than determining hierarchical levels.

Algorithm 1: FactorHD algorithm of HV factorization

Input: Input the encoded target HVs **s**, the number of different classes **F**, the dimension of HVs **D**, each class includes **L1** subclasses in subclass 1 level, each subclass in subclass 1 level includes **L2** subclasses in subclass 2 level, the base HV dictionary for subclass 1 level **IM1=[im1**$_j$**]**$_{F\times L1\times D}$ and subclass 2 level **IM2 = [im2**$_j$**]**$_{F\times L1\times L2\times D}$, etc., the labels **L = [L**$_j$**]**$_{F\times D}$, the similarity threshold **TH**, the number of objects **obj**.
Output: Factorized items of each subclass levels **R1**, **R2**, etc.

```
while s is not pure noise do
    for f ← 1 to F do
        s_f ← s;
        for f_dec ← 1 to F do
            if f_dec ≠ f then
                s_f ← bind s_f and L[f_dec];
        for i ← 1 to L1 do                    /* sim(,): similarity measurement. */
            S1[f, i] ← sim(s_f, IM1[f, i]);
    if obj == 1 then                          /* foreknowledge of 1 object */
        R1 ← Index of the largest term of each class in S1;
        Get R2, R3, ... of other subclass levels using similar process as line 7-10;
    if obj > 1 or unknown then
        ID ← all the indices where S1 > TH for subclass 1 level;
        C ← all possible item combinations of ID;
        for comb in C do
            TM ← initialize to 1;
            for f ← 1 to F do
                TM ← bind TM and IM1[f, comb[f]];
                R1[f] ← comb[f];
            if sim(TM, s) > TH then
                flag ← 1;
                Get R2, R3, ... of other subclass levels using similar process as line 7-21;
                break for loop;
        if not flag == 1 then                 /* s becomes pure noise. */
            break while loop;
        else
            Reconstruct the object HV s_tmp using results in R1, R2,...;
            s ← s − s_tmp;
```

To encode objects in FactorHD, different subclass levels belonging to the same class can be considered equally and combined together using the bundling operation, and different classes are linked using the binding operation. We then use the bundling operation to connect different objects just as a class-class structure, forming a bundling-binding-bundling structure. An example of FactorHD encoding is $H = (LABEL1 + a_{11} + a_{112}) \odot (LABEL2 + a_{22} + a_{221}) \odot \ldots$, which includes redundant class labels $LABEL1$, $LABEL2$, and subclass items a_{11}, a_{112}, a_{22} and a_{221}, etc. Among them, a_{11} belongs to class a_1, a_{112} belongs to a_{11}, and a_{22} belongs to class a_2, a_{221} belongs to a_{22}. Note that even if the items of a specific class is not associated with an object, FactorHD still reserves its class label and bundles the label with a global HV representing 'NULL'. With such encoding approach, FactorHD eliminates the need for foreknowledge of specific classes associated in the object, while C-C models require exhaustive factorization if contained class is unknown [13, 14]. Below we elaborate on how our proposed FactorHD model factorize different encoded object representations with class-subclass relation.

B. FactorHD Factorization Algorithm

Algorithm. 1 summarizes the proposed factorization algorithm of FactorHD. The loop starts from the original target HV. For each selected class of the object, we first unbind all other unselected class labels. Then, we compute the similarity between the unbound HV and all subclass items of the selected class. When the target HV only represents single object, we simply select the subclass item with the largest similarity to the unbound HV as the factorized subclass item of the selected class, and repeat the similarity measurement and subclass item selection process for all subclass levels. On the other hand, when the target HV represents multiple objects, or the number of objects is unknown, we need to select all the subclass items at the same subclass level with a similarity larger than a predefined "threshold similarity" (TH) to their respective unbound class HVs, as these subclass items are possibly associated with the multiple objects. These subclass items are then selectively bound, such that each of the resultant product HVs contain exactly one subclass item of every subclass. If one of the combinations is similar to the target HV, i.e., the similarity is larger than TH, it indicates that the target HV contains an object possessing the subclass items of this specific combination at this subclass level. We then repeat the similarity measurement and subclass item selection process for lower subclass levels. Only when the subclass items at all subclass levels are determined, we infer that the corresponding object possessing all these subclass items exists in the target HV. The HV of the object will then be reconstructed using the factorized class labels and subclass items, and excluded from the target HV. The new target HV will move to the next loop until no more objects can be factorized from the target HV.

To further illustrate the algorithm, we discuss the factorization for 3 different target class-subclass representations, from the common case to more complex cases.

Representation 1 (Rep 1): Fig. 2(a) shows the factorization process for single object with single subclass level. The unconsidered class labels can be eliminated by binding the same label: $(LABEL2 + a_{21}) \odot LABEL2 \approx 1$. In this way, we factorize the information associated with the selected class:

$$TargetHV \odot LABEL2 \odot \ldots \approx (LABEL1 + a_{12}) \quad (1)$$

Eq. 1 shows the unbinding result only contains the bundling clause with the selected class $LABEL1$. By computing the similarity between the unbinding result with all the subclass item HVs of class $LABEL1$, we can easily identify the subclass item with the highest similarity as the factorized one.

Representation 2 (Rep 2): Fig. 2(b) illustrates the factorization process for single object with multiple (e.g., 2) subclass levels. On the basis of Rep. 1, after factorizing out the item at subclass 1 level, i.e., a_{12}, we compute similarity specifically with the item HVs under a_{12}, and select the item HV with the highest similarity again, i.e., a_{121}, as the factorized item at subclass 2 level. Such process can continue to subsequent subclass levels, which minimizes the number of similarity computations required to factorize all subclass items of the target object by searching from higher to lower levels.

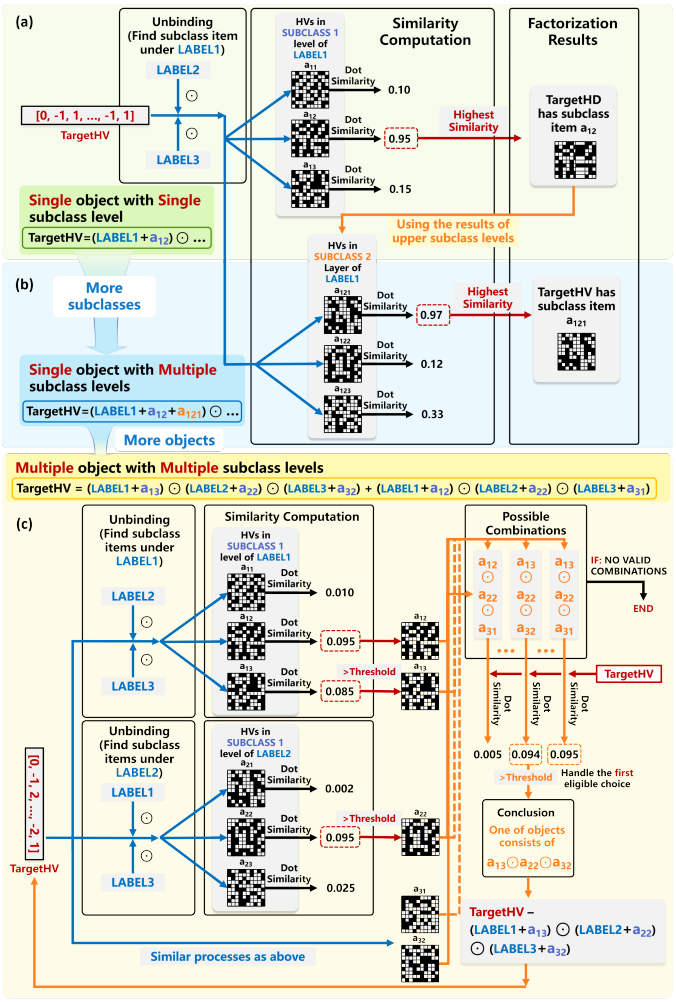


Fig. 2. (a) Illustration of FactorHD factorizing the common Rep 1 case. (b) Factorizing single object with multiple subclass levels (Rep 2). (c) Factorizing the multiple objects with multiple subclass levels (Rep 3).

Representation 3 (Rep 3): In the most complex scenario, we handle multiple objects with multiple subclass levels. Current methods compare the target HV with all subclass combinations, which is computationally intensive. When factorizing target HV with multiple objects expressed in Fig. 2(c) using FactorHD, simply binding all unselected class labels with $TargetHV$ is not sufficient, as multiple subclass items under the same class remain. For instance, binding $TargetHV$ with $LABEL2$ and $LABEL3$ results in multiple subclass items of class $LABEL1$, i.e., a_{13} and a_{12}, and selecting one subclass item each time may cause "superposition catastrophe". Therefore, FactorHD only considers items with a similarity higher than TH. After unbinding classes and selecting subclass items, though we cannot determine specific item combinations, we can determine that the class $LABEL1$ contains a_{12} and a_{13}, the class $LABEL2$ contains a_{22}, and the class $LABEL3$ contains a_{31} and a_{32}. Theoretically, only 4 subclass item combinations need to be considered (e.g., $a_{12} \odot a_{22} \odot a_{31}$). If any combination exceeds TH, we conclude that $TargetHV$ contains that specific combination. Further processes have been detailed in Algorithm. 1. FactorHD significantly reduces

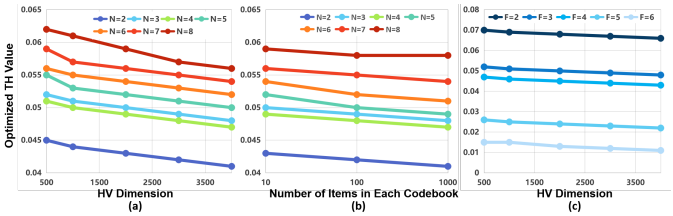


Fig. 3. The optimal value of TH for factorizing representations of multiple objects with a single subclass level varies with (a) HV dimension D and the number of objects N given $M = 10$ and $F = 4$, (b) the number of items in each codebook M given $D = 2000$ and $F = 4$, and (c) the number of factors F in the composite HVs given $N = 3$ and $M = 10$.

the number of combinations needed for similarity computation, addresses the "superposition catastrophe" [11] and "the problem of 2" [12], and adapts to any number of objects.

Fig. 3 illustrates the method for selecting the optimal TH value TH^* across specific factorization problems. TH^* is defined as the value such that FactorHD achieves the highest factorization accuracy on Rep 3. TH^* is determined by HV dimension, representation structure, and factorization problem complexity, increases with the number of objects N and decreases with the number of factors F. Moreover, TH^* shows a roughly linear relationship with HV dimension D and the logarithmic value of M. Therefore, given N and F, TH^* can be approximately fitted and predicted. And values near TH^*, though not the optimal, also yield high factorization accuracy. Thus, a rough value selection can be fit as follows:

$$TH^* = 0.001(104 + 2N - 15F - 0.001D - log(M)) \quad (2)$$

In contrast, the performance of state-of-the-art (e.g., the IMC Factorizer) heavily depends on the selection of specific activated values and their thresholds [13], even for factorizing Rep 1, and these values are completely irregular.

IV. Performance Evaluations

In this section, we validate and evaluate the performance of FactorHD. We first compare it with other HDC models on the common Rep 1 problem. Then, we evaluate the factorization accuracy of FactorHD on Rep 2 and Rep 3, as well as on practical datasets such as RAVEN, Cifar-10, and Cifar-100.

A. Experimental Setup

Algorithmic experiments are implemented on the GeForce RTX 2080 GPU platform. For the basic experiments on Rep 1-3, we perform 1024 factorization trials respectively, with batch size of 512. We first evaluate the performance of FactorHD on Rep 1, comparing with C-C models on factorization accuracy and time cost. The HV dimensions used here are consistent with those in [13, 14, 30], namely $D = 1500$ for $F = 3$ and $D = 2000$ for $F = 4$. FactorHD operates in $\{-1, 0, 1\}^D$ space, using 2 bits per dimension, thus D of FactorHD reduces by half to match the storage space of other models to ensure fair comparisons. The number of code vectors in each class codebook M is varied, determining the total problem size as M^F. Then when comparing to C-I models [28], we set $D = 256$ for $F = 3$ and $D = 512$ for $F = 4$ in the factorization of C-I models. We similarly reduce D for FactorHD by half.

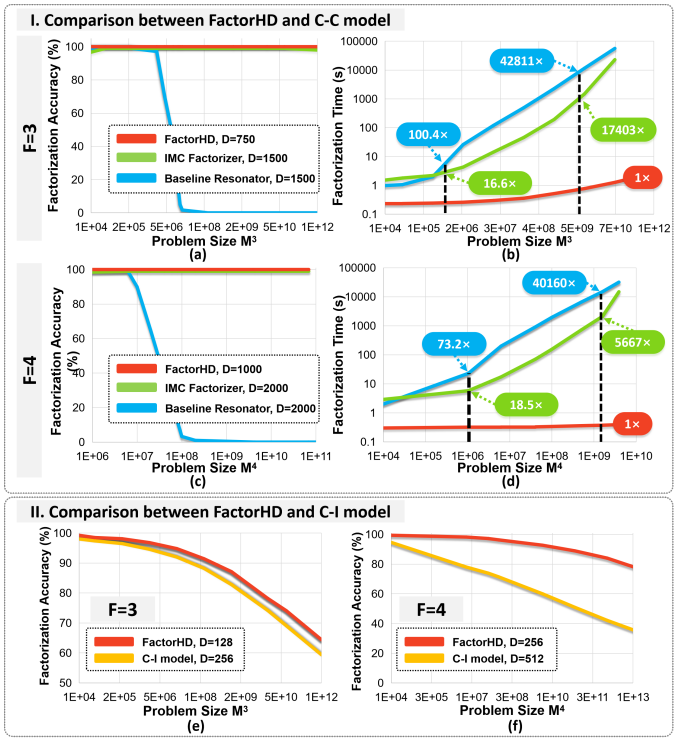


Fig. 4. I. The factorization performance comparison between FactorHD and C-C models using baseline resonator network, IMC factorizer with varying HV dimensionality: When $F = 3$, (a) the factorization accuracy and (b) factorization time with scaling problem size M^3. When $F = 4$, (c) the factorization accuracy and (d) factorization time with scaling problem size M^4. II. The factorization performance comparison between FactorHD and C-I models when (e) $F = 3$ and (f) $F = 4$ with varying HV dimensionality.

Given that prior HDC designs fail to accommodate complex representations, we simply evaluate FactorHD on Rep 2 and 3. The testing representations include one or two objects, each with two subclass levels. The top-level classes consist of 256 subclasses, each having 10 sub-subclasses.

Finally, we present the training and factorization performance of FactorHD with the ResNet-18 network as the neuro part for feature extraction, using the RAVEN [16], Cifar-10, and Cifar-100 datasets. The RAVEN dataset includes objects characterized by attributes like type, size, color, and position. The first codebook represents the position attribute, the second represents color, and the third combines size and type attributes, resulting in 30 size-type combinations. Image patterns in RAVEN dataset consists of 1-9 objects. Cifar-10 images are encoded by binding the image label with a dummy label. Cifar-100 datasets naturally have two class levels (coarse and fine labels), the two levels of labels are bound for encoding, and later factoring out either the coarse or fine information for specific images. During training, images containing both coarse and fine aspect information are fed into neural network to extract features, which are then encoded into HVs. During inference, HVs are factorized to reveal the specific composition of the images.

B. Experimental Results

Rep 1. Fig. 4(a-d) show the factorization accuracy and factorization time of FactorHD and C-C models using dif-

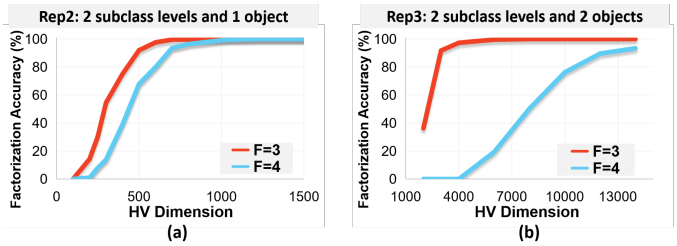


Fig. 5. The performance of FactorHD on factorizing (a) Rep 2 and (b) Rep 3 with varying HV dimensionality.

ferent factorization methods with $F = 3$ and $F = 4$, respectively. The results suggest that FactorHD maintains a high factorization accuracy above 99% even at lower HV dimensions, maintaining stability as problem size increases. On the contrary, the C-C model using resonator network fails when the problem size reaches 10^6, and the IMC factorizer drops from the accuracy of 99% at problem size of 10^{12}.

Moreover, FactorHD exhibits exceptional factorization efficiency. By employing a novel symbolic encoding method that embeds an extra HDC bundling clause and factorization algorithms, the factorization process requires much less iterations, resulting in significantly reduced time cost and making the factorization process well-suited for parallel computing. Consequently, as the problem size grows, the factorization time increase of FactorHD is negligible compared to approaches factorizing C-C models. The time complexity of FactorHD is approximately $O(N_M)$, where N_M represents the maximum number of subclass items within each class, while the IMC factorizer [13] and resonator network [14] have a time complexity larger than $O(N_M^2)$. Thus, the speedup of FactorHD over existing models on factorization grows with problem size, achieving a minimum speedup of $18.5\times$ at 10^6 problem size and reaching $5667\times$ at 10^9 problem size.

Fig. 4(e) and (f) further demonstrate the factorization accuracy of FactorHD and the baseline C-I model with $F = 3$ and $F = 4$, respectively. The results show that FactorHD achieves higher factorization accuracy across various problem sizes compared to the simpler C-I model, demonstrating its ability to maintain high accuracy while representing more complex structures. The factorization times for both models are comparable across different problem sizes due to the similar algorithms of label elimination during factorization.

Rep 2&3. Fig. 5(a) shows the FactorHD performances on factorizing Rep 2. Compared to Rep 1, factorizing representations with multiple subclass levels requires higher HV dimensions due to larger problem sizes and more complex structures. Nevertheless, factorization accuracy reaches 100% at $D = 1000$. Fig. 5(b) shows the FactorHD performances on factorizing Rep 3. In this case, the algorithm does not require prior knowledge of the actual number of objects, necessitating higher HV dimensions to achieve high factorization accuracy.

Inference accuracy. We evaluate FactorHD's accuracy on the RAVEN [16], Cifar-10 and Cifar-100 datasets. Table I shows the accuracy of factorizing RAVEN test sets. Training and factorization with $D = 1000$ achieve over 90% accuracy for most RAVEN patterns. With reduced HV dimensionality, decent accuracies are still maintained. Table II further shows the factorization accuracy of FactorHD integrated with ResNet-18 on the Cifar-10 and Cifar-100 datasets. Compared to the classification accuracy using ResNet-18 neural network [31] and other emerging neural networks (e.g., WideResNet-28, WideIsoNet-28) in traditional neural classification tasks, the accuracy loss of FactorHD model integrated with ResNet-18 is less than 3% for Cifar-10. The accuracy loss can be further minimized with an increase in HV dimension. We also adjust the number of bundled image inputs during training to assess the ability to factorize superposition [29]. FactorHD still gets high factorization accuracy with image inputs in superposition, particularly in the simpler Cifar-10 dataset, thus improving the training efficiency with reduced training epochs. Compared to MIMOConv [29] that specifically designed for bundled images training, the testing results demonstrate the stability of our FactorHD, maintaining relatively high accuracy even integrated with simpler neural networks, i.e., ResNet-18.

TABLE I
FACTORIZATION ACCURACY ON RAVEN TEST SET

	Center	2x2	3x3	L-R	U-D	O-IC	O-IG
D=1000	100.00	90.04	75.88	99.62	99.96	99.93	82.23
D=500	99.88	80.62	67.44	94.67	95.02	97.23	72.57

Training: 40 epochs at D=1000 and 20 epochs at D=500.

TABLE II
FACTORIZATION ACCURACY ON CIFAR-10 AND CIFAR-100 TEST SET

# inputs/pass		Cifar-10		Cifar-100	
		1	2	1	2
FactorHD + ResNet-18	D=64	92.19 (-2.70*)	89.19	69.91 (-6.73#)	59.60
	D=128	92.48 (-2.41*)	89.71	70.58 (-6.06#)	63.07
ResNet-18 [24]		94.89	\	76.64	\
WideResNet-28-10		96.82	\	81.62	\
WideIsoNet-28-10		97.31	\	82.38	\
MIMOConv static [22]		97.49	96.93	83.19	82.30

* Accuracy loss comparing to software training results in the work of Kodge et al. [24]

V. CONCLUSION

In this article, we propose FactorHD, a novel neuro-symbolic model that overcome the challenges of HDC representation and factorization for multiple objects with multiple subclass levels. FactorHD employs a novel symbolic encoding method, coupled with an efficient factorization algorithm that efficiently identifies items of interest without exhaustive similarity measurements. Experimental evaluations demonstrate that FactorHD not only improves speed and accuracy for the common representation but also effectively handles complex representations involving multiple objects and subclass levels. Compared to existing HDC models on factorizing the common representation, i.e., single object with single subclass level, FactorHD achieves approximately $18.5\times$ speedup at the problem size of 10^6, and $5667\times$ speedup at the problem size of 10^9, while maintaining 99% accuracy. Furthermore, FactorHD integrated with ResNet-18 neural network achieves 92.48% factorization accuracy on the Cifar-10 dataset.

ACKNOWLEDGMENT

This work was partially supported by NSFC (92164203) and SGC Cooperation Project (Grant No. M-0612).

REFERENCES

[1] S. Chaudhuri, K. Ellis, O. Polozov, R. Singh, A. Solar-Lezama, and Y. Yue, "Neurosymbolic programming," *Foundations and Trends in Programming Languages*, vol. 7, pp. 158–243, 12 2021.

[2] M. K. Sarker, L. Zhou, A. Eberhart, and P. Hitzler, "Neuro-symbolic artificial intelligence," *AI Communications*, vol. 34, pp. 197–209, 3 2022.

[3] P. Kanerva, *Hyperdimensional Computing: An Algebra for Computing with Vectors*, pp. 25–42. 2023.

[4] M. Imani *et al.*, "Searchd: A memory-centric hyperdimensional computing with stochastic training," *IEEE TCAD*, vol. 39, no. 10, pp. 2422–2433, 2019.

[5] X. Yin, H. E. Barkam, F. Müller, Y. Jiang, M. Imani, S. Abdulazhanov, A. Vardar, N. Laleni, Z. Zhao, J. Duan, *et al.*, "A remedy to compute-in-memory with dynamic random access memory: 1fefet-1c technology for neuro-symbolic ai," *arXiv preprint arXiv:2410.15296*, 2024.

[6] Q. Huang, Z. Yang, K. Ni, M. Imani, C. Zhuo, and X. Yin, "Fefet-based in-memory hyperdimensional encoding design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 3829–3839, 2023.

[7] P. Poduval, H. Alimohamadi, A. Zakeri, F. Imani, M. H. Najafi, T. Givargis, and M. Imani, "Graphd: Graph-based hyperdimensional memorization for brain-like cognitive learning," *Frontiers in Neuroscience*, vol. 16, p. 757125, 2022.

[8] P. Kanerva, "What we mean when we say" what's the dollar of mexico?": Prototypes and mapping in concept space," in *2010 AAAI fall symposium series*, 2010.

[9] M. Hersche, M. Zeqiri, L. Benini, A. Sebastian, and A. Rahimi, "A neuro-vector-symbolic architecture for solving raven's progressive matrices," *Nature Machine Intelligence*, vol. 5, pp. 363–375, 3 2023.

[10] D. A. Rachkovskij, E. M. Kussul, and T. N. Baidyk, "Building a world model with structure-sensitive sparse binary distributed representations," *Biologically Inspired Cognitive Architectures*, vol. 3, pp. 64–86, 1 2013.

[11] D. Rachkovskij, "Representation and processing of structures with binary sparse distributed codes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, pp. 261–276, 2001.

[12] R. Gayler, "Vector symbolic architectures answer jackendoff's challenges for cognitive neuroscience," in *ICCS/ASCS International Conference on Cognitive Science*, 01 2003.

[13] J. Langenegger, G. Karunaratne, M. Hersche, L. Benini, A. Sebastian, and A. Rahimi, "In-memory factorization of holographic perceptual representations," *Nature Nanotechnology*, vol. 18, pp. 1–7, 03 2023.

[14] E. P. Frady, S. J. Kent, B. A. Olshausen, and F. T. Sommer, "Resonator networks, 1: An efficient solution for factoring high-dimensional, distributed representations of data structures," *Neural Computation*, vol. 32, pp. 2311–2331, 12 2020.

[15] D. Kleyko, D. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part ii: Applications, cognitive models, and challenges," *ACM Comput. Surv.*, vol. 55, jan 2023.

[16] C. Zhang, F. Gao, B. Jia, Y. Zhu, and S.-C. Zhu, "Raven: A dataset for relational and analogical visual reasoning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[17] T. A. Plate and G. Hinton, *Distributed Representations and Nested Compositional Structure*. PhD thesis, CAN, 1994. AAINN97247.

[18] P. Poduval, M. Issa, F. Imani, C. Zhuo, X. Yin, H. Najafi, and M. Imani, "Robust in-memory computing with hyperdimensional stochastic representation," Institute of Electrical and Electronics Engineers Inc., 2021.

[19] D. Kleyko, D. A. Rachkovskij, E. Osipov, and A. Rahimi, "A survey on hyperdimensional computing aka vector symbolic architectures, part i: Models and data transformations," *ACM Computing Surveys*, vol. 55, 12 2022.

[20] T. Yu, Y. Zhang, Z. Zhang, and C. M. De Sa, "Understanding hyperdimensional computing for parallel single-pass learning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1157–1169, 2022.

[21] C.-K. Liu *et al.*, "Cosime: Fefet based associative memory for in-memory cosine similarity search," in *IEEE/ACM ICCAD*, pp. 1–9, 2022.

[22] P. Kanerva *et al.*, "Fully distributed representation," *PAT*, vol. 1, no. 5, p. 10000, 1997.

[23] K. Ni, X. Yin, A. F. Laguna, S. Joshi, S. Dünkel, M. Trentzsch, J. Müller, S. Beyer, M. Niemier, X. S. Hu, *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[24] X. Yin, F. Müller, Q. Huang, C. Li, M. Imani, Z. Yang, J. Cai, M. Lederer, R. Olivo, N. Laleni, *et al.*, "An ultracompact single-ferroelectric field-effect transistor binary and multibit associative search engine," *Advanced Intelligent Systems*, vol. 5, no. 7, p. 2200428, 2023.

[25] T. A. Plate, "Holographic reduced representations," *IEEE Transactions on Neural networks*, vol. 6, no. 3, pp. 623–641, 1995.

[26] X. Yin, Q. Huang, H. E. Barkam, F. Müller, S. Deng, A. Vardar, S. De, Z. Jiang, M. Imani, U. Schlichtmann, *et al.*, "A homogeneous fefet-based time-domain compute-in-memory fabric for matrix-vector multiplication and associative search," *IEEE TCAD*, 2024.

[27] L. Ge and K. K. Parhi, "Classification using hyperdimensional computing: A review," *IEEE Circuits and Systems Magazine*, vol. 20, no. 2, pp. 30–47, 2020.

[28] G. Camposampiero, M. Hersche, A. Terzić, R. Wattenhofer, A. Sebastian, and A. Rahimi, "Towards learning abductive reasoning using vsa distributed representations," in *Neural-Symbolic Learning and Reasoning*, (Cham), pp. 370–385, Springer Nature Switzerland, 2024.

[29] N. Menet, M. Hersche, G. Karunaratne, L. Benini, A. Sebastian, and A. Rahimi, "Mimonets: Multiple-input-multiple-output neural networks exploiting computation in superposition," in *Advances in Neural Information Processing Systems* (A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, eds.), vol. 36, pp. 39553–39565, Curran Associates, Inc., 2023.

[30] S. J. Kent, E. P. Frady, F. T. Sommer, and B. A. Olshausen, "Resonator networks, 2: Factorization performance and capacity compared to optimization-based methods," *Neural Computation*, vol. 32, pp. 2332–2388, 12 2020.

[31] S. Kodge, G. Saha, and K. Roy, "Deep unlearning: Fast and efficient gradient-free class forgetting," *Transactions on Machine Learning Research*, 2024.

Accelerating design-technology co-development using neural compact modeling and data-driven SPICE simulation.

Yongjeong Lee
Samsung Electronics Co. Ltd.
Hwasung-si, Korea
Yongj.lee@samsung.com

Seungsoo Lee
Samsung Electronics Co. Ltd.
Hwasung-si, Korea
Ss2.lee@samsung.com

Jeongyeol Kim
Samsung Electronics Co. Ltd.
Hwasung-si, Korea
Passion.kim@samsung.com

Jungyun Choi
Samsung Electronics Co. Ltd.
Hwasung-si, Korea
Jungyun74.choi@samsung.com

Zhaojie Li
Synopsys Inc.
Shanghai, China
Zhaojie.li@synopsys.com

Dehuang Wu
Synopsys Inc.
Shanghai, China
Dehuang.wu@synopsys.com

Joddy Wang
Synopsys Inc.
Sunnyvale, USA
Joddy.wang@synopsys.com

***Abstract*— This paper proposes a new design-technology co-optimization framework that expedites circuit optimization by utilizing the neural compact modeling (NCM) and a data-driven SPICE simulation. An efficient retargeting strategy of NCM and its improved design capability through a direct data driven SPICE simulation were leveraged at the industry level in response to increasingly challenging current development situations. To facilitate rapid feedback for extensive trial and error in technology optimization, the NCM swiftly fine-tune itself using pre-trained model. Then, the data interpolation and derating techniques are utilized to provide the same design environment as before such as instance binning, process variations, and layout dependent effects. Demonstrating the robustness of our framework, we achieved a 95% reduction in PDK release time while maintaining model consistency and performance at a mid-scale design of 15k transistors, with no SPICE run time and accuracy loss. This solution allows for rapid incorporation of process changes into the design, supporting quick path-finding during a design-technology co-development.**

Keywords—Neural compact model, Data-driven SPICE model, Design-technology co-optimization

I. INTRODUCTION

In recent years, there has been an increasing demand for the concurrent development of technology and design, that chip design and technology optimization are carried out in tandem rather than sequentially. This approach aims to streamline development and improve product performance in the face of today's increasing complexity of technology, which necessitates extensive design explorations and Design-technology co-optimization (DTCO). Although rapid feedback of process changes to circuit designers is important for quick path-finding in this stage, the slow turnaround time (TAT) for model parameters extractions (MPE) of compact models for the SPICE simulation is a major bottleneck in such co-development efforts. A compact model is a crucial component that bridges technology and design by implementing device characteristics in SPICE simulations for integrated circuits. When device characteristics change, a new model must be developed through MPE. However, because hundreds of model parameters (MPs) in conventional model such as BSIM [1] need to be adjusted, this process is time-consuming and tedious. As a result, frequent process changes can lead to significant delays in product development schedules due to the long TAT of MPE.

So far, efforts have been made to automate the MPE of conventional equation-based compact models using genetic algorithms [2] or deep learning [3, 4] to reduce the MPE's TAT. However, these algorithms face challenges in defining which parameters to use, determining their sequence and range. Also, managing hundreds of MPs impose complexity and limitation on achieving model accuracy. Therefore, fully automating the development of traditional equation-based model optimization remains limited due to the involvement of numerous MPs.

To solve such limitation, we applied the Neural Compact Model (NCM) as a solution of rapid development of SPICE model while reflecting variation of device characteristics effectively. Recently, the compact model community has begun to pay attention to the development of NCM for high fitting accuracy of emerging devices [5-8] and fast fitting capabilities because of MPE being unnecessary. Wang and co-workers [5] adopted the conversion function at the output layer of the NCM to make drain current $I_D = 0$ at drain voltage $V_{DS} = 0$ and effectively train the significant order of magnitude difference between the sub- and above-threshold region. Tung et. al. [6] validated I_D characteristics of NCM up to 3rd-order derivatives and Jeong et. al. [7] simulated circuits based on the NCM using SPICE porting method through Verilog-A. Also, M. Cho and co-workers [8] developed the NCM by transfer learning to enhance accuracy and reduce training time for emerging organic field-effect transistors.

Although many recent studies have been developed NCM to be more physical and practical, several factors still make it challenging to apply in the industrial domain. First, the model retargeting to the selected electrical targets (ETs) has been rarely studied to construct the new model. Normally, when device characteristics change due to process modifications in the fab, representative ETs that reflect new device properties are used to retarget a new model instead of the full I-V and C-V curves to reduce TAT of the new model development. Although Wang et. al. [6] suggested ETs targeting by adjusting the last hidden layer of NCM, it was presented without sufficient validation of the model's consistency across the entire operating regime except for specific targets. Additionally, NCM still lacks design capacity compared to the existing design framework like equation-based compact models. Further research is necessary to address not only ET retargeting but also essential aspects of IC design such as binning model across design input variables, managing process variation, and developing layout-dependent models (LDEs).

In this work, we propose an efficient NCM self-retargeting strategy by transfer learning method with new input ETs. Furthermore, we introduce a direct data-driven SPICE

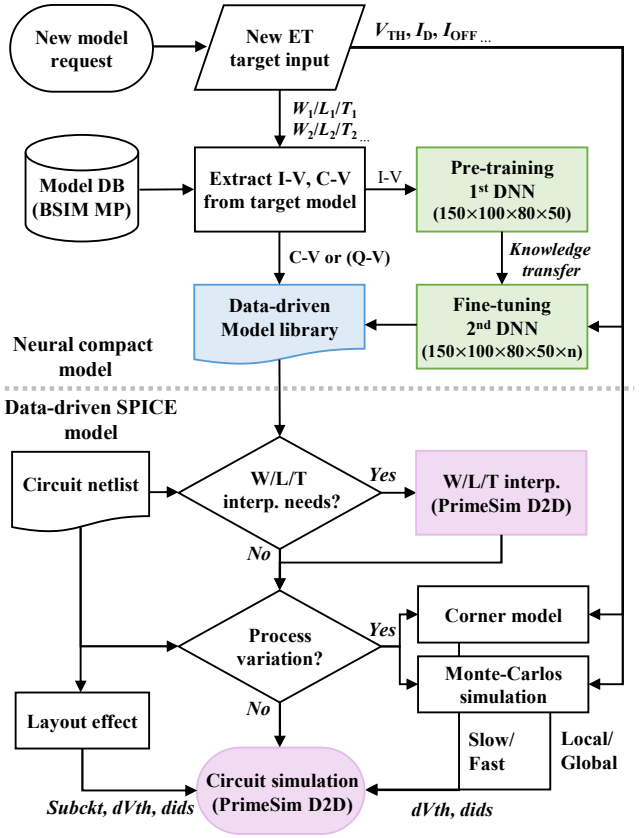


Figure 1. Workflow overview from the self-retargeting of the NCM to an enablement of the data-driven SPICE model.

simulation framework, DataSPICE, exploiting the commercial SPICE simulator PrimeSim [9]. This allows fast and accurate analysis for design-technology co-development by combining NCM and various design capabilities through data interpolation and derating techniques of the trained dataset.

II. Workflow Overview

Fig. 1 illustrates the workflow of the proposed NCM and data-driven model-based design framework presented in this paper. When device characteristics fluctuate due to technological changes, such as ion implantation or modifications of device structure, and a new SPICE model is required, process engineers provide new ETs such as threshold voltage (V_{TH}), I_D and off current (I_{OFF}). In traditional equation-based compact model frameworks, retargeting the model to new ETs involves adjusting hundreds of MPs from previously published models in database (DB). This process is time-consuming and can take several weeks, depending on the model.

We developed the self-retargeting method of the NCM to reduce TAT of the new model development. First, we pre-trained the I-V and C-V (or Q-V) datasets of the reference model extract from the DB using a Deep Neural Network (DNN). We then apply knowledge transfer to fine-tune the model for the new ETs. Finally, we can create a data-driven model library based on the retargeted I-V and C-V datasets. If there are only DC variations due to technology changes irrelevant to capacitance, the dataset of C-V are directly generated from the reference model, considering only the V_{TH} shift, without additional training. In this work, we consider this situation so that only I-V datasets are used for self-retargeting of NCM.

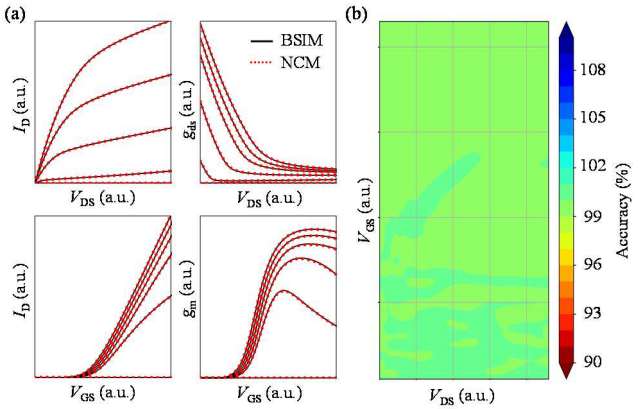


Figure 2. Pre-training result of NCM (red dotted line) versus the BSIM (black solid lines).

Once the data model library is ready, we perform direct data-driven SPICE simulation of the circuit netlist using the PrimeSim D2D [9]. To date, most NCMs have used Verilog-A to port SPICE and the drain current DNN models, but it poses difficulties in being compatible with conventional MPs-based design frameworks, such as binning, process variation and LDEs. By leveraging the fact that the inputs and outputs of the NCM are dataset, we employ data interpolation and derating techniques to integrate seamlessly into existing design environments. First, if design variables defined in the netlist such as width (W), length (L), and temperature (T) are not present in the input dataset, D2D performs interpolation internally with respect to $W/L/T$. Additionally, if the netlist requires process variation, we can adjust the dataset to slow or fast corners using data derating or generate normal distributions of V_{TH} and I_D to perform Monte-Carlo based statistical simulations. If consideration of LDEs is necessary, sub-circuit (subckt) and data derating enable to combine custom models that emulate LDEs.

III. Retargeting Strategy of NCM

The NCM consists of two training steps: The pre-training I-V characteristics of the reference model in the 1st DNN and the fine-tuning it to new ETs through the 2nd DNN using the transfer learning. We extract sequential dataset of I_D with respect to four input variables of NCM, gate (V_{GS}), drain (V_{DS}), body (V_{BS}) voltage and T. The bias range of V_{GS} and V_{DS} is selected to be sufficiently large to cover V_{DD} of target circuits, with steps of approximately 30 to 70 mV while V_{BS} is set with a wider interval of 100 to 300 mV. T is split into 3 different conditions, cold, room and hot temperature. Finally, we have a dataset consisting of over 30,000 I_D points per transistor and use it to train the 1st DNN with four hidden layer with different number of neurons (150, 100, 80, 50) and one output layer for I_D. We adopt conversion function $I_D = I_0 \cdot V_{DS} \cdot 10^y$ [6] and the mean absolute percentage error (MAPE) loss function of I_D, g_m, g_{ds} for training as follows:

$$Loss = a \cdot \mathrm{MAPE}(y) + b \cdot \mathrm{MAPE}(g_m) + c \cdot \mathrm{MAPE}(g_{ds}) \quad (1)$$

The 1st DNN for the pre-training uses optimal constants in loss function, $a = 1$ and $b = c = 0.1$ to get high consistency for all I_D, g_m and g_{ds}. The model includes the adaptive moment estimation ("Adam"), tanh as the activation function and trains 150,000 epochs to capture full dataset. To evaluate pre-training consistency of NCM, we examine a dataset of n-type planar field-effect transistor. Fig. 2 shows that characteristics

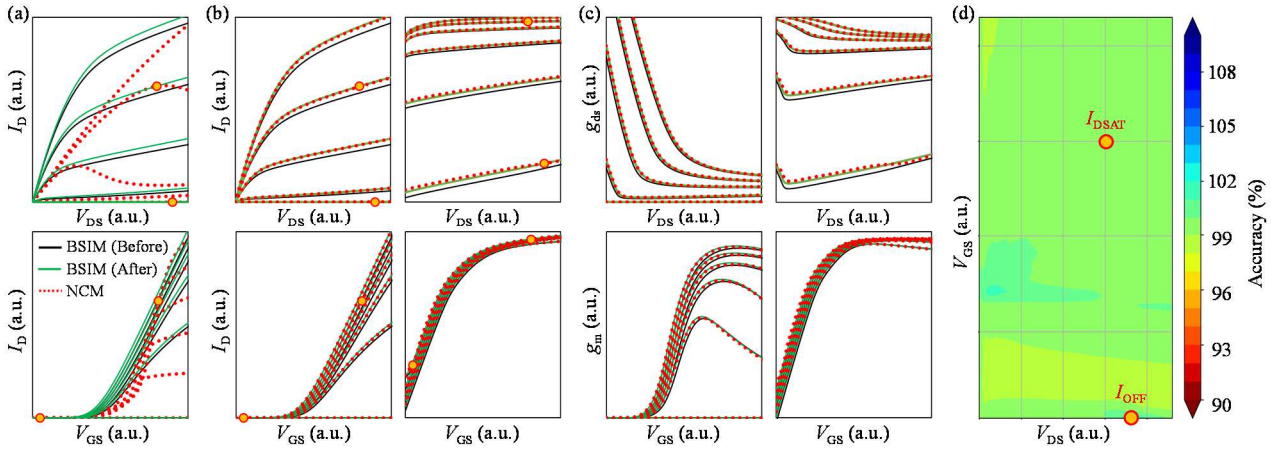


Figure 3. Fine-tuning result of NCM (red dotted lines), referenced BSIM for both before (black solid lines) and after (green solid lines) retargeting with 2 ETs case. (a) denotes result with $a = 1$, $b = c = 0$ in (1), (b-d) shows result with $a = 2$, $b = c = 1$ in (1) for I_D, g_m, g_{ds} and 2D plot of I_D regarding V_{DS} and V_{GS}.

of I_D, g_m and g_{ds} are fully captured by DNN. Particularly, 2D accuracy plot for I_D with respect to V_{GS} and V_{DS} is present, calculating the accuracy on a linear scale in the above-threshold region and on a logarithmic scale in the sub-threshold region using V_{TH} as the boundary. This validates that the consistency of the model is effective in the sub-threshold regime despite its significantly large order difference compared to the above-threshold region. In addition, the derivatives of I_D, g_m and g_{ds} are also precisely trained in the entire bias regime.

After pre-training, we construct 2nd DNN by knowledge transfer to retarget input ETs while retaining the pre-trained device properties in the 1st DNN. The 2nd DNN adds a new 5th hidden layer (150, 100, 80, 50, n) while weights and bias in first four hidden layers of 1st DNN are fixed. The number of neurons in the new layer, n, equals to number of input ETs to prevent the divergence of the model during fine-tuning. Here, we examine 2 input ETs case for I_{DSAT} (V_{DS}=V_{GS}=V_{DD}) and I_{OFF} (V_{DS}=V_{OFF}, V_{GS}=0 V), therefore n = 2. The 2nd DNN also adopts ADAM estimation, tanh activation function and MAPE as loss function similar to the 1st DNN. The key difference of the second DNN is that the dataset of ETs for retargeting contains normally fewer than 10 ETs, which is significantly smaller compared to the pre-training. Therefore, if the fine-tuning considers only this small ETs dataset, it is difficult to expect accuracy in regions outside the ETs due to overfitting (Fig. 3a). Although the I_D consistency at ETs is close to 100%, the curve profile is completely deviated from the reference model.

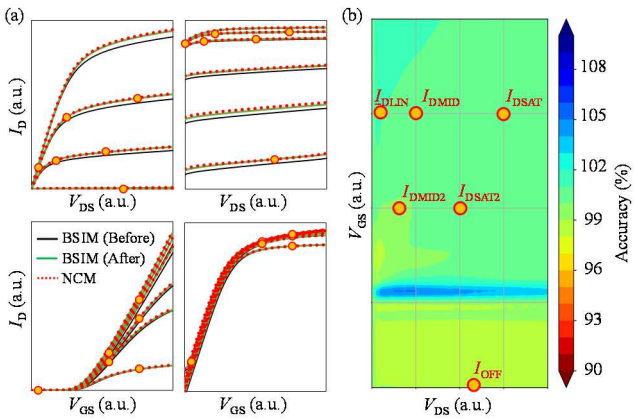


Figure 4. Fine-tuning result of NCM (red dotted lines), referenced BSIM for both before (black solid lines) and after (green solid lines) retargeting with 6 ETs case.

To solve the overfitting problem during self-retargeting, we modify the components in the loss function by considering derivatives of I_D, g_m and g_{ds}, to reflect the curve profile during the fine-tuning as well as ETs in (1). Here, the train dataset of g_m and g_{ds} are selected from the previous model before retargeting. By adjusting a, b and c in the loss function (1), this physic-informed like DNN can successfully predict a new model. We set $a = 2$ and $b = c = 1$ to give more weight on ETs than the curve profile of the previous model to hinder the model from ignoring input ETs and following the previous curve. In Fig. 3b and c, self-retargeted NCM show good accuracy in I_D and its derivatives in both linear and logarithmic scale with respect to the retargeted one based on the BSIM model. Even in 2D accuracy plot, the consistency of the self-retargeted NCM is almost perfect in all operating ranges.

In real cases, the number of ETs that need to be targeted by MPE varies depending on the model. For example, a single target is sufficient when circuits validation is enough at V_{DD}, but for circuits operating various regions, multiple ETs are necessary. In Fig.4, we demonstrates the accuracy of self-retargeted NCM based on 6 ETs input, I_{DSAT}, I_{DSAT2}, I_{DMID}, I_{DMID2}, I_{DLIN} and I_{OFF}. The fine-tuned I-V curves properly capture all 6 ETs including linear and off regions while containing the curve profile of the reference model. The average accuracy is over 99 % at 6 targets and over 95% at other regions, which means this retargeting algorithm is reliable even for more complex input cases.

Regarding TAT of model development, self-retargeting NCM strategy has advantages over the conventional framework. In our system with conventional equation-based model, TAT of new model development is nearly 100 hours for a model consisting of 10 W/L sizes configured with 6 ETs. In contrast, since the pre-training and fine-tuning of a transistor takes almost 0.5 hour based our computational condition in our NCM method, it can dramatically reduce TAT by 95%. In other words, we can achieve 20 times more activities of the design exploration and DTCO within same development period. Moreover, we can further reduce TAT if the computational capabilities improve.

IV. DATA-DRIVEN SPICE SIMULATION

A. *Instance interpolation*

To enhance the design capacity of NCM, we adopt a direct data-driven SPICE simulation, DataSPICE, via the commercial SPICE simulator, PrimeSim D2D [9]. It accepts

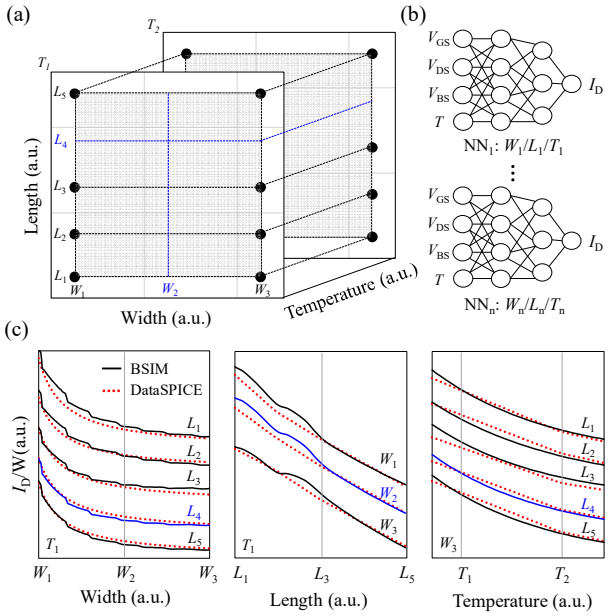


Figure 5. (a) *W*/*L*/*T* Interpolation schematic, (b) neural-network of NCM and (c) interpolation results of conventional SPICE with BSIM (solid lines) versus DataSPICE with NCM (dotted lines) for various design variables. Blue lines denotes result of variables not included in the input dataset.

I-V, C-V (or Q-V) dataset and enables SPICE simulation by designed interpolation and extrapolation schemes. This secures the simulation speed and robustness across all bias ranges. Also, D2D provides *W*/*L*/*T* binning of discontinuous dataset input by conducting scaling across design variables, like *W*/*L*/*T*, during dataset read-in stage of PrimeSim. In the conventional design methodology, few *W*/*L*/*T* sizes are enough to cover a wide range of the model coverage by seamless binning of key model parameters, such as *VTH0* and *U0* in BSIM [1]. In contrast, NCM should employ a large global model to guarantee a broad coverage. However, this approach not only incurs significant computational cost but also has limitations in accurately predicting characteristics over a wide range. Thus, we use an interpolation method to connect data from each small local model, ensuring comprehensive coverage and reducing both overhead and data size.

Fig. 5a is a schematic of the *W*/*L*/*T* interpolation of DataSPICE. First, when we receive a new model request, NCM-based self-retargeting method generates new datasets using the ETs for the primarily used sizes in the design, marked as black dots on the *W*/*L*/*T* plane. This process involves using one neural network (NN) for each unique *W*/*L*/*T*, such as NN_1: $W_1/L_1/T_1$ ⋯ NN_n: $W_n/L_n/T_n$ (Fig. 5b), and linearly interpolating these nodes over respective instance to implement model coverage (gray area in Fig. 5a).

Fig. 5c demonstrates the consistency of the interpolation. For various length inputs, L_1 to L_5, the interpolation over the width coverage range from W_1 to W_3 (red dotted lines) exhibits accurate predictions compared to conventional seamless binning-based BSIM models (black solid lines). The length interpolation for width inputs, W_1 to W_3, and the temperature interpolation at specific *W*/*L* conditions both shows high accuracy, within 3% of error rates with the reference. Notably, even for conditions such as W_2 and L_4, which were originally not selected as input sizes but instead interpolated (blue solid lines), the interpolation results accurately matches the reference data. This demonstrates that data interpolation can comprehensively ensure model coverage.

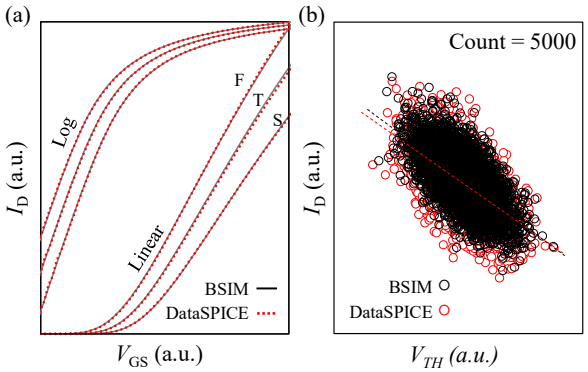


Figure 6. Two process variation result by conventional method (black) versus DataSPICE (red). (a) I_D-V_{GS} curve for slow and fast corner model in both linear and logarithmic scale and (b) V_{TH} – I_D correlation of 5000 times MC simulation.

B. *Process variation*

D2D also supports two representative process variation models, process corner model and Monte-Carlos (MC) based statistical simulation. In fact, the data-driven model had a drawback of not being compatible with process variations, which has limited its application. To overcome this limitation, we adopt data derating techniques, provides *dvth* and *dids* parameters that can directly adjust V_{TH} and I_D within given dataset. In other words, these parameters can make process corners or random distribution of the input dataset.

Fig. 6a illustrates linear and logarithmic scales of the generated slow and fast corners using the derating method of a typical dataset. When requesting a new model, slow and fast corner skew for V_{TH} and I_D is normally provided. We first horizontally shift the dataset's table using *dvth* to account for the V_{TH} skew, and then use *dids* to achieve the final targeting for any I_D skew. Because this method tunes V_{TH} and I_D simultaneously, the result demonstrates consistent simulation performance in both sub-threshold and above-threshold region which are comparable with the original BSIM-based corner model. Moreover, from the standpoint of data size of the data-driven model, it helps model lightweight by reducing dataset size, instead of offering all corners as datasets.

The full statistical simulation through the MC model is also possible using the same *dvth* and *dids* derating concept by the generation of normally distributed V_{TH} and I_D from the given dataset. In Fig. 6b, 5000 times of MC simulation of input typical dataset verifies the statistical modeling in DataSPICE has similar variation coverage with the conventional model. The V_{TH} versus I_D correlation between the data-driven MC and the original BSIM-based MC simulation shows almost identical distribution, and the trend lines of both data correlations nearly coincide. The R^2 value of the V_{TH} - V_{TH} correlation between the two MC models is 0.99, while the R^2 value of the I_D - I_D correlation stands at 0.91. Through data derating method in DataSPICE, NCM completely overcomes its incompatibility with the process variation simulation.

C. *Layout depedent effect*

Here, we investigated the application of the LDEs on the NCM. LDEs indicate variations in electrical performance of devices depending on the surrounding layout structure. The SPICE simulation receives structural information from input netlists, which modifies the properties of device parameters

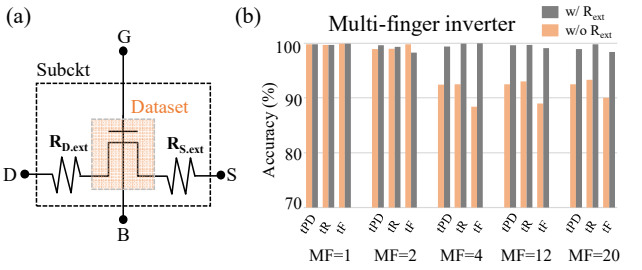


Figure 7. (a) Schematic of subckt with dataset and exteranal resistance R_{ext}. (b) Accuracy comparison of tPD, tR, tF for various MF inverters with R_{ext} (gray) and without R_{ext} (orange).

such as V_{TH}, I_D and contact resistance (R_C). Among various LDEs, we can consider effects like gate-to-contact distance, trench isolation space and length of diffusion, which are described through variations in V_{TH} and I_D [10], into the model using the previously explained *dvth*, *dids* derating method. However, the effect that alters R_C is still challenging to integrate.

A notable example is the multi-finger (MF) devices induced LDEs to the R_C. Generally, MF devices are employed to achieve high current capabilities and layout efficiency, but their sharing junctions can lead to unexpected variations in R_C. While conventional equation-based compact models like BSIM offer internal parameters for adjusting resistance and can also be implemented using subckt, NCM has been limited because a relevant design methodology has not yet been established.

To solve this limitation, we adopt the same subckt structure of the conventional model library. Our proposed DataSPICE with PrimeSim D2D supports the subckt structure while connecting I-V and C-V (or Q-V) datasets to each device node, allowing integration with custom models used in the subckt. As shown in Fig. 7a, we attached an R_{ext} within the subckt of dataset and integrated a custom model that describes R_C variations for MF devices. By increasing MF from 1 to 20 and examining the inverter's propagation delay time (*tPD*), rising time (*tR*), and falling time (*tF*), we found that the error exceeded 10% without R_{ext} as MF increased. However, we reduced the error to below 1% by implementing subckt and R_{ext} as an adjustable external resistance depending on the number of MF.

TABLE I. RELATIVE CIRUICT RUNTIME AND ACCURACY EVALUATION OF CONVENTIONAL SPICE WITH BSIM VERSUS DATASPICE WITH NCM

Circuits	P/V/T corner	Relative runtime	Accuracy index	Relative accuracy
29-stage RO	TT/V_{DD}/T_1	95.6 %	tPD	99.4 %
	SS*/V_{DD}/T_2	98.8 %		99.0 %
	TT/V_{DD2}**/T_3**	102.1 %		98.8 %
PB unit	TT/V_{DD}/T_1	94.9 %	Trip time	99.3 %
	SS*/V_{DD}/T_2	94.2 %		99.2 %
	TT/V_{DD2}**/T_3**	103.7 %		98.6 %

* Data derating for process variation

** Data interpolation for not input V/T variables

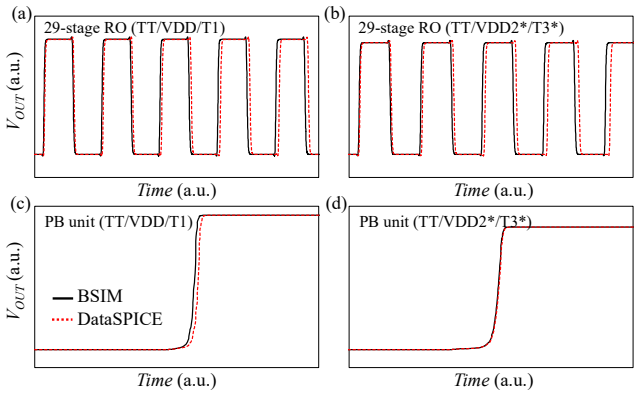


Figure 8. Circuit wavform of small-scale 29-stage RO with (a) TT/V_{DD}/T_1 and (b) TT/V_{DD2}/T_3 and mid-scale PB unit with (c) TT/V_{DD}/T_1 and (d) TT/V_{DD2}/T_3. Black solid lines denote conventional SPICE with BSIM and red dotted lines shows DataSPICE with NCM.

V. CIRCUIT VALIDATION

We compared simulation accuracy and runtime degradation of our NCM-based DataSPICE framework against traditional BSIM-based SPICE modeling using both small-scale and mid-scale circuits. Since the data-driven model is known to encounter challenges with SPICE runtime and convergence, particularly at large-scale circuits, we evaluated circuits of two different scales to assess comprehensively. For the small-scale circuits, we used 29-stage of ring oscillator (RO) to compare with tPD, while for the mid-scale circuits, we utilized the page buffer (PB) unit of the NAND Flash Memory to compare with trip time of specific latch operation. Here, transistors used in each circuit were approximately 200 and 15000, respectively. We split the simulation conditions into 3 different process/voltage/temperature (*P*/*V*/*T*) corners to appropriately validate both interpolation and process variation. TT and SS in Table I, each represent typical and slow process corners for both n-type and p-type field-effect transistors and SS corner is generated by data derating method. Also, V_{DD}, T_1, and T_2 are all conditions used as data inputs, whereas V_{DD2} and T_3 are not, necessitating additional interpolation with respect to *V* and *T*, respectively, to perform SPICE simulations.

Table. I presents the relative performance metrics of SPICE runtime and accuracy of the NCM-based DataSPICE with respect to the BSIM-based conventional SPICE simulation for each *P*/*V*/*T* corner. Firstly, small scale of the 29-stage RO showed a runtime reduction of 95.6% and 98.8% as consistency of tPD exhibited over 99% under the two conditions where interpolation was unnecessary, TT/V_{DD}/T_1 and SS/V_{DD}/T_2. In the case of TT/V_{DD2}/T_3, however, the runtime increased slightly by 102.1% due to the interpolation process while the consistency was validated at a similar level of 98.8%. For the larger-scale PB unit, the runtime also decreased to about 94% in the first two conditions without interpolation, and the consistency of trip time remained over 99%. Similar to the RO, the TT/V_{DD2}/T_3 condition requiring interpolation showed an increase in runtime to 103.7%, with a slight decrease in consistency to 98.6%. From this result, we demonstrated that the data-driven model can perform circuit simulations without convergence issues even for mid-level scales with over 15k transistors. Notably, when interpolation was not used, the runtime decreased with remaining the original level of circuit's consistency. Demonstrating the effectiveness of our design framework, this methodology achieves notable accuracy and simulation runtime as well as

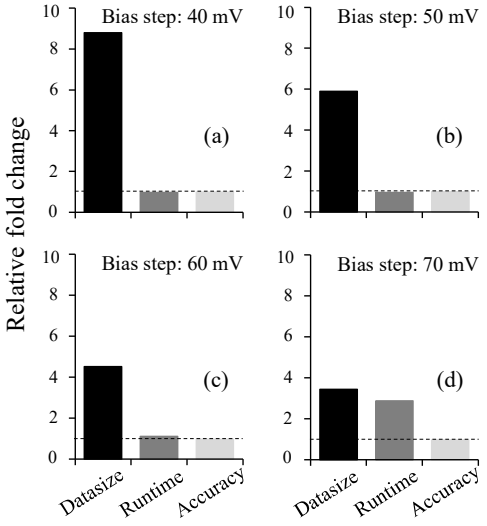


Figure 9. Relative fold change of datasize, circuit runtime and accuracy of NCM-based DataSPICE compared to conventional SPICE simulation for V_{DS} and V_{GS} step is (a) 40 mV, (b) 50 mV, (c) 60 mV and (d) 70 mV .

the robust circuit waveform (Fig. 8) comparable to the conventional SPICE simulation.

Since runtime and accuracy are inherently size-dependent in this model, selecting optimal bias steps of NCM is important. A finer dataset step reduces the number of interpolations, improving runtime and accuracy at the cost of increased data size, whereas a sparser step reduces data size but increases interpolation, causing both runtime and accuracy to deteriorate. To prove this, we constructed the model by dividing V_{DS} and V_{GS} steps in the dataset, categorized as 40, 50, 60, and 70 mV, and evaluated the PB unit with TT/V_{DD}/T_1 conditions. Fig. 9 shows how the data driven model's data size, circuit runtime, and accuracy changed by several folds compared to the BSIM-based conventional simulation. The data model with the finest bias step of 40mV had a data-size increase of 8.8 times, but it gradually decreased as the bias step increased, reaching 3.3 times at 70mV. The runtime was similar to the conventional model up to 60 mV, but it increased by a factor of 2.8 for 70 m V because the wider bias interval results in an increase of interpolation. Also, we confirmed that the accuracy of the simulation is consistent with conventional method in all conditions regardless of data size or runtime. Therefore, it is crucial to carefully consider the trade-off between data size and runtime when determining the step size of NCM.

VI. Conclusion

In this work, we proposed a NCM that enables efficiently retargets to new input ETs and its data-driven design infrastructure to guarantee equivalent design environment with the existing framework. In particular, the retargeting of device model using the transfer learning of NCM showed high consistency of I_D prediction over 95% throughout the entire operating range while constructing model 20 times faster than the conventional equation-based model. This will significantly reduce the communication delay between technology and design, allowing for more design exploration and DTCO activities within a limited timeframe and contributing to product completeness. In addition, we introduced a direct data-driven SPICE simulation, DataSPICE, to provide I-V and C-V modeling of NCM, as well as its *W*/*L*/*T* interpolation, process variation, and LDEs modeling. This expands the design capability of NCM and facilitates its application in practical IC chip design. We validated this framework at the mid-level circuit scale, demonstrating its utility, and enabling a rapid evaluation of DTCO's impact at a larger scale beyond the cell-level analysis. Additionally, this approach supports not only NCM but also TCAD and measurement-based datasets, facilitating swift and flexible collaboration between technology and design.

References

[1] J. P. Duarte, S. Khandelwal, A. Medury, C. Hu, P. Kushwaha, H. Agarwal, A. Dasgupta, and Y. S. Chauhan, "BSIM-CMG: Standard FinFET compact model for advanced circuit design," in Proc. 41st Eur. Solid-State Circuits Conf. (ESSCIRC), Sep. 2015, pp. 196–201

[2] Yiming LI and Y-Y. Cho, Intelligent BSIM4 Model Parameter Extraction for Sub-100nm MOSFET Era, Jpn. J. Appl. Phys., 43, 1717 (2004)

[3] F. Chavez, C-T. Tung, M-Y. Kao, C. Hu, J-H. Chen and S. Khandelwal, Deep learning-based I-V Global Parameter Extraction for BSIM-CMG, Solid State Electronics, 209, 108766 (2023)

[4] Z. Peng, Y. Wang, Z. Yuan and X. Wang, iPREFER: An Intelligent Parameter Extractor based on Features for BSIM-CMG Models, arXiv https://arxiv.org/abs/2404.07827 (2024)

[5] J. Wang, Y-H. Kim, J. Ryu, C. Jeong, W. Choi and D. Kim, Artificial Neural Network-Based Compact Modeling Methodology for Advanced Transistors, IEEE Trans. Electron Devices, 68, 3 (2021)

[6] C-T. Tung, M-Y. Kao and C. Hu, Neural Network-Based I–V and C–V Modeling With High Accuracy and Potential Model Speed, IEEE Trans. Electron Devices, 69, 11 (2022)

[7] H. Jeong, S. Woo, J. Choi, H. Cho, Y. Kim, J-T. Kong and S.Kim, Fast and Expandable ANN-Based Compact Model and Parameter Extraction for Emerging Transistors, IEEE J. Electron Devices Soc., 11, 153 (2023)

[8] M. Cho, M. Franot, O-J. Lee and S. Jung, A neural compact model based on transfer learning for organic FETs with Gaussian disorder, J. Mater. Chem. C, 12, 16691 (2024)

[9] PrimeSim User Guide: Pro and SPICE, Synopsys, CA, USA (2023)

[10] S. Kim, S. Yang, H. Lim, H. Lee, J. Jeon, J. Choi and J. Kim, Accurate Layout-Dependent Effect Model in 10 nm-Class DRAM Process Using Area-Efficient Array Test Circuits, IEEE Access, 11, 70691 (2023)

Unlocking a New Rust Programming Experience: Fast and Slow Thinking with LLMs to Conquer Undefined Behaviors

Renshuang Jiang[†¶], Pan Dong[†¶], Zhenling Duan[†], Yu Shi[†], Xiaoxiang Fang[†], Yan Ding[†],
Jun Ma[†], Shuai Zhao[‡], Zhe Jiang[§*]
[†]National University of Defense Technology, China. [‡]Sun Yat-sen University, China. [§]Southeast University, China.

***Abstract*—To provide flexibility and low-level interaction capabilities, the "unsafe" tag in Rust is essential, but undermines memory safety and introduces Undefined Behaviors (UBs) that reduce safety. Eliminating UBs requires a deep understanding of Rust's safety rules and strong typing. Traditional methods require depth analysis of code, which is laborious and depends on knowledge design. The powerful semantic understanding capabilities of LLM offer new opportunities to solve this problem. Although existing large model debugging frameworks excel in semantic tasks, limited by fixed processes and lack adaptive and dynamic adjustment capabilities. Inspired by the dual process theory of decision-making ("Fast and Slow Thinking"), we present a LLM-based framework called RustBrain that automatically and flexibly minimizes UBs in Rust projects. Fast thinking extracts features to generate solutions, while slow thinking decomposes, verifies, and generalizes them abstractly. To apply verification and generalization results to solution generation, enabling dynamic adjustments and precise outputs, RustBrain integrates two thinking through a feedback mechanism. Experimental results on Miri dataset show a 94.3% *pass* rate and 80.4% *execution* rate, improving flexibility and Rust projects safety.**

I. Introduction

Rust, a systems programming language, is popular for its guarantees of memory safety and concurrency [1], [2]. For flexibility, Rust has a secondary language hidden inside it which does not enforce memory safety guarantees: it's called Unsafe Rust and works just like regular Rust (it's called Safe Rust). Safe Rust ensures safety through strict compiler checks. Unsafe Rust uses *unsafe* keyword to grant direct control over memory and system resources, bypassing certain compiler checks, with safety guarantees that depend on the developer. Therefore, this capability introduces the potential for Undefined Behaviors (UBs), a class of run-time bugs that are difficult to detect and can lead to severe security vulnerabilities and instability [3], [4]. Since Unsafe Rust cannot be avoided [5], especially in system software, how to perform fixes is a key in ensuring Rust's safety.

The main methods for fixing UBs in Unsafe Rust include safe substitution, inserting assertions, and semantic modification [6]–[9]. Rust's strongly typed system, along with its strict ownership, lifetime, and borrowing rules, makes this process challenging, requiring an in-depth understanding of complex code semantics [40]. Although the types of Unsafe Rust operations are finite (five types of unsafe operation [1], [5]), minor semantic variation significantly affect how code modification is approached [3], [9], leading to situations where similar UB may not yield comparable solutions. Therefore, understanding semantics and possessing specialized knowledge are essential to addressing Rust safety issues. Traditional methods [33]–[37] require engineers to apply specialized knowledge along with strict logical reasoning, judgment, and verification processes, where the efficiency of resolution is closely tied to experience and expertise. The powerful self-reasoning and semantic understanding capabilities of large language models (LLMs) offer new opportunities to enhance the safety of Rust programs [10]–[13]. Due to high training costs and a lack of high-quality data in specific domains, conventional LLMs struggle with deep understanding and complex relational reasoning when handling specialized languages like Rust. In contrast, existing large model repair frameworks [12], [13], while providing assistance for such problems, typically follow pre-designed and fixed repair processes, lacking the professional capabilities to tailor solutions based on the specific characteristics of the code.

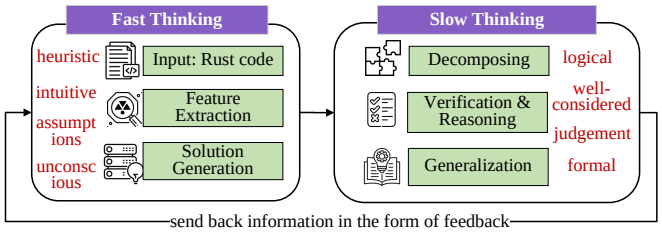


Fig. 1: RustBrain: Fast and Slow thinking LLM processes.

Inspired by human cognitive psychology, *"Thinking Fast and Slow"* [14], we observe that LLMs provide a broad, generalized understanding that resembles the "fast thinking" of humans. This thinking is rapid and intuitive, but it is also error-prone reasoning. Yet, unlike human cognition, LLMs lack a "slow thinking" mechanism, employing specialized knowledge and logical reasoning to enhance accuracy, identify discrepancies, and mitigate errors. This is crucial when addressing complex problems that demand deliberate and systematic approaches to detect inaccuracies and ensure correctness. These two complementary processes – fast, intuitive reasoning and slow, analytical reasoning – form a holistic framework for understanding and solving complex problems, e.g., solving UBs in Rust.

In light of this, we introduce RustBrain, a framework that integrates these cognitive processes, leveraging their respective strengths to autonomously and flexibly minimize UBs. As depicted in Fig. 1, in the fast-thinking process, it extracts code information, identifies error types, and analyzes them using a broad knowledge base to generate multiple repair solutions. In the slow-thinking process, the framework in-depth analyzes and validates the repair solutions based on the four design principles: decomposition, verification, reasoning and generalization. First, the solutions are decomposed and executed using multiple agents. In this stage, based on observations of Rust, we designed several agents,including three types of error-fixing agents, adaptive rollback agent, and abstract reasoning agent, to support better decomposition and verification. Then, the solution is fine-tuned based on verification results in slow-thinking, further optimizing the solution. Finally, to enhance the framework's generalization capability and better support precise solutions for similar errors, RustBrain establishes an evaluation and self-learning mechanism. This feeds the execution and evaluation results from the slow-thinking stage back into the fast-thinking, continuously optimizing solution generation and refining decision-making.

To evaluate RustBrain, we utilize a dataset containing UBs provided by Miri repository [17]. The results demonstrate that it achieves 94% *pass* rate and 80% execution semantic acceptability rate. Compared to the state-of-the-art Rust fix tool (RustAssistant [13]), it *pass* rate has increased by 30% . Additionally, compared to engineer experts, the speed evaluation shows an improvement of up to 18x.

¶ Both authors contributed equally to this research.
* represents corresponding authors. Emails: zhejiang.uk@gmail.com.

II. RustBrain: An Overview

A. Scope

Our goal is to build a toolchain that flexibly and systematically evaluates the capabilities of RustBrain for eliminating UBs in Rust at compile time. Repaired code must pass the Miri compiler while preserving its expected semantics. Miri [17] is a UB detection tool for Rust. Since many Rust code repairs are closely tied to context, the acceptability of semantics before and after the repair may vary significantly. Therefore, this paper validates semantics using test benchmarks composed of developer-repaired code. Compared to direct testing through Miri, this semantic judgment approach may introduce a degree of subjectivity. This is due to factors such as the size of the benchmark (i.e., the probability of covering all possible semantics) and other variables. Besides, we use Miri to detect Rust UBs and only considers Rust source files (i.e, files with .rs extension). Errors not detected by Miri are beyond the scope of this paper. In addition, RustBrain uses LLMs to guide the reduction of UBs, and we temporarily disregard Rust code that exceeds LLM tokens limits.

B. RustBrain Overview

According to Kahneman's theory [14], human decision-making relies on two main processes: fast thinking, which handles intuitive, quick, often unconscious decisions, and slow thinking, which tackles complex situations through logical reasoning and careful analysis for well-considered choices. These are suitable for solving highly specialized problems that require comprehensive analysis and precise judgment. We observe that unsafe operations in Rust are limited and rely on intuitive, pattern-driven processes, similar to fast thinking. However, Rust's strong type system requires specialized verification and improvements, similar to slow thinking. Therefore, RustBrain is designed as a dual-phase thinking framework incorporating fast and slow thinking (see Fig. 2). Fast thinking is intuitive and rapidly processes tasks, generating potential repair solutions based on code features which include error identification, code feature extraction. Slow thinking provides deliberate and in-depth processing. It combines decomposition, validation, reasoning and generalization to better understand the essence of the problem, delivering more targeted and comprehensive solutions for these types of UBs. By leveraging these two types of thinking, the framework achieves adaptive and high-precision repairs, improving both efficiency and reliability in eliminating UBs. RustBrain framework involves the following stages:

- **Stage F1 and F2:** During fast thinking process, Rust code is used as input, and Miri is utilized to detect UBs. If no UBs are found, the process terminates, outputting "pass". Conversely, relevant errors, code, and design hints are sent to a LLM agent for feature extraction. Subsequently, these features are integrated with another LLM to rapidly generate multiple repair solutions.
- **Stage S1:** These solutions are decomposed, validated, reasoned and generalized during the slow thinking process. Each solution is broken down into refined steps to enable distributed processing and resolution by different agents. The order of these steps reflects diverse repair strategies, enabling flexible process combinations to optimize repair paths and outcomes.
- **Stage S2:** This stage integrates multiple agents to verify solutions, including agents designed in previous work [27], [41], and new agents specifically developed based on Rust's unique characteristics. These agents include: (i) safe-replacement, assertion-adding, and code-modification agents fix UBs to improve code safety. (ii) adaptive rollback agent to reduce hallucinations and propagation in LLMs by reverting to the optimal code state; (iii) abstract reasoning agent supports deep semantic understanding and provides more professional fixes. Additionally, the evaluation mechanism is designed to conduct multidimensional assessments of solutions and further refine them through reasoning. For instance, increasing the iteration count, adjusting parameters in the repair steps, optimizing the execution path, etc.
- **Stage S3:** To provide precise solutions for similar errors, we design a feedback mechanism to match error features with optimized solutions, combining in-depth evaluation of slow thinking with generation of fast thinking. This self-learning method integrates both stages' strengths, reduces reliance on knowledge, and enables dynamic solution optimization.

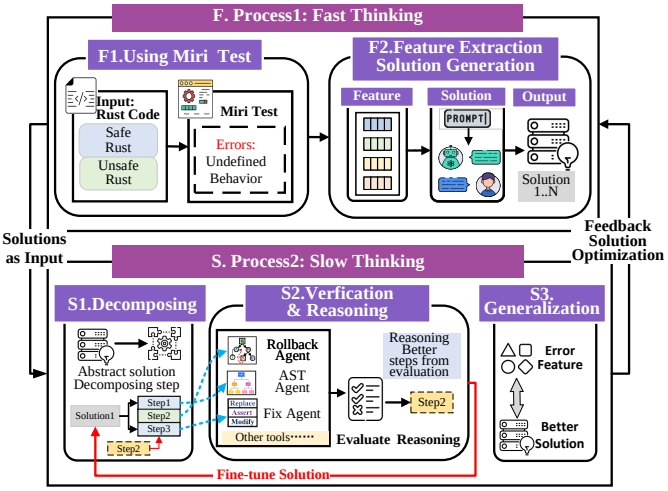


Fig. 2: RustBrain Overview: it leverages Miri for static analysis (F1). Extracts and combines features for solution generation (F2). Abstraction and decomposition into steps (S1), integrates multi-agent verification to reduce UBs. Evaluate and reason to fine-tune solution (S2). Generalization provides a solution for such UBs(S3).

Overall, RustBrain integrates fast and slow thinking to eliminate Rust-UBs, overcoming the limitations of traditional repair frameworks. It also combines decomposition, verification, reasoning, and generalization to achieve deep semantic understanding and expert-level support, providing more precise solutions for similar errors while reducing reliance on knowledge bases.

III. RustBrain Implementation

This section explores the technical implementation of the "Fast and Slow Thinking" framework. First, we classify five types of unsafe Rust operations and design the fast thinking process for generating solutions (Section III-A). Then, we present the slow thinking process to decompose solutions and execute them with multi-agents: agents for errors fixing with LLMs (Section III-B1); agents of rollback mechanisms and optimal code selection (Section III-B2) to mitigate hallucinations; and abstract reasoning agents leveraging Rust semantics for expert guidance(Section III-B3). Finally, the results from the slow-thinking are feedback into the fast-thinking to refine evaluations and generate more specialized solutions (Section III-C).

A. Classification of Unsafe Rust and Fast Thinking Process

To match the error types as closely as possible with the solutions generated by the fast thinking process, we further classify Unsafe Rust from the perspective of error repair. This classification allows for targeted problem-solving by identifying the unique characteristics and challenges of each type of unsafe operation.

*1) **Classification of Unsafe Rust**:* Rust's unique design ensures that programs written in Safe Rust have no memory safety issues [5], [22]. However, Unsafe Rust allows programmers to bypass these

safety checks and perform certain operations. Unlike C, where unsafe operations are everywhere and are barely checked by the compiler [23], we observe that Rust explicitly classifies unsafe operations and limits them to "unsafe" code blocks. Even errors arising from interactions between unsafe and safe code occur within the "unsafe" block. This design significantly reduces the surface area for potential memory safety bugs, thereby facilitating error localization.

Unsafe Rust can be divided into five categories [1]: dereference raw pointers, call unsafe functions, implement unsafe traits, access/modify mutable static variables, and access union fields. Based on experience and research [4], [9], we further categorize Unsafe Rust from the perspective of error fixing as follows: **(i) Operations with Safe Alternatives:** Certain unsafe operations can be replaced with safer APIs from Rust's standard library. **(ii) Operations Checking via Assertion:** Adding assertions helps detect boundary and type violations, enhancing safety without fundamentally altering the program's structure. **(iii) Operations Semantic Modification:** Understanding the underlying intentions of the original code, modifying erroneous semantics, and ensuring that these intentions are preserved in the modifications. Given Rust's strong type system, we observe that similar unsafe operations often require distinct handling methods. For instance, in case (i), as shown in Fig 3, the same unsafe API may require different replacement APIs depending on the context.

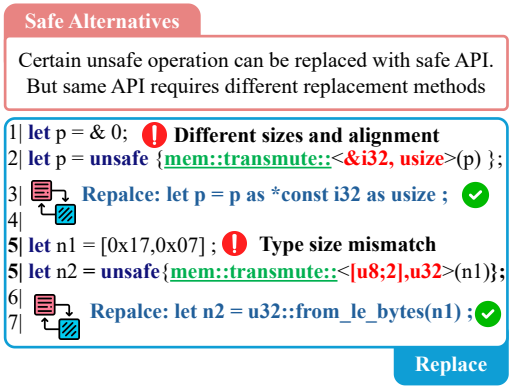


Fig. 3: The same API requires different substitution methods.

Therefore, based on this classification, the following principles are proposed in this paper to guide the framework design:

- **Principle 1:** All Unsafe Rust operations are explicitly marked with the "unsafe" keyword, reducing the surface area for potential memory safety vulnerabilities facilitates error localization.
- **Principle 2:** The limited types of UBs caused by Unsafe Rust can be further classified into three categories: replaced by Safe Rust, prevented by assertions and require semantic modification.
- **Principle 3:** Under Rust's strong typing characteristics, a single piece of code may exhibit multiple patterns, necessitating that changes be combined with semantic analysis.

*2) **Fast Thinking Process**:* Undefined behaviors usually occur at runtime. We believe that *prevention is better than cure*, therefore RustBrain integrates tools such as Miri compiler to analyze the Rust code. This static analysis occurs at compile time, where the compiler identifies potential UBs within the code, and records them for further examination. Once the detection phase is complete, the framework enters the fast thinking process.

In this process, RustBrain quickly identifies and generates fix solutions based on intuitive Unsafe Rust error pattern recognition. Firstly, RustBrain, based on five types of unsafe operations, employs a general-knowledge LLM to rapidly analyze code features, leveraging its generalization capability to identify typical error patterns and guide solution generation. According to **Principle 1**, UBs are generally confined to unsafe regions, which narrows the scope of code analysis and reduces the noise impact of irrelevant code on feature extraction. Then, since Unsafe Rust operations are limited in types, even with strong typing considerations, similar repair solutions can typically be found for most issues. Based on this, RustBrain generates multiple repair solutions tailored to different code characteristics by integrating **Principle 2**, ensuring that the repair methods are both adaptable and diverse in addressing the unique traits of the code.

B. Slow Thinking Process

After fast thinking generates preliminary solutions, RustBrain transitions into slow thinking for deeper thinking. Due to **Principle 3**, we need to further decompose, validate, infer, and generalize solutions. First, RustBrain splits solutions into multiple steps, analyzing and addressing UBs layer by layer. Then, it integrates agents design to validate and evaluate each generated solution's compatibility with type and safety requirements. We introduce new agents based on Rust features to support deep and professional fixes.

*1) **Three Types of Error Fixing Agents**:* To better guide LLM in fixing UBs errors, inspired by **Principle 2**, RustBrain constructs three types of special agents: equivalent replacement, assertion and modification. We observe that unsafe Rust is often misused, with some cases where safe code could achieve the same functionality [3], [4], [9]. Therefore, RustBrain designs an agent to identify situations where safe alternatives can replace unsafe operations. This minimizes unsafe code usage, enhancing the overall safety and reliability of Rust. Besides, we design an assertion agent that adds assertions in appropriate places to capture potential UBs at runtime before they can propagate. This avoids altering the code's semantics while detecting and stopping potential UBs during execution in a timely manner, preventing it from leading to more severe security vulnerabilities. However, in certain complex scenarios, UBs involve deeper logical errors or dependencies on external states, and assertions alone cannot directly resolve these issues. To address this, RustBrain introduces a modification agent, where adjust the erroneous semantics in the Rust code, ensuring that the code's functionality remains unchanged while effectively reducing the risk of UBs. Prompts are shown in the Fig. 4.

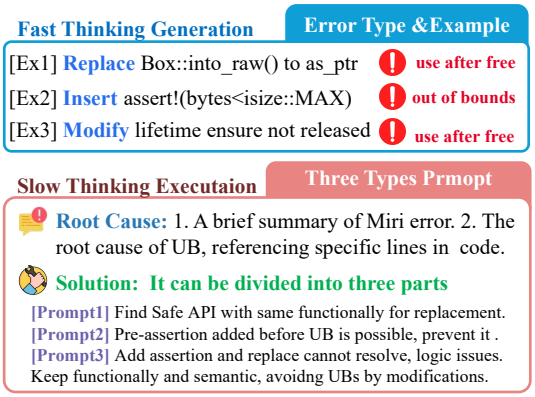


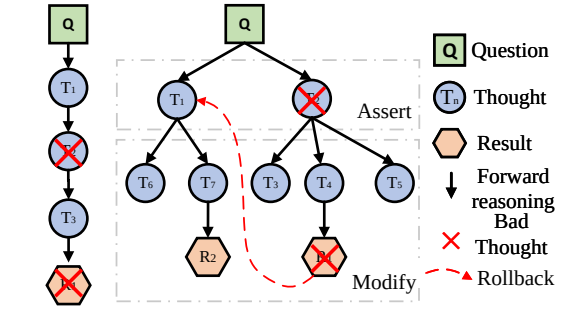
Fig. 4: Three types of prompt strategies.

*2) **Adaptive Rollback and Optimal Code Selection Agent**:* Using multiple fix agents can significantly reduce UBs in Rust code (Section IV). However, we observe that in some cases, the number of errors increases after repair, and analyzed and discussed this.

We define two sequences: $T = \{T_0, T_1, \ldots, T_p\}$ for thoughts generated by slow thinking, and $N = \{n_0, n_1, \ldots, n_p\}$ for errors detected by Miri, where T_i represents thoughts, n_i the errors count, and p the iteration count. During the slow thinking, the error sequence N may continue to grow, as in the example sequence $N_1 = \{1, 3, 4, 6, 9\}$. This phenomenon is known as "large model

hallucination" [24], [25]. As shown in Fig. 5 (a), without rollback mechanism, the process would proceed from the current error stage to the next stage, exacerbating the impact of hallucinations. Existing debugging frameworks [26], [27] typically rollback directly to the initial state T_0 to eliminate accumulated errors and mitigate the propagation of incorrect fixes. However, this discards valuable partial corrections made during the iteration process and creates a large overhead ($c * T_n$, where c is the number of rollbacks) [26].

We observe another type of error sequence, such as $N_2 = \{3, 1, 5, 2, 0\}$. This indicates that while errors may increase in certain iterations, the overall trend shows a fluctuating decline. This behavior is similar to humans often providing frequent and random errors during reasoning, but eventually decrease them and converge to a solution. Therefore, we suggest that slow thinking a certain degree of self-correcting capability, allowing it to gradually converge towards the correct through multiple iterations.



(a) No rollback mechanism (b) Two stage adaptive rollback mechanism

Fig. 5: No Rollback Mechanism and Adaptive Rollback Mechanism.

Based on the analysis, RustBrain introduces an adaptive rollback mechanism that selects intermediate states intelligently, enabling a more precise and efficient correction process. As shown in Fig. 5 (b), before proceeding to the next stage, the process rollback to the optimal code state (the fewest detected errors) to prevent error propagation. This ensures that subsequent iterations are based on the most refined and stable version of the code, minimizing error propagation and improving overall fix accuracy. The rollback overhead is expressed as: $c * T_{n-a}$ (where a is the number of thoughts produced in the assertion). Compared to direct rollback, RustBrain significantly retains the LLM's reasoning capabilities while reducing the overhead generated by reasoning.

*3) **Abstract Reasoning Agent**:* To better understand Rust's unique programming semantics and conventions, we propose incorporating a abstract reasoning knowledge agent to facilitate complex relational reasoning and deeper insights.

Traditional debugging frameworks typically rely on source code snippet libraries, syntax and API specifications, as well as error patterns and repair strategy libraries [9], [28], [29] to design knowledge bases. However, these methods are not suitable for Rust. As explained in **Principle 3**, Rust's strong typing characteristics mean that a single unsafe API may exhibit multiple patterns simultaneously, requiring knowledge bases with precise error pattern comprehension and adaptable fixes. Existing designs typically rely on pattern matching and learning, which makes it difficult to cover all potential error scenarios [9]. Additionally, the complex syntax and nested structures in Rust can introduce a large amount of invalid or misleading noise during the construction of the knowledge base, which interferes with the judgment and generation of LLM and limits the effectiveness of these methods. Therefore, RustBrain introduces a knowledge base centered on AST (Abstract Syntax Tree) and semantic analysis.

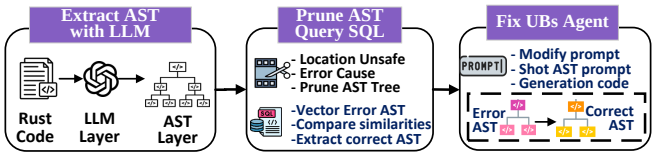


Fig. 6: Abstract Reasoning Agent.

As shown in Fig. 6, RustBrain extracts the code's AST using LLM instead of traditional Rust parsing tools like *syn*. The AST constructed by *syn* is overly complex and lacks higher-level semantic understanding. LLM not only preserves the semantic context of the code but also flexibly handles the complex contextual relationships. However, the AST extracted contains irrelevant or noisy information, which distracts the model's attention and decreasing fix performance. According to **Principle 1**, all Unsafe Rust operations are marked with "unsafe" keyword. RustBrain designs a pruning algorithm (as shown in Algo.1) to retain safety-relevant and functional fix components, minimizing the interference of irrelevant code for the LLM, thus improving the efficiency and accuracy of the repair process.

Algorithm 1: Pruning irrelevant nodes for Rust AST.

Input: Original Rust AST: $\mathcal{A}_{orig}$; Miri errors: $\mathcal{E}_{Miri}$;
Output: Pruned AST: $\mathcal{A}_{pruned}$;

Initialize pruned AST $\mathcal{A}_{pruned} \leftarrow \emptyset$; **foreach** *node in* $\mathcal{A}_{orig}$ **do**
  if *node contains* unsafe *keyword* **then**
    Mark the node as unsafe operation; Add node to $\mathcal{A}_{pruned}$;
foreach *unsafe node in* $\mathcal{A}_{pruned}$ **do**
  if $\mathcal{E}_{Miri}$ *!= 0* **then**
    Analyze context of error node;
    if *context node is irrelevant to unsafe operation* **then**
      delete context node to $\mathcal{A}_{pruned}$;
return $\mathcal{A}_{pruned}$;

Finally, RustBrain vectorizes the pruned AST and uses similarity-based search to query the knowledge base, retrieving repair solutions for error-prone AST structures. These solutions are added to prompts to guide the LLM in generating precise fixes for semantically similar code, which improves repair accuracy (Section IV).

C. Feedback Mechanism in Fast and Slow Thinking

Fast and slow thinking improves the flexibility and accuracy in eliminating Rust UBs. However, the quality of the results from slow thinking are not evaluated against fast thinking solutions. It limits dynamic adjustment of repair solution based on discovered issues, leading to the persistence of incorrect or inadequate repair solutions and affecting the overall effectiveness of the repair process.

RustBrain designs a feedback mechanism that integrates validation results into solution generation, allowing further adjustments to initial solutions. Specifically, we establish a triplet: *(accuracy, acceptability, overhead)*, to evaluate solutions from multiple dimensions. *Accuracy* ensures the solution passes Miri; *acceptability* checks semantics consistency; and *overhead* measures execution time and resource use. Through multi-dimensional evaluation and feedback mechanism, fast-thinking is able to absorb the precise analysis and validation information from slow-thinking, allowing the generated repair solutions to achieve a better balance between speed and accuracy, effectively enhancing the overall quality and reliability of the repairs. Moreover, this self-learning process enhances RustBrain's generalization ability,

supports similar errors repairs, reduces dependency on the knowledge base, and avoids the limitations posed by highly specialized issues.

IV. Evaluation

The evaluation of RustBrain designed to answer the following research questions:

1) **RQ1 (Flexibility):** How does RustBrain flexibility affect repair *pass* rate and performance?
2) **RQ2 (Accuracy):** What extent is RustBrain successful in fixing Rust UBs errors as measured by metrics of *pass* and *exec*?
3) **RQ3 (Sensitivity):** How does LLM configurations in GPT-4 impact the performance of RustBrain?
4) **RQ4 (Advancement):** How does the effectiveness and performance of RustBrain in eliminating UBs compared to the state-of-the-art method and human expert?

Firstly, we give a brief explanation of the datasets, large language models, baselines and metrics used in this paper.

Datasets. We use a dataset collected from the official Miri [17], containing code with UBs. The dataset is categorized into different types, including alloc, dangling pointers, and concurrency issues.

LLMs. We use four LLMs: GPT-3.5 [18], GPT-4 [19], GPT-O1 [20], and Claude 3.5 [21], each with a default temperature of 0.5.

Baselines. Apart from the Miri test, the comparison includes state-of-the-art approaches such as human expert repair (Thetis-Lathe, 2023) [3], and a LLM-based fix method (RustAssistant, 2024) [13].

Metrics. All experiments report *pass* rate(%) and *execution* rate(%), among which *pass* rate refers to the percentage of code passing Miri after repair, and *execution* rate refers to the percentage of code passing Miri with acceptable semantics.

Subsequently, we analyze the results to answer these questions.

RQ1 (Flexibility): We select a type of UB (requiring semantic modification) and input it into RustBrain. Ten solutions are generated through fast thinking, followed by reasoning and validation using slow thinking, as depicted in Fig. 7.

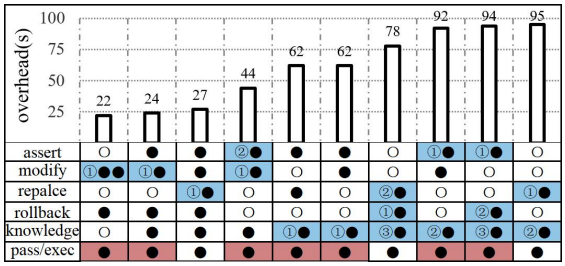


Fig. 7: RustBrain flexibly fixes UBs. Agents, shown on the left, can be enabled [●] or disabled [○], with serial numbers indicating repair order. Blue denotes agents used in execution, and red indicates semantically acceptable code, otherwise it only passes Miri.

RustBrain provides more flexible solution and improves the possible and accuracy of fixing UBs. The results indicate that: **(i)** For same problem, RustBrain uses fast thinking to generate diverse solutions, overcoming the limitations of fixed-process frameworks that offer only one option, enhancing flexibility and repair effectiveness. **(ii)** Comparing Group 1 with Groups 5, 6, 8, and 9, knowledge base can enhance problem-solving abilities, but introduce 2x-4x overhead, and depends on its size. Feedback mechanism supports more accurate solution selection, improves repair performance and reduces reliance on knowledge base. **(iii)** Fixed repair framework can cover a wide range of issues, it often includes numerous generic steps, which may introduce unnecessary complexity and overhead (Group 3). RustBrain dynamically designs solutions based on code features, avoiding redundant steps and improve accuracy. **(iv)** Specific scenarios exist for the three types of repair solutions for UBs. Fixed repair frameworks often provide inappropriate solutions, exacerbate hallucination and reduce accuracy. In Groups 3, 7, and 10, even with knowledge base support, semantic acceptable fixes could not be achieved. RustBrain, based on code features and feedback mechanism, better guides the generation of solutions for similar error types, avoiding unsuitable fixes and improving both repair accuracy and efficiency.

RQ2 (Accuracy): We evaluated RustBrain's repair effectiveness and robustness using various configurations, including different LLMs and knowledge base usage. Experimental results, shown in Fig. 8 and Fig. 9, compare repair *pass* rates and semantic *exec* rates.

Results indicate that for GPT-4 model, integrating RustBrain achieved a *pass* rate of 90.5% and an *execution* rate of 70.2%, representing an improvement of 25-35% compared to using GPT-4 alone for repair. Furthermore, when RustBrain is integrated with the knowledge base, it achieves the highest *pass* and *execution* rates, averaging 94.3% and 80.4% , respectively. Additionally, although GPT-3.5 performed lower than GPT-4, combining it with RustBrain achieved a comparable level to GPT-4 in terms of fix *pass* and *execution* rate. This suggests that the design in RustBrain compensates for the limitations of the base model, improving the accuracy and consistency in handling UBs. For Claude-3.5, it exhibits initial semantic capabilities in reducing UBs, comparable to GPT-4. However, when combined with RustBrain, Claude-3.5 performs less effectively than GPT-4 in understanding complex dependencies and resolving issues. Compared to standalone Claude-3.5, RustBrain+Claude-3.5 improves the *pass* rate and *execution* rate by 17% and 20%, respectively.

Besides, we also compare RustBrain with the state-of-the-art GPT model, GPT O1. Due to O1's high cost, a subset of error types was analyzed, with results shown in Fig. 10. Despite GPT-O1's exceptional reasoning, its repair effectiveness remains inferior to RustBrain. For uncommon errors like *panic* errors, GPT-O1 fails to provide suitable solutions based on code features. In contrast, RustBrain, leverages the combination of fast and slow thinking to generate more targeted solutions, improving *execution* rate by 35.6%.

RQ3 (Sensitivity): Temperature is a key parameter affecting output diversity in model generation [30]. We conduct sampling tests in Miri dataset and evaluate the stability of the repair results by calculating the confidence intervals [31]. The Confidence Interval(CI) reflects the uncertainty of fixing UBs in LLMs, ensuring that the reported pass and *exec* rates are reliable within a certain probability range (c=95%). In the experiments, we analyze the impact of different temperature settings on the performance of RustBrain in GPT-4.

As shown in Fig. 11, *pass* and *exec* rates reach highest points of 97% and 77% with a temperature of 0.5. Higher temperatures increase the flexibility of outputs. However, this flexibility comes at the cost of semantic integrity, leading to a decrease in semantic acceptability (e.g, 0.7). In contrast, lower temperatures limit flexibility and potentially missing opportunities to repair UBs.

RQ4 (Advancement): To evaluate the effectiveness and efficiency of RustBrain, compared to RustAssistant (the state-of-the-art Rust fix tool) and human experts. As shown in Fig. 12, compared to RustAssisant, the *pass* rate and *execution* rate of RustBrain increased by 33% and 41%, respectively. This indicates that RustBrain offers greater flexibility, breaking the limitations of existing repair frameworks. It is able to address issues that the current frameworks cannot solve, thereby reducing UBs and improving overall safety.

To evaluate RustBrain's effectiveness compared to human experts, we assessed their performance across various error types, as shown in Table I. While human experts excel in debugging, the framework demonstrates competitive performance in addressing UBs caused by complex semantic rules. On average, RustBrain achieves a 7x speedup across different error types, with the gap widening for more complex

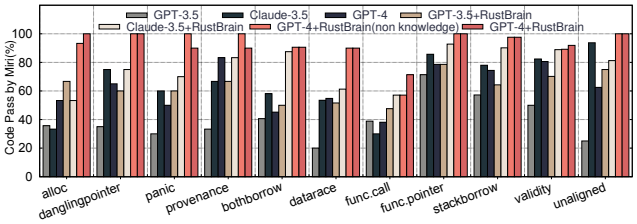


Fig. 8: RustBrain fixes UBs pass by Miri rate.

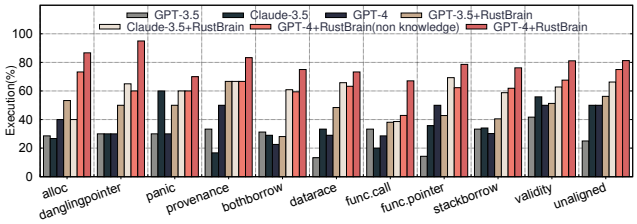


Fig. 9: RustBrain fixes UBs semantic acceptability rate.

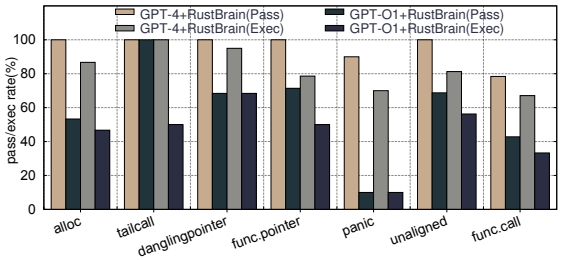


Fig. 10: Comparison of RustBrain with GPT-O1 on UBs Repair.

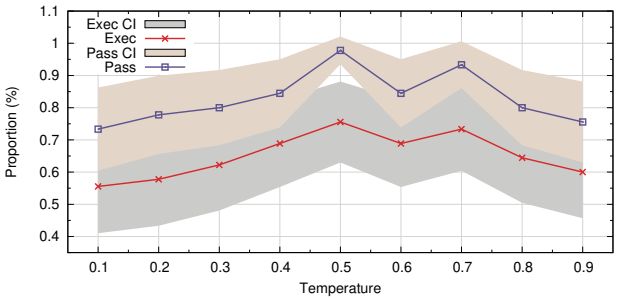


Fig. 11: Different Temperatures on RustBrain Repair Efficacy.

or rare errors. For example, in *function pointer* UBs, which mainly involve type conversion problems and require deeper Rust expertise for guidance, RustBrain shows up to 18x. Furthermore, we note that when the dataset contains similar UBs, RustBrain can reduce dependency on the knowledge base and generate precise solutions through the feedback mechanism, effectively minimizing UBs and overhead, as shown in the red sections of Table I.

Overall, for a fixed framework (such as RustAssisant), the speedup in problem-solving compared to human experts relies on the knowledge in the knowledge base, and its problem-solving ability is limited by the knowledge it contains. RustBrain leverages a feedback mechanism that feeds back evaluated solutions from the slow-thinking phase into the fast-thinking phase. Through this self-learning process, the fast-thinking can generate more precise solutions, without being fully dependent on the design of the knowledge base, reducing the overhead and time to fix UBs

V. Related Work

Efforts to enhance Rust's safety focus on ensuring the safety of Unsafe Rust, categorizes as manual, semi-automated, and fully automated. Manual methods, e.g., formal methods [33], [34], static analysis [35], [36], rely on developer's expertise, enabling precise fixes but are time-consuming and error-prone. Semi-automated approaches use tools for initial error detection, followed by developers' detailed analysis and repair. Examples include symbolic executionn [37], [38]. Full automated aim to minimize human involvement by leveraging machine learning [9], [39] and LLMs [12], [13]. However, its repair accuracy and coverage are limited by the knowledge base, and deep contextual understanding remains a challenge. Our approach employs a collaboration of fast and slow thinking to provide flexible and precise repair solutions while maintaining the safety of the Rust.

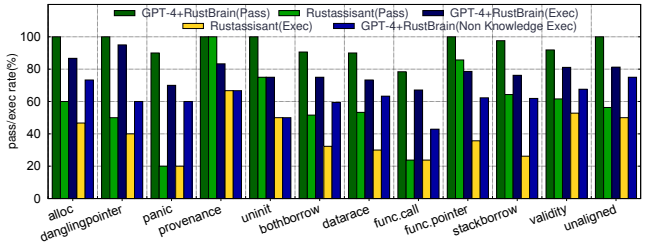


Fig. 12: Comparison of RustBrain with RustAssisant on UBs Repair.

TABLE I: Execution Time of RustBrain against Human

Types	Average Time/s			Speedup
	GPT-4+RustBrain		Human	
	No_knowledge	knowledge		
stack borrow	86	123	366	2.81x
unaligned pointer	74	118	222	4.63x
validity	**79**	**109**	**678**	**8.58x**
alloc	**40**	**77**	**450**	**11.25x**
func. pointer	**39**	**50**	**480**	**12.3x**
provenance	45	64	240	5.33x
panic	76	107	336	4.42x
func. calls	**65**	**74**	**1176**	**18.1x**
dangling pointer	**70**	**74**	**114**	**1.63x**
both borrow	**69**	**87**	**762**	**11x**
concurrency	45	45	144	3.2x
data race	63	91	336	5.3x
Average	**62.6**	**84.9**	**442**	**7.4x**

VI. Conclusion

In this work, we propose a systematic, automated, and dynamically adjustable framework, called RustBrain, to eliminate UBs in Rust project. This framework combine fast and slow thinking two phases, breaking the limitations of traditional frameworks, providing more flexible fix framework, and successfully enhancing Rust code safety. The fast-thinking phase extracts code features to quickly generate multiple feasible fixes. The slow-thinking phase designs Rust feature based agents (including fixing, adaptive rollback and knowledge, etc.) to execute the solutions. The execution results feedback to the fast thinking to continuously improve and optimize the decision-making process. The experimental results show that RustBrain achieve pass rate of 94.3% and execute rate 80.4% in Miri dataset, with up to 7x speedup compared with human experts. For future work, we are interested in providing automated safety enhancements for complex Rust code involving multi-module calls.

VII. Acknowledgments

We would like to thank the anonymous reviewers for the helpful feedback. This work was supported by National Key Research and Development Program (Grant No.2024YFB4405600), the National Natural Science Foundation of China [Grant No.62472086 and No.62472432], the Pre-Research Project [Grant No.31511100101], the Science and Technology Major Special Program of Jiangsu (No. BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

REFERENCES

[1] Matsakis N D, Klock F S. The rust language[J]. ACM SIGAda Ada Letters, 2014, 34(3): 103-104.

[2] StackOverflow. https://stackoverflow.blog/2024/07/24/developers-want-more-more-more-the-2024-results-from-stack-overflow-s-annual-developer-survey/

[3] Jiang R, Dong P, Ding Y, et al. Thetis: A Booster for Building Safer Systems Using the Rust Programming Language[J]. Applied Sciences, 2023, 13(23): 12738.

[4] Qin B, Chen Y, Yu Z, et al. Understanding memory and thread safety practices and issues in real-world Rust programs[C]//Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. 2020: 763-779.

[5] Zhu S, Zhang Z, Qin B, et al. Learning and programming challenges of rust: A mixed-methods study[C]//Proceedings of the 44th International Conference on Software Engineering. 2022: 1269-1281.

[6] Bae Y, Kim Y, Askar A, et al. Rudra: finding memory safety bugs in rust at the ecosystem scale[C]//Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles. 2021: 84-99.

[7] Bang I, Kayondo M, Moon H, et al. TRust: A Compilation Framework for In-process Isolation to Protect Safe Rust against Untrusted Code[C]//32nd USENIX Security Symposium (USENIX Security 23). 2023: 6947-6964.

[8] Cui M, Chen C, Xu H, et al. SafeDrop: Detecting memory deallocation bugs of rust programs via static data-flow analysis[J]. ACM Transactions on Software Engineering and Methodology, 2023, 32(4): 1-21.

[9] Dong Y, Zhang Z, Cui M, et al. SafeNet: Towards mitigating replaceable unsafe Rust code via a recommendation-based approach[J]. Software Testing, Verification and Reliability, 2024: e1875.

[10] Zhong L, Wang Z, Shang J. Ldb: A large language model debugger via verifying runtime execution step-by-step[J]. arXiv preprint arXiv:2402.16906, 2024.

[11] Li H, Hao Y, Zhai Y, et al. Enhancing Static Analysis for Practical Bug Detection: An LLM-Integrated Approach[J]. Proceedings of the ACM on Programming Languages, 2024, 8(OOPSLA1): 474-499.

[12] Wu X, Cheriere N, Zhang C, et al. RustGen: An Augmentation Approach for Generating Compilable Rust Code with Large Language Models[J].

[13] Deligiannis P, Lal A, Mehrotra N, et al. RustAssistant: Using LLMs to Fix Compilation Errors in Rust Code[C]//2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE). IEEE Computer Society, 2024: 267-279.

[14] Daniel K. Thinking, fast and slow[M]. 2017.

[15] Booch G, Fabiano F, Horesh L, et al. Thinking fast and slow in AI[C]//Proceedings of the AAAI Conference on Artificial Intelligence. 2021, 35(17): 15042-15046.

[16] Ganapini M B, Campbell M, Fabiano F, et al. Thinking fast and slow in AI: The role of metacognition[C]//International Conference on Machine Learning, Optimization, and Data Science. Cham: Springer Nature Switzerland, 2022: 502-509.

[17] Miri.https://github.com/rust-lang/miri.

[18] OpenAI. GPT-3.5. https://platform.openai.com/docs/models/gpt-3-5.

[19] OpenAI. GPT-4. https://platform.openai.com/docs/models#gpt-4-turbo-and-gpt-4.

[20] OpenAI. GPT-O1-Preview. https://platform.openai.com/docs/models#o1

[21] Claude 3.5. https://www.anthropic.com/claude/sonnet.

[22] Jung R, Jourdan J H, Krebbers R, et al. RustBelt: Securing the foundations of the Rust programming language[J]. Proceedings of the ACM on Programming Languages, 2017, 2(POPL): 1-34.

[23] Ling M, Yu Y, Wu H, et al. In Rust we trust: a transpiler from unsafe C to safer Rust[C]//Proceedings of the ACM/IEEE 44th international conference on software engineering: companion proceedings. 2022: 354-355.

[24] Rawte V, Sheth A, Das A. A survey of hallucination in large foundation models[J]. arXiv preprint arXiv:2309.05922, 2023.

[25] Jiang C, Xu H, Dong M, et al. Hallucination augmented contrastive learning for multimodal large language model[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2024: 27036-27046.

[26] Chen S, Li B. Toward Adaptive Reasoning in Large Language Models with Thought Rollback[C]//Forty-first International Conference on Machine Learning.

[27] Xu K, Sun J, Hu Y, et al. MEIC: Re-thinking RTL Debug Automation using LLMs[J]. arXiv preprint arXiv:2405.06840, 2024.

[28] Feng S, Shi W, Wang Y, et al. Don't Hallucinate, Abstain: Identifying LLM Knowledge Gaps via Multi-LLM Collaboration[J]. arXiv preprint arXiv:2402.00367, 2024.

[29] Sun K, Xu Y E, Zha H, et al. Head-to-tail: How knowledgeable are large language models (llm)? AKA will llms replace knowledge graphs?[J]. arXiv preprint arXiv:2308.10168, 2023.

[30] Renze M, Guven E. The effect of sampling temperature on problem solving in large language models[J]. arXiv preprint arXiv:2402.05201, 2024.

[31] Smithson M. Confidence intervals[M]. Sage, 2003.

[32] Dewey J. Reconstruction in philosophy[M]. Beacon Press, 1971.

[33] Jung R, Jourdan J H, Krebbers R, et al. RustBelt: Securing the foundations of the Rust programming language[J]. Proceedings of the ACM on Programming Languages, 2017, 2(POPL): 1-34.

[34] Astrauskas V, Müller P, Poli F, et al. Leveraging Rust types for modular specification and verification[J]. Proceedings of the ACM on Programming Languages, 2019, 3(OOPSLA): 1-30.

[35] Li Z, Wang J, Sun M, et al. MirChecker: detecting bugs in Rust programs via static analysis[C]//Proceedings of the 2021 ACM SIGSAC conference on computer and communications security. 2021: 2183-2196.

[36] Cui M, Chen C, Xu H, et al. SafeDrop: Detecting memory deallocation bugs of rust programs via static data-flow analysis[J]. ACM Transactions on Software Engineering and Methodology, 2023, 32(4): 1-21.

[37] Lindner M, Fitinghoff N, Eriksson J, et al. Verification of safety functions implemented in Rust-a symbolic execution based approach[C]//2019 IEEE 17th International Conference on Industrial Informatics (INDIN). IEEE, 2019, 1: 432-439.

[38] Lindner M, Aparicius J, Lindgren P. No panic! Verification of Rust programs by symbolic execution[C]//2018 IEEE 16th International Conference on Industrial Informatics (INDIN). IEEE, 2018: 108-114.

[39] Lu X, Zhang Z, Xu H. Safe Rust Code Recommendation Based on Siamese Graph Neural Network[C]//2022 IEEE 27th Pacific Rim International Symposium on Dependable Computing (PRDC). IEEE, 2022: 1-11.

[40] Jung R. Understanding and evolving the Rust programming language[J]. 2020.

[41] Kang S, Chen B, Yoo S, et al. Explainable automated debugging via large language model-driven scientific debugging[J]. arXiv preprint arXiv:2304.02195, 2023.

ATLAS: A Self-Supervised and Cross-Stage Netlist Power Model for Fine-Grained Time-Based Layout Power Analysis

Wenkai Li[†], Yao Lu[†], Wenji Fang, Jing Wang, Qijun Zhang, Zhiyao Xie[*]
Hong Kong University of Science and Technology
{wlidm, yludf, wfang838, jwangjw, qzhangcs}@connect.ust.hk, eezhiyao@ust.hk
([†]equal contribution, [*]corresponding author)

***Abstract*—Accurate power prediction in VLSI design is crucial for effective power optimization, especially as designs get transformed from gate-level netlist to layout stages. However, traditional accurate power simulation requires time-consuming back-end processing and simulation steps, which significantly impede design optimization. To address this, we propose ATLAS, which can predict the ultimate time-based layout power for any new design in the gate-level netlist. To the best of our knowledge, ATLAS is the first work that supports both time-based power simulation and general cross-design power modeling. It achieves such general time-based power modeling by proposing a new pre-training and fine-tuning paradigm customized for circuit power. Targeting golden per-cycle layout power from commercial tools, our ATLAS achieves the mean absolute percentage error (MAPE) of only 0.58%, 0.45%, and 5.12% for the clock tree, register, and combinational power groups, respectively, without any layout information. Overall, the MAPE for the total power of the entire design is <1%, and the inference speed of a workload is significantly faster than the standard flow of commercial tools.**

I. Introduction

Power is an increasingly important objective in modern chip design. As design complexity keeps scaling up, it is increasingly costly for chip designers to get accurate power values of their design. It is even more challenging to simulate time-based (e.g., per-cycle) power values, which enables the analysis of peak power and power fluctuations (Ldi/dt). As Fig. 1 shows, starting at a gate-level netlist, designers need to complete the design whole layout process and then employ target workloads (in .fsdb or .vcd format) to simulate time-based power consumption based on commercial EDA tools [1], [2]. Both layout and power simulation can take days or even weeks for large designs or workloads. In summary, accurate, efficient, time-based power models are in high demand.

Prior Power Modeling Works. In recent years, various novel data-driven power modeling techniques [3]–[9] have been proposed, providing unprecedented early-stage and fast power evaluations based on machine learning (ML) power models. Representative power modeling works have been summarized in Table I. However, no prior works can provide *time-based power* and generalize across *different designs* simultaneously. We categorize existing power modeling solutions into two main types:

- The first type of works (e.g., PRIMAL [3], APOLLO [4]) can provide accurate and time-based power values, but they do not support general cross-design models. Instead, they [3], [4] require training a new design-specific power model from scratch for each new design. This development process, especially the label collection step, is highly time-consuming.
- The second type of works [5]–[9] can provide a general power model that applies to new designs, which are unknown during model training. Perhaps due to the difficulty of generalization, none of these works further provide time-based power values. Moreover, most of these works [5]–[7], [9] do not model power based on different workloads. Instead, they only model the average power value based on the propagation of user-defined toggle rates (i.e., vectorless power values).

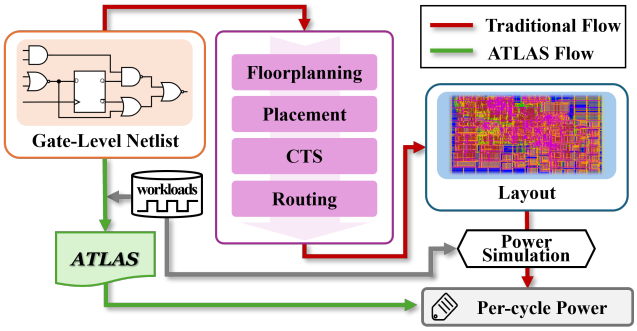


Fig. 1: Overview of ATLAS for time-based netlist power modeling. Standard traditional power simulation for post-layout design is time-consuming due to both layout steps and per-cycle power simulation. ATLAS achieves significant acceleration over the standard flow.

In addition, many prior power modeling solutions [3]–[7] do not model the ultimate power after design layout. For faster data collection, they adopt power values simulated at the gate-level netlist stage as labels, skipping the layout process. As a result, they are not validated on capturing ultimate post-layout power, which is heavily affected by many factors such as accurate metal wire capacitance, timing optimization on netlist (e.g., buffer insertion, netlist reconstruction), the clock tree, etc.

Our Solution ATLAS. In this work, we propose ATLAS, targeting efficient evaluation of the time-based post-layout power of any given gate-level netlist. To the best of our knowledge, ATLAS is the first work that supports both *time-based* power simulation and general *cross-design* power modeling. Moreover, ATLAS targets the most accurate power label from *layout stage* and *realistic designs* (e.g., out-of-order CPU designs instead of small blocks). Compared with standard simulation flow based on EDA tools, the general ATLAS solution bypasses both the layout process and time-based power simulation, achieving significant speedups.

ATLAS achieves unprecedented general time-based power modeling based on a customized *pre-training* and *fine-tuning* paradigm. It proposes the following novel strategies:

- **Sub-module generation:** ATLAS first splits each design into

Power Models	Applied Stage	Support Workloads	Time-Based	Cross-Design	Target Layout
PRIMAL [DAC'20] [3]	RTL	Yes	Yes	No	No
APOLLO [MICRO'21] [4]					
Sengupta et al. [ICCAD'22] [5]		No	No	Yes	
SNS [ISCA'22] [6]					
SNS V2 [MICRO'23] [7]					
MasterRTL [ICCAD'23] [8]		Yes			Yes
PowPredicCT [DAC'24] [9]	Layout	No			
ATLAS	Netlist	**Yes**	**Yes**	**Yes**	**Yes**

*GRANNITE [10] estimates toggle rate instead of power, thus not listed in the table. It is neither time-based nor targeting layout.

TABLE I: Summary of representative ML-based power models.

non-overlapping sub-modules. ATLAS will evaluate the per-cycle post-layout power of each small sub-module.

- **Pre-training:** A general netlist encoder model is pre-trained based on multiple self-supervised learning tasks without power labels. The encoder, based on graph transformer, is trained by guessing the masked toggle rate and node type, as well as recognizing the alignment between regular post-synthesis netlist[1] and the ultimate post-layout netlist. The encoder will encode each sub-module into a general embedding (i.e., vector) with rich design information. Such an informative embedding effectively supports challenging power modeling tasks.
- **Fine-tuning:** Based on the embedding from the pre-trained encoder, we fine-tune different lightweight models for three power groups: combinational logic, register, and clock tree.

We evaluate ATLAS on different designs with 300K to 600K cells under realistic workloads. ATLAS achieves high accuracy in per-cycle power modeling, with only $< 1\%$ error percentage on average. ATLAS is up to $1000\times$ faster than the traditional commercial flow by bypassing both the layout process and standard time-based power simulation. The results indicate the superior predictive capability of ATLAS, demonstrating its effectiveness in cross-stage, cross-design, and time-based power prediction.

II. Methodology Overview

This section provides an overview of ATLAS. ATLAS includes three major steps: design netlist preprocessing (Sec. III and Fig. 2), pre-training (Sec. IV and Fig. 3), and fine-tuning (Sec. V and Fig. 4).

Preprocessing (Sec. III). The flow begins with netlist preprocessing, preparing the dataset for ATLAS pre-training. In this step, we split each design in the gate-level netlist into many non-overlapping sub-modules. We transform each sub-module into the graph format and annotate related features (e.g., cell type, cell power from liberty file) to each node (i.e., cell instance). During training, the pre-processed data will be adopted to pre-train the encoder. During inference, ATLAS will encode each sub-module and predict its per-cycle power value.

Pre-training (Sec. IV). The pre-training step will train a general circuit encoder, targeting two general learning goals: 1) recognizing the structure and functionality of netlists; 2) learning the alignment between netlist and layout. This encoder is pre-trained with five novel and customized self-supervised tasks to recognize the inherent semantics and structures of sub-modules, without relying on any power labels. After the pre-training, the encoder is expected to capture the transformations made on each sub-module (e.g., buffer insertion, reconstruction, clock tree) during layout and encode the information into its output embedding vector.

Unlike previous self-supervised learning tasks on netlists [11]–[13] that only target combinational circuits, we target realistic sequential designs. Moreover, we incorporate per-cycle toggle information into one of the five self-supervised learning tasks, allowing ATLAS to learn about signal propagation.

Fine-tuning (Sec. V). The final step of ATLAS fine-tunes different lightweight models for three different power groups: combinational logic, sequential logic, and clock network. These models leverage the embedding vectors generated during pre-training as input, generating fine-grained per-cycle power value of each power group for each sub-module. Summing up all power groups of all sub-modules provides the total power for each cycle.

[1]In this manuscript, if not explicitly described, by default "netlist" refers to the post-synthesis gate-level netlist before layout. The post-layout netlist (with timing optimization and clock tree) will be explicitly mentioned.

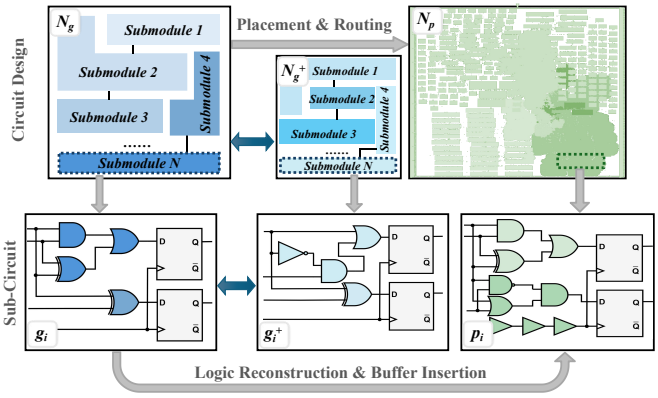


Fig. 2: Netlist Preprocessing of ATLAS. The preprocess includes circuit segmentation in both gate-level netlist N_g, N_g^+ and post-layout netlist N_p. N_g^+ represents a Boolean-equivalent expression of N_g and both are functionally equivalent with similar circuit structures. For each sub-circuit g_i in N_g, there exists an equivalent gate-level expression g_i^+ and a transformed expression p_i after layout.

III. Netlist Preprocessing

The netlist preprocessing step is essential for preparing the dataset used in ATLAS. This involves generating netlists from different stages and collecting key features that facilitate effective learning.

A. Sub-module Format

Fig. 2 illustrates the netlist preprocessing process. Given a gate-level netlist N_g, we split it into a set of non-overlapping sub-modules $N_g \in \{g_1, g_2, ..., g_i\}$. Then this original netlist N_g will be transformed to a corresponding post-layout netlist N_p, which will similarly yield a set of sub-modules $N_p \in \{p_1, p_2, ..., p_i\}$. During this standard layout process, the netlist will be optimized for better timing through buffer insertion, netlist reconstruction, etc. Also, the clock tree will be synthesized and added to the design.

Most previous works on circuit quality prediction [6]–[8] choose to split circuits into multiple *logic cones*, then model the PPA of each logic cone. Each input code corresponds to one flip-flop, including all input logic and flip-flops that drive it. In comparison, we split the circuit into sub-modules, and then encode and evaluate the power of each sub-module. This approach offers obvious advantages:

Non-overlapping: Works [6]–[8] based on logic cones unavoidably involve significant overlapping between different cones, making them actually inappropriate for power modeling. The significant logic overlap will make the summing up of the power of each cone much larger than the total design power. In contrast, our approach uses non-overlapping sub-modules defined by specific functional roles. We can accurately estimate the power of a larger component or the whole design by summing the powers of all its constituent sub-modules.

Fine-grained Analysis: By splitting based on sub-module, the ATLAS model achieves fine-grained predictions not only in terms of time but also at the component level. Since each sub-module is determined by the inherent functionality of the netlist, the power of each sub-module provides fine-grained feedback to designers.

B. Netlist Collection for Self-supervised Learning

To support the learning of the general intrinsic netlist information across stages, we will collect different netlists during preprocessing. The collected netlists will be applied in self-supervised encoder pre-training in the next step. As shown in Fig. 2, we will generate three types of netlist for model pre-training: N_g, N_p and N_g^+. The N_g is the original gate-level netlist by synthesizing RTL code. The N_g^+ and N_p are introduced below.

1) For recognition of structure and functionality: By applying logic-invariant transformations on N_g, we can derive another gate-level netlist N_g^+, which has the same functionality but a different structure. As a basic property of circuits, different gate-level netlists can implement the same functionalities. Our encoder model will learn the structural and functional similarities among sub-modules by recognizing structures with the same functionality. During encoder pre-training, as we will introduce, N_g provides original samples, while N_g^+ provides corresponding positive samples. Sub-module that differs from the target sub-module serves as a negative sample.

2) For alignment with the layout stage: We employed commercial tools to perform layout on N_g, with each layout step involving optimization. Ultimately, we obtained the post-routing design layout, and we refer to the corresponding post-layout netlist as N_p. During encoder pre-training, as we will introduce, the objective is to learn the cross-stage alignment between N_g and N_p, making the embedding of the gate-level netlist E_g to capture and reflect the information of post-layout netlist E_p.

C. Feature Collection

We split all netlists from Section III-B into sub-modules. Then we convert each sub-module to a directed graph (DG). Each node corresponds to each cell instance and directed edges are the wires connecting these nodes. By representing sub-modules as DGs, we will leverage the advantages of graph transformer models to capture the circuit structures and semantics.

After converting all sub-modules to DGs, we proceed to collect features for every graph node. To benefit the subsequent learning process, we have carefully selected four types of features: Node Type, Per-cycle Toggle, Cell Internal and Leakage Power.

1) Node Type: We categorize all cells according to their functionality into 18 types, using one-hot encoding to represent the cell type. ATLAS will learn to recognize the node type and node type effectively conveys each cell's functional characteristics.

For example, the clock tree in post-layout netlist N_p involves the interconnection of many clock-related cells, such as *Clock Buffer*, *Clock AND*, and *Clock Multiplexer*, all categorized as the `CK` type by us. By learning to recognize masked `CK`-type nodes during pre-training, the encoder is guided to capture the effect of the clock tree.

2) Per-cycle Toggle: Per-cycle toggle represents switching activity at a node's output port during a specific timestamp under real workloads. Including a per-cycle toggle aims for ATLAS to learn the signal propagation relation between nodes and to predict signal propagation. During the pre-training, the encoder will learn by predicting the masked toggle behavior.

3) Cell Internal and Leakage Power: For each cell instance, both internal and leakage power values will be parsed from the lookup tables in the `.lib` files from the technology library according to its cell type. These standard cell power values provide important cell information and are directly power-related.

IV. ATLAS Pre-training

ATLAS trains a general circuit encoder to convert each sub-module into a representation. Fig. 3 illustrates the entire self-supervised pre-training process of the encoder model based on efficient graph transformer [14]. The encoder will generate an embedding for each individual node, as well as an additional overall graph embedding for the whole input graph (i.e., corresponding sub-module).

The encoder is pre-trained on the unlabeled circuit dataset, targeting the following two general learning goals: 1) recognizing the structure, functionality, and toggle propagation of netlists; 2) learning the alignment between netlist and layout. The encoder is expected to

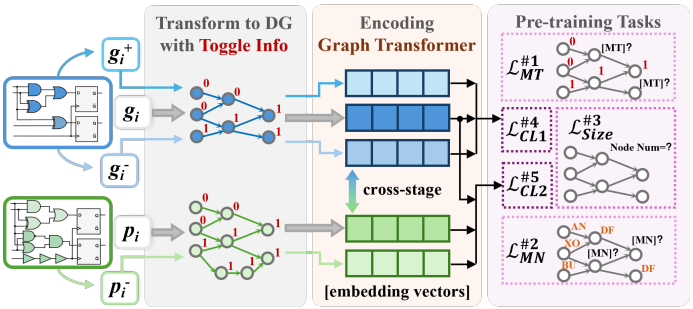


Fig. 3: ATLAS Pre-training. This framework employs a self-supervised and cross-stage flow that incorporates two types of contrastive learning. Additionally, ATLAS includes predictions for `[MASK_TOGGLE]` ([MT]) sub-module size and `[MASK_NODE_TYPE]` ([MN]).

encode the learned information into its output embedding vector. To achieve this, we carefully designed five encoder pre-training tasks: ❶ Masked toggle propagation learning; ❷ Masked node type learning; ❸ Sub-module size learning; ❹ Gate-level netlist contrastive learning; ❺ Cross-stage alignment contrastive learning.

Our proposed five self-supervised learning tasks can be roughly categorized into three types, as summarized below.

1) Masked node recovery (Tasks ❶, ❷). We will mask (i.e., hide) important node properties (e.g., type, toggle) of randomly selected nodes. A temporary MLP-based classification head will predict the masked node properties based on the encoder-generated node embedding. During pre-training, the encoder and classification head (to be discarded after pre-training) will be trained end-to-end to maximize accuracy. The encoder will thus learn to encode the structural information of connected circuit operators within the circuit graph.
2) Size recognition (Task ❸). A temporary MLP-based regression head will predict the total number of nodes in the given graph (i.e., sub-module netlist) based on encoder-generated graph embedding. During pre-training, the encoder and regression head (to be discarded after pre-training) will be trained end-to-end to maximize accuracy. The encoder will learn to encode more global graph information in the overall graph embedding.
3) Netlist contrastive learning (Tasks ❹, ❺). The encoder will be trained to minimize the embedding distance of netlists of the same sub-module with the same functionality (i.e., positive samples), while maximizing the embedding distance of different sub-modules (i.e., negative samples).

We formally introduce and formulate each learning task below.

Task #1 Masked toggle propagation learning: To train our encoder model to handle per-cycle workloads, we mask the per-cycle toggle information of randomly selected nodes. A temporary MLP classifier will predict whether the masked node is toggling based on encoder-generated node embeddings. This pre-training task trains the encoder to capture the toggle information and the propagation of toggles. Specifically, we randomly mask the toggle feature (0 or 1) of selected nodes and replace it with a special token `[MASK_TOGGLE]`. Subsequently, we predict whether these nodes will toggle based on the embedding of the masked node, which also captures neighboring nodes' information. We represent the input circuit in masked graph format as $\hat{G}$. The ground-truth toggle feature of the masked nodes is represented by $\boldsymbol{t}_{\hat{G}}^{msk}$. The predicted toggle feature for the masked nodes is represented by $\boldsymbol{p}^{msk}(\hat{G})$, which is based on node embeddings[2]. The objective is to minimize the ross-entropy (CE) loss

[2]To simplify the formulation, the encoder model does not directly appear in the loss term in Eq. (1). The prediction $\boldsymbol{p}^{msk}(\hat{G})$ is based on encoder model.

between $\boldsymbol{t}_{\hat{G}}^{msk}$ and $\boldsymbol{p}^{msk}(\hat{G})$, as expressed in the following formula:

$$\mathcal{L}_{\text{MT}}^{\#1} = \mathbb{E}_{\hat{G}\sim\mathcal{D}} CE\left(\boldsymbol{t}_{\hat{G}}^{msk},\ \boldsymbol{p}^{msk}(\hat{G})\right) \tag{1}$$

where $\mathbb{E}_{\hat{G}\sim\mathcal{D}}$ represents the expectation $\mathbb{E}$ over the circuit graph dataset $\mathcal{D}$.

Task #2 Masked node type learning: Similar to the toggle task, we randomly mask the node type information and replace it with a special token [MASK_NODE_TYPE]. A temporary MLP classifier will predict the one-hot encoding for the masked node types based on the embeddings of the masked node. The input circuit's masked representation is denoted as $\hat{G}$, with the ground-truth type for the masked nodes represented by $\boldsymbol{c}_{\hat{G}}^{\text{msk}}$ and the predicted type denoted by $\boldsymbol{q}^{\text{msk}}(\hat{G})$. The objective is to minimize the cross-entropy (CE) loss between the true one-hot encoding $\boldsymbol{c}_{\hat{G}}^{\text{msk}}$ and the predicted one-hot encoding $\boldsymbol{q}^{\text{msk}}(\hat{G})$, expressed as:

$$\mathcal{L}_{\text{MN}}^{\#2} = \mathbb{E}_{(E_g)\sim\mathcal{D}} CE\left(\boldsymbol{c}_{\hat{G}}^{\text{msk}},\ \boldsymbol{q}^{\text{msk}}(\hat{G}))\right) \tag{2}$$

Task #3 Sub-module size learning: The size of a circuit design is often directly correlated with its overall power consumption. We propose a pre-training task to recognize the size of each sub-module. A temporary MLP regressor will predict the number of nodes (Num_{pre}) in the whole graph (i.e., cell count in sub-module) based on the overall graph embedding. The objective is to minimize the MSE between predicted node number Num_{pre} and the actual number of nodes Num_{true}:

$$\mathcal{L}_{\text{Size}}^{\#3} = \mathbb{E}_{\hat{G}\sim\mathcal{D}}\left[(\text{Num}_{pre} - \text{Num}_{true})^2\right] \tag{3}$$

Task #4 Gate-level netlist contrastive learning: To capture the functionalities of each sub-module, we employed contrastive learning on the netlist. Specifically, for the gate-level netlist N_g with many sub-modules g_i, we generate another N_g^+ through functionally equivalent transformations. N_g^+ provides new sub-modules g_i^+ with identical functionalities but different structures. At this task, we consider g_i and g_i^+ as a pair of positive samples. All other sub-modules in the batch that exhibit different functionalities are treated as negative samples (g_i^-) of g_i. The contrastive objective pulls functionally similar sub-modules closer together in their respective embedding spaces while pushing dissimilar sub-modules further apart. We denote the embeddings of g_i, g_i^+, and g_i^- as E_g, E_g^+, and E_g^-, respectively. Specifically, we minimize the InfoNCE loss [15] used for sub-module embeddings, defined as CL, as follows:

$$\mathcal{L}_{\text{CL1}}^{\#4} = \mathbb{E}_{(E_g)\sim\mathcal{D}} CL\left(E_g, E_g^+, E_g^-\right) \tag{4}$$

Task #5 Cross-stage alignment contrastive learning: For cross-stage contrastive learning, similar to Task #4, we aim for gate-level netlist N_g to be closer in the embedding space to the corresponding post-layout netlist N_p. This alignment task targets capturing layout information from N_p in netlist embedding. For such contrastive learning, the original sample is still g_i, while the positive sample is p_i, and the negative samples p_i^- comprise all other functionally distinct sub-modules in the batch. We denote the embeddings of p_i and p_i^- as E_p and E_p^-, respectively. We also adopt the InfoNCE loss function for this purpose:

$$\mathcal{L}_{\text{CL2}}^{\#5} = \mathbb{E}_{(E_g,E_p)\sim\mathcal{D}} CL\left(E_g, E_p, E_p^-\right) \tag{5}$$

Finally, we formulate the complete self-supervised pre-training objective of our model by jointly employing the five tasks:

$$\mathcal{L} = \mathcal{L}_{\text{MT}}^{\#1} + \mathcal{L}_{\text{MN}}^{\#2} + \mathcal{L}_{\text{Size}}^{\#3} + \mathcal{L}_{\text{CL1}}^{\#4} + \mathcal{L}_{\text{CL2}}^{\#5} \tag{6}$$

Regarding the encoder of ATLAS, traditional Graph Neural Networks (GNNs) [16] and Graph Transformers [17] [18] can be resource-intensive when applied to large-scale graphs. Considering that some of our DGs contain tens of thousands of nodes, we employ an efficient Graph Transformer, SGFormer [14], as the encoder for ATLAS. In our DGs, SGFormer achieves propagation capabilities across any node using a single global attention network of linear complexity, while maintaining linear complexity with respect to the number of nodes. Furthermore, it does not require positional encodings, feature or graph preprocessing, or additional loss functions.

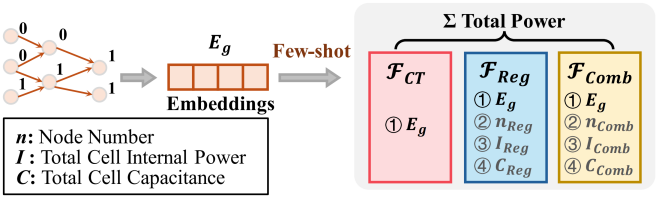


Fig. 4: ATLAS Fine-Tuning. The total post-layout power is divided into different groups as three separate fine-tuning tasks, which are predicted by three fine-tuning models. Other than embedding vectors E_g, ATLAS uses cell internal power, cell capacitance, and node number as features, avoiding the need for cumbersome feature engineering and complex machine learning models.

V. ATLAS Fine-Tuning

To apply pre-trained ATLAS on downstream power modeling tasks, as shown in Fig. 4, we divide the total power into three power groups: the clock tree power, the combinational logic power, and the register power[3]. Each power group will be modeled by its respective fine-tuning model, denoted as $\mathcal{F}_{CT}$, $\mathcal{F}_{Comb}$, and $\mathcal{F}_{Reg}$. These fine-tuned power models differ from purely supervised methods by utilizing the embedding E_g from the encoder as features, rather than relying on cumbersome feature engineering and complex ML models to process raw circuits.

Regarding the clock tree group, it is completely absent in the post-synthesis netlist N_g and appears only in post-layout netlist N_p, which is unknown during inference. Therefore, we do not incorporate any additional features in addition to the embedding. We rely solely on the embedding E_g to predict the power of the clock tree group. This approach will demonstrate that our encoder model has effectively learned the alignment between N_g and N_p.

For both combinational and register groups, we choose to combine the embeddings E_g with additional relevant features available from netlist N_g. For the combinational group, we utilize the number of combinational nodes n_{Comb} of the post-synthesis netlist, along with the total cell internal power I_{Comb} and total cell capacitance C_{Comb} of the combinational group as features. The cell internal power and cell capacitance of each node are collected from the lookup tables in the `.lib` file of the technology library. To calculate the total cell internal power I_{Comb} and total cell capacitance C_{Comb}, we multiply each cell internal power and capacitance in the combinational section by the per-cycle toggle of their output pins and sum up the results.

As for the register group, similar to the combinational group, we adopt the number of register nodes n_{Reg}, the total cell internal power I_{Reg} and total cell capacitance C_{Reg} of the register group from N_g as additional features for fine-tuning.

As a result, our fine-tuning models are lightweight and easily integrable, including tree-based models, such as XGBoost [19]. So

[3]Our register power group includes (and is dominated by) the power of the clock pin in each register. Accordingly, our clock tree power group does not include such register clock pin power.

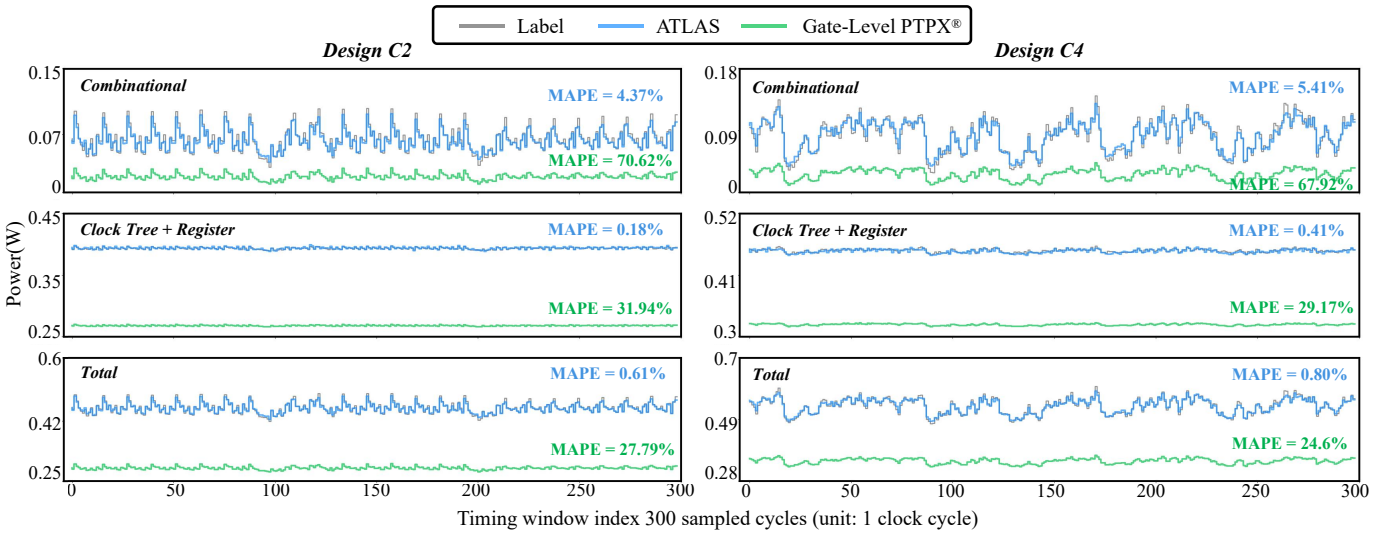


Fig. 5: Power prediction results from ATLAS across 300 cycles for *C2* and *C4* under W_1. The labels were obtained by simulating the netlist and SPEF files using PTPX® after detailed routing with Innovus®. The Gate-Level PTPX® values, which are obtained from the gate-level netlist, illustrate the significant differences in netlist power between gate level and layout.

the total predicted power is:

$$P_{total} = \mathcal{F}_{CT}(E_g) + \mathcal{F}_{Comb}(E_g, n_{Comb}, I_{Comb}, C_{Comb}) + \mathcal{F}_{Reg}(E_g, n_{Reg}, I_{Reg}, C_{Reg}) \quad (7)$$

VI. Experimental Results

A. ATLAS Implementation and Experiment Data Generation

We implement our ATLAS using PyTorch and PyTorch Geometric (PyG), a library built on top of PyTorch that facilitates writing and training GNNs. Our pre-training is conducted on a Linux machine equipped with an NVIDIA A6000 GPU, while the fine-tuning tasks are performed on an Intel Xeon processor with 128GB of memory.

RTL simulation on workloads is based on Synopsys VCS®[20] (VCS). The logic synthesis is executed at a clock frequency of 1GHz using Synopsys Design Compiler®[21] (DC). In our experiments, we utilized the TSMC 40nm standard cell library [22], along with the corresponding Memory Compiler. We utilized the Innovus® to perform mixed-size placement, clock tree synthesis, and routing, with each step including timing optimization, ultimately resulting in the chip design layout. After detailed routing with Innovus®, we dump out the post-layout ultimate netlist and corresponding RC values in Standard Parasitic Extraction Format (SPEF) files. We conduct accurate layout-stage power simulation based on the post-layout netlist, SPEF file, and different workloads (denoted as W_1, W_2). The per-cycle ground-truth layout power simulation is performed based on Synopsys PrimeTime PX® [1] (PTPX®).

In this work, we generated a dataset with six different realistic designs (named C1 to C6 from smallest to largest) that support workload simulation. Table II presents the cell count statistics for six different designs at both the gate-level and post-layout stages. The cell counts range from approximately 300,000 to 600,000, indicating their difference. For the same design, the increase in cell count from the gate-level stage to the post-layout stage reflects the impact of timing optimization and clock tree synthesis during the layout process.

For ATLAS pre-training stage, our dataset comprises aligned 3,253 DG pairs of N_g, N_g^+, N_g^- and N_p for each cycle. We employed four designs (*C1*, *C3*, *C5* and *C6*) for training and two designs (*C2* and *C4*) for testing, ensuring absolute no overlap between the training and testing designs. Ultimately, the number of DGs in the training and testing datasets is 92,500 and 37,640, respectively. During the encoder pre-training process, the five self-supervised tasks are trained simultaneously for 60 epochs, requiring approximately 21 hours. Despite such training runtime, please notice that the encoder only needs to be pre-trained once and then applied to different designs. We employ ReLU as the activation function and set the batch size to 16, with a learning rate of 1e-4 using the Adam optimizer. In the fine-tuning phase, we use XGBoost with 500 estimators and a depth of 5, taking only several seconds for training.

	C1	*C2*	*C3*	*C4*	*C5*	*C6*
Gate-level	289384	322664	389120	399486	465129	597877
Post-layout	301650	340923	412186	422391	494614	638666

TABLE II: The statistics of the gate counts for the six designs at the gate-level and post-layout stages.

B. Power Modeling Experiment Setup

Commericial Tool as Baseline. Our experiment will evaluate the modeling accuracy and efficiency of per-cycle post-layout power based on post-synthesis netlists (without any layout information). Moreover, the tested design is strictly not *seen* by the power model during training. As summarized in the introduction, prior works either require a design-specific model for time-based power modeling [3], [4] or can only evaluate the average power [5]–[9]. As a result, no prior ML-based power models apply to this challenging cross-design per-cycle power modeling task. Moreover, it is very challenging to naturally adapt prior works to our brand-new tasks not supported originally. Therefore, we choose to compare ATLAS with the standard commercial tool at the post-synthesis netlist stage.

Exclusion of Memory Group from Results. In our analysis of power consumption, memory (i.e., SRAM) accounts for almost half of the total power in the entire design. But for this memory power group, we can predict its power with very high accuracy, without too much effort at the netlist stage. Since the SRAM macro is unchanged during layout, we developed a basic ML model based on the toggle activities of memory ports from workload, and energy values from lookup tables in SRAM `.lib` files. Our model achieves an error of only 0.5% without relying on any power simulators. Given such high accuracy in memory power prediction, incorporating the memory power model into ATLAS would lead to a lower error, but primarily dominated by the memory group. To demonstrate the benefits of our ATLAS effectively, we choose to exclude this easier memory group in results. Instead, we focus on predicting the more challenging other powers of the combinational, clock tree, and register groups.

Design & Workload		Error Percentage of ATLAS					Error Percentage of Gate-Level PTPX®				
		Combinational	Clock Tree	Register	Clock Tree + Register	Total	Combinational	Clock Tree	Register	Clock Tree + Register	Total
C2	W_1	4.37%	0.17%	0.27%	0.18%	0.61%	70.62%	100%	2.36%	31.94%	27.79%
	W_2	5.35%	0.15%	0.32%	0.22%	0.57%	71.01%	100%	2.22%	31.93%	27.81%
C4	W_1	5.41%	1.07%	0.54%	0.41%	0.80%	67.92%	100%	2.34%	29.17%	24.60%
	W_2	5.34%	0.93%	0.68%	0.67%	1.13%	69.36%	100%	2.44%	29.23%	25.08%
Average		5.12%	0.58%	0.45%	0.37%	0.78%	69.73%	100%	2.34%	30.57%	26.32%

TABLE III: MAPE (%) of design *C2* and *C4* under two workloads W_1 and W_2.

As introduced Section V, we implemented three corresponding fine-tuned models: $\mathcal{F}_{CT}$, $\mathcal{F}_{Comb}$, and $\mathcal{F}_{Reg}$.

We evaluate the accuracy of per-cycle power modeling with Mean Absolute Percentage Error (MAPE) between label Y_i and prediction $\hat{Y}_i$, assuming altogether m cycles in the tested workload.

$$\text{MAPE} = \frac{1}{m}\sum_{i=1}^{m}\left|\frac{Y_i - \hat{Y}_i}{Y_i}\right| \times 100\% \tag{8}$$

C. Power Prediction Results

Fig. 5 visualizes power prediction results (ATLAS) and the actual power values (Labels) over 300 cycles for testing designs *C2* and *C4*, under workload W_1. For comparison, we present the commercial tool's simulation result on exactly the same gate-level netlist and workload (Gate-Level PTPX®). For both *C2* and *C4*, the overall power trace shapes of ATLAS across the three power groups closely resemble the labels, with the total MAPE being only 0.61% and 0.80%, respectively. In comparison, the gate-level PTPX® error is higher than 25% for both designs. Its power trace also differs significantly from the label. This error shows the gap between the power of the gate-level netlist and the power of the ultimate layout.

In Table III, we provide a more comprehensive comparison of results for ATLAS and Gate-Level PTPX®, which displays the MAPE for different power groups under both workloads W_1 and W_2. Gate-Level PTPX® still exhibits the $> 25\%$ prediction accuracy in total power in all cases, while ATLAS shows $< 1\%$ error.

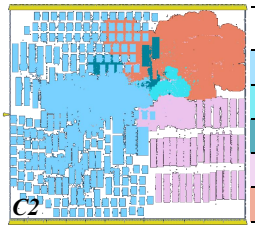
In terms of the most challenging combinational group prediction, ATLAS shows an average MAPE of 5%, while Gate-Level PTPX® has a considerably higher MAPE of 69%. Since there is no clock tree in the gate-level netlist, Gate-Level PTPX® shows a MAPE of 100%. ATLAS demonstrates a remarkably low MAPE of only 0.58%.

In summary, standard commercial tools at the gate-level stage show $> 25\%$ error on time-based layout power due to the lack of layout information. ATLAS, by pre-training and fine-tuning, well captures the potential impact of layout and demonstrates a remarkable $< 1\%$ error for the per-cycle power of new circuit design.

Fig. 6 further illustrates the ATLAS-predicted power of five major components in the design $C2$. The prediction of each component's power is the summation of the predicted power of all its sub-modules. Combined with the layout, the component power predictions further provide power distributions on the layout. In the table on the right of Fig. 6, we present each component's labels, ATLAS power predictions, and the associated MAPE values. It is evident that the power predictions vary across different components, with some over-estimation and others under-estimation. The component error is slightly higher than total power but mostly maintained $< 5\%$.

D. Runtime Comparison

Table IV presents the time taken for back-end processing using Innovus, the time for power simulations using PTPX®, and the time taken by ATLAS to directly predict power. Here we included the runtime used for the entire preprocessing step in ATLAS. We calculated the average time (in seconds) for six designs. Compared with the complete traditional flow that performs both complete layout and time-based power simulations, ATLAS finished estimating the 300-cycle workload with $> 1000\times$ less runtime.



C2 Component	Label (*W*)	ATLAS (*W*)	MAPE (%)
frontend	0.1894	0.1898	1.01
lsu	0.0217	0.0229	5.59
ptw	0.0094	0.0092	2.07
dcache	0.0551	0.0551	1.44
core	0.1821	0.1791	1.68

Fig. 6: Component-level power analysis of *C2* under W_1, illustrating the division into five major components, each comprised of their respective sub-modules, highlighting ATLAS's accuracy in the component level. C2 is an out-of-order CPU design, its main components include the CPU frontend, load-store unit (LSU), data cache (dcache), etc.

Design	ATLAS			Traditional Flow		
	Pre.	Infer	Total	P&R	Simulation	Total
C1	62	3.2	65.2	46392	96	46488
C2	66	4.8	70.8	55327	98	55425
C3	67	4.1	71.1	69624	105	69709
C4	72	4.6	76.6	79836	112	79948
C5	75	5.1	80.1	105742	131	105873
C6	86	4.9	90.9	124860	156	125016
Average	72	4	**76**	80297	116	**80413**

TABLE IV: Runtime (in seconds) comparison for 300 cycles extracted from W_1. Column names represent: Pre. (Data Preprocessing), Infer (Inference), Simulation (Time-based Power Simulation).

In summary, compared to standard design flow based on commercial tools, ATLAS offers three significant advantages: ❶ it eliminates the need for the time-consuming layout process; ❷ it reduces the time required for per-cycle power simulation; ❸ the accuracy of power predictions for post-layout netlists by ATLAS far exceeds that of Gate-Level PTPX® at the early-stage of gate-level netlists.

VII. Conclusion

In this paper, we propose ATLAS, a pioneering framework for fine-grained per-cycle power prediction. ATLAS is the first work that supports both time-based power simulation and general cross-design power modeling. It achieves unprecedented general time-based power modeling based on a customized pre-training and fine-tuning paradigm. When evaluated on the prediction of per-cycle post-layout power based on gate-level netlist, ATLAS achieves a remarkable MAPE of $< 1\%$ for total power estimation with inference speeds that are $> 1000\times$ faster than traditional flow.

VIII. Acknowledgments

This work is supported by National Natural Science Foundation of China (NSFC) 62304192, Hong Kong Research Grants Council (RGC) ECS Grant 26208723, and ACCESS–AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and National Supercomputing Center in Guangzhou Nansha Sub-center for computational resources.

REFERENCES

[1] Synopsys, "PrimePower: RTL to signoff power analysis," 2023. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/signoff/primepower.html

[2] Siemens, "PowerPro® RTL Low-Power," https://eda.sw.siemens.com/en-US/ic/powerpro/, 2021.

[3] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "Primal: Power inference using machine learning," in *DAC*, 2019.

[4] Z. Xie *et al.*, "Apollo: An automated power modeling framework for runtime power introspection in high-volume commercial microprocessors," in *MICRO*, 2021.

[5] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, "How good is your Verilog RTL code? A quick answer from machine learning," in *International Conference on Computer-Aided Design (ICCAD)*, 2022.

[6] C. Xu, C. Kjellqvist, and L. W. Wills, "Sns's not a synthesizer: a deep-learning-based synthesis predictor," in *ISCA*, 2022.

[7] C. Xu, P. Sharma, T. Wang, and L. W. Wills, "Fast, robust and transferable prediction for hardware logic synthesis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 167–179.

[8] W. Fang, Y. Lu, S. Liu, Q. Zhang, C. Xu, L. W. Wills, H. Zhang, and Z. Xie, "Masterrtl: A pre-synthesis ppa estimation framework for any rtl design," in *ICCAD*, 2023.

[9] Y. Du, Z. Guo, X. Jiang, Z. Chai, Y. Zhao, Y. Lin, R. Wang, and R. Huang, "Powpredict: Cross-stage power prediction with circuit-transformation-aware learning," in *DAC*, 2024.

[10] Y. Zhang, H. Ren, and B. Khailany, "Grannite: Graph neural network inference for transferable power estimation," in *DAC*, 2020.

[11] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: learning neural representations of logic gates," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[12] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "Deepgate2: Functionality-aware circuit representation learning," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.

[13] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[14] Q. Wu, W. Zhao, C. Yang, H. Zhang, F. Nie, H. Jiang, Y. Bian, and J. Yan, "Sgformer: Simplifying and empowering transformers for large-graph representations," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[15] A. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 07 2018.

[16] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2019.

[17] Q. Wu, C. Yang, W. Zhao, Y. He, D. Wipf, and J. Yan, "DIFFormer: Scalable (graph) transformers induced by energy constrained diffusion," in *The Eleventh International Conference on Learning Representations*, 2023.

[18] Q. Wu, W. Zhao, Z. Li, D. Wipf, and J. Yan, "Nodeformer: a scalable graph structure learning transformer for node classification," in *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2024.

[19] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2016.

[20] "VCS® functional verification solution," https://www.synopsys.com/verification/simulation/vcs.html, 2021.

[21] "Design Compiler® RTL Synthesis," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-nxt.html, 2021.

[22] *TSMC 40nm LP process technology*, https://www.tsmc.com/ english/dedicatedFoundry/technology/logic/l_40nm, 2008.

VEDA: Efficient LLM Generation Through Voting-based KV Cache Eviction and Dataflow-flexible Accelerator

Zhican Wang[1], Hongxiang Fan[2], Haroon Waris[3], Gang Wang[1], Zhenyu Li[1], Jianfei Jiang[1], Yanan Sun[1], Guanghui He[1*]
[1]*State Key Laboratory of Micro/Nano Engineering Science, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China,*
[2]*Imperial College London, United Kingdom,* [3]*Institute of Space Technology, Pakistan*
{wang_zhican,wangganginjstu, ambitious-lzy, jiangjianfei, sunyanan, guanghui.he}@sjtu.edu.cn
, hongxiangfan@imperial.ac.uk, haroon.waris@ist.edu.pk

***Abstract*—Large Language Models (LLMs) excel in natural language processing tasks but pose significant computational and memory challenges for edge deployment due to their intensive resource demands. This work addresses the efficiency of LLM inference by algorithm-hardware-dataflow tri-optimizations. We propose a novel *voting-based* KV cache eviction algorithm, balancing hardware efficiency and algorithm accuracy by adaptively identifying unimportant kv vectors. From a dataflow perspective, we introduce a *flexible-product* dataflow and a runtime reconfigurable PE array for matrix-vector multiplication. The proposed approach effectively handles the diverse dimensional requirements and solves the challenges of incrementally varying sequence lengths. Additionally, an *element-serial* scheduling scheme is proposed for nonlinear operations, such as softmax and layer normalization (layernorm). Results demonstrate a substantial reduction in latency, accompanied by a significant decrease in hardware complexity, from $O(N)$ to $O(1)$. The proposed solution is realized in a custom-designed accelerator, *VEDA*, which outperforms existing hardware platforms. This research represents a significant advancement in LLM inference on resource-constrained edge devices, facilitating real-time processing, enhancing data privacy, and enabling model customization.**

***Index Terms*—LLM, Algorithm, Dataflow, Accelerator, Hardware**

I. Introduction

Large language models (LLMs) [1], [15] have demonstrated remarkable performance across various natural language processing tasks, such as machine translation, text summarization, dialogue systems, and code generation. This potential has led to a growing trend of deploying private LLMs on edge devices (e.g., personal computers, smartphones, robotics, and intelligent vehicles), enabling real-time inference, data privacy, and model customization. However, the impressive performance of LLMs comes at the cost of significant computational effort and memory storage requirements, exacerbating the disparity between the substantial resource demands of LLMs and the limited resources available on edge devices.

The inference process of LLMs can be summarized in two phases: prefilling and generation. In the prefilling phase, input tokens are provided to the model in parallel, and the model encodes these tokens to kv vectors and concatenates them to form the KV cache. The primary operation in this phase is general-purpose matrix multiplication (GEMM), which is typically compute-bound. This process is similar to that of encoder-only Transformers, such as BERT and ViT [4], [5], with numerous dedicated accelerators being proposed for this task [6], [9], [16]. In the generation phase, tokens are generated sequentially in an auto-regressive manner. Each token attends to the entire KV cache formed in previous steps and extends the KV cache by adding the kv vector of the current token. Since only one token is generated at each step, the primary operation is general-purpose matrix-vector multiplication (GEMV), which is memory-bound. Very few solutions have been proposed to optimize the latter phase. Recent work [19] suggests batch scheduling to enable weight sharing across multiple requests, which packages GEMV operations into GEMM for linear layers. While this is effective in cloud environments with multi-batch processing, edge devices typically use single-batch inference. Additionally, although batch scheduling improves performance for linear layers, it has limited impact on the attention process, as each user has a distinct KV cache, requiring the use of GEMV operations. Therefore, to enhance LLM inference efficiency on edge devices, particularly in the generation phase, this work proposes a comprehensive solution encompassing algorithmic improvements, dataflow optimization, and hardware architecture.

From the algorithmic perspective, numerous studies have shown that the attention process in LLM inference is highly sparse, with sparsity levels approaching 95% [21]. This means that only a small fraction of the KV cache is actively attended to, suggesting the feasibility of locating and maintaining only the pivotal positional kv vectors while discarding the others. Reducing the size of the KV cache not only decreases the computational effort required for critical modules of LLMs (e.g., the attention process) but also reduces off-chip memory access and memory consumption. However, existing algorithms have several limitations. **Some are simple to implement but suffer from accuracy loss.** For example, [18] proposes a sliding window strategy that retains only the earliest and most recent positional KV caches. While simple, it loses accuracy due to the inherent information loss from discarded out-of-window KV caches. [21] introduces a KV cache eviction strategy based on accumulated attention scores, but this approach has its own limitations, which we will discuss in this paper. **Other methods offer better accuracy but involve complex operations,** such as real-time top-k engines [16], complicated formula [2], which are inefficient for deployment. To achieve both hardware efficiency and algorithmic accuracy, we propose a novel *voting-based KV cache eviction* algorithm. Our approach is inspired by the voting process in democratic systems, where each token acts as a "voter" with a unique attention score. This score inherently reflects each token's attention and relation to the kv vectors of prior tokens. Each token "votes" to determine which kv vector should be evicted, and by synthesizing these votes, we can efficiently remove unimportant kv vectors. This algorithm achieves superior accuracy compared to existing methods while maintaining hardware simplicity.

From the dataflow and hardware perspectives, we identify several challenges and propose corresponding solutions. **For GEMV operations:** First, widely ranged dimension size, the dimension size of matrix-vector multiplications varies significantly across different models and even within the same model. For example, in a $(1, k) \times (k, n) = (1, n)$ multiplication, the size of k can range from as small as 128 to as large as 11008 in Llama 7B. Second, dynamic inference process, in the generation phase, the KV cache increases incrementally with each step, meaning that the

This work was supported by the National Science Foundation of China under Grant 92464302. (*Corresponding author: Guanghui He).

dimensions of k or n change dynamically. However, existing GEMV accelerators, such as a multiplier array equipped with an adder tree, are designed with a fixed addition size, which leads to underutilization when k is smaller than the fixed size. For instance, suppose the multiplier array and the adder tree can perform 256− input multiplication and addition in parallel, but if the k is less than 256, the multiplier and adder tree are underutilized. Similarly, if k increases from 256 to 257, it increases one epoch of computation from 1 to 2, and the hardware is extremely underutilized for the second epoch. More importantly, the inability to utilize the sequence variation may make KV cache reduction less helpful to inference speed. Third, memory access irregularity, the transpose operation on the K matrix hinders inference due to irregular memory access patterns. Storing these vectors in the correct format offline is impractical because they are generated online rather than as static weights. While dedicated hardware modules could solve this issue, they would sacrifice efficiency. To mitigate these dataflow inefficiencies, we resort to mathematical interpretations for inspirations. Since GEMV can be interpreted as either inner-product or outer-product calculations mathematically, we propose *flexible-product dataflow* with a runtime reconfigurable GEMV PE array. This dataflow, combined with the dedicated PE array, efficiently handles size variations and eliminates the need for transpose operations, resulting in optimized inference.

For nonlinear operators, especially softmax and layernorm, we face several challenges. First, these operators introduce data dependencies that prevent parallel computation until all required data is available. Second, they occur among GEMV operations, and their latency can block subsequent GEMV computations. Third, the exponentiation and division operations required by these operators are more computationally expensive than multiplications and additions, and placing sufficient hardware resources for these operations can reduce latency but introduces significant overhead. To address these unprecedented challenges, we leverage the *flexible-product* dataflow and further propose a *element-serial scheduling* scheme. This approach not only resolves the latency bottleneck through parallel processing but also reduces the number of high-cost modules from $O(N)$ to $O(1)$ through architectural co-optimization.

Finally, integrating these comprehensive optimizations, we present *VEDA*, a dedicated LLM accelerator. To the best of our knowledge, this is one of the pioneer works to apply algorithm-hardware-dataflow tri-optimizations to accelerate LLM generation. Extensive results demonstrate the efficiency of our solution. In summary, we make the following contributions:

- We propose a novel *voting-based KV cache eviction* algorithm for LLM acceleration, achieving both efficiency and accuracy.
- We provide an insightful analysis of the challenges posed by GEMV and nonlinear operators in LLMs from both hardware and dataflow perspectives.
- We propose a *flexible-product* dataflow with a runtime reconfigurable PE array to address GEMV challenges and introduce *element-serial scheduling* to resolve the nonlinear operator bottleneck.
- We design *VEDA*, a dedicated LLM accelerator that efficiently supports both prefilling and decoding.

II. Background

Figure 1 illustrates the structure of an LLM, with typical dimension sizes annotated. An input sequence, denoted as X_{in}, has a length of l tokens, each with a hidden dimension size of D, represented as (l, D). The LLM computation consists of four main steps: Step 1 - QKV Generation: The input X_{in} is multiplied by W_Q^i, W_K^i, and W_V^i to generate Q^i, K^i, and V^i for $i = 1, 2, ..., H$, where H is the number of attention heads. Q^i, K^i, and V^i all have dimensions of (l, d), with $D = Hd$, where d represents the head dimension. Step 2 - Attention Process: Within each head, Q^i is multiplied by K^{iT} to produce S^i of size (l, l). Notice that due to the causality of LLM, each token only relates to the token before it, the upper triangle of S^i needs to be set $-\infty$. Each element of S^i is scaled to ensure numerical stability, followed by a row-wise softmax operation to generate the attention score matrix S'^i. S'^i is then multiplied by V^i, resulting in the output matrix O^i with dimensions (l, d). Step 3 - Linear Projection: The outputs O^i from all heads are concatenated into a single matrix of size (l, D), which is then multiplied by W_o. Step 4 - Feed-forward Layer: In this step, the input is multiplied by W_1 of size $(D, 4D)$ in FFN1, producing an intermediate result of size $(l, 4D)$, followed by a non-linear activation function like GELU or ReLU. The output is then multiplied by W_2 of size $(4D, D)$ in FFN2, after performing the layernorm (or RMSnorm), it produces the final output X_{next}.

LLM Inference Phase: LLM inference is divided into a prefilling phase and a generation phase. In the prefilling phase, l corresponds to the prompt length P. For example, in Figure 1, the prompt is "DAC is a" with $P = 3$. A key-value (KV) cache is constructed for each token during this phase, ultimately generating the output token "pioneer", which marks the end of the prefilling phase. During the generation phase, l is always one, and activations are reduced to vectors. In the example, "pioneer" serves as input to the LLM, engaging in an attention process with previous KV caches and extending the KV cache with the current key-value vector. The output "conference" is then generated as the next token.

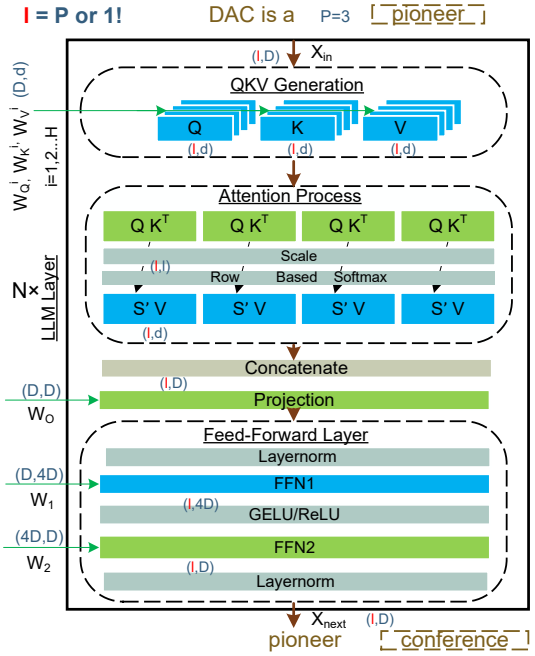


Fig. 1. Detailed LLM structure with sizes denoted. In this paper, matrix-form activations are represented in capital letters(e.g., KV cache), while lowercase letters denote vector-wise activations (e.g., attention score vector s'). Green and blue differentiate the dataflows, which collectively provide the optimal dataflow for LLM, which will be introduced in Section IV-B

III. Voting-based KV Cache Eviction

Pivotal and Unimportant kv Vectors Identification. The key challenge in effective KV cache eviction is that we cannot preemptively determine which KV tensors are pivotal, and which are

unimportant at each new token generation step. Fortunately, some studies suggest that many pivotal kv vectors are likely to remain pivotal, and vice versa [8]. However, defining the importance of kv vectors is non-trivial, and accurately predicting the preferences of newly generated tokens remains challenging. Some research indicates that the most recent tokens are often pivotal, leading to the use of sliding window attention to maintain a fixed KV cache size. However, this approach suffers from accuracy degradation due to the loss of out-of-window tokens. Another commonly adopted method utilizes accumulated attention scores to define the importance of a kv vector. As illustrated in Figure 2 (a), the method accumulates the attention scores vertically across each column, generating an importance vector with a length matching the KV cache. Each element of the importance vector corresponds to the significance of each positional KV. The positional KV with the minimum value in the importance vector is selected for eviction, such as position 6 in the figure. However, several limitations have been identified:

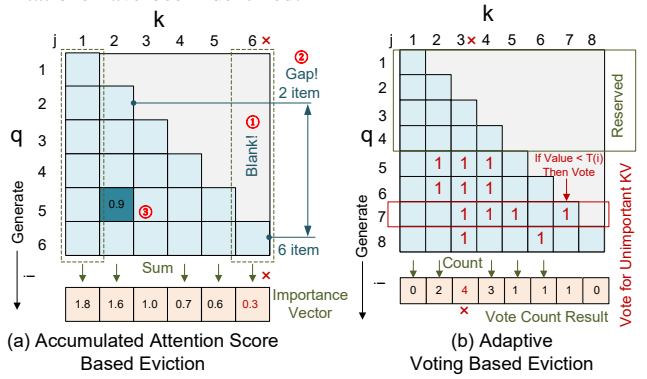


Fig. 2. A comparison between the influential accumulation-based eviction and our voting-based strategy.

Limitations of Existing Method. First, Item Count Bias, as indicated by the red ① in the figure. The number of items summed across different columns varies significantly, with 6 items for the first token, but only 1 for the 6_{th} token due to causal attention, where 5 elements are blank. This discrepancy leads to biased eviction, making recent KV tensors more likely to be pruned, which contradicts both intuition and studies suggesting that recent tokens are more important. Second, Criteria Bias, denoted by ② in the figure. When examining different rows, 2 items in the second row have a total attention score of 1, resulting in a mean of $\frac{1}{2}$, while 6 items in the 6_{th} row yield a mean of $\frac{1}{6}$. Hence, an attention score of $\frac{1}{3}$ may be considered insignificant in the second row but sufficiently important in the 6_{th}. Due to these varying criteria, simply summing values across rows may not yield optimal results. Third, Outlier Bias, indicated by ③. Some tokens have extremely high attention scores, indicating significance at certain positions but potentially less so at others. Nevertheless, the high scores are summed into the importance vector, leading to their consistent consideration as significant.

Overview. To address these challenges, we propose a *voting-based* strategy to differentiate pivotal and unimportant kv vectors. An overview is shown in Figure 2 (b). The method treats each generated token as a voter, fully leveraging the information in each attention score while adaptively adjusting the judgment criteria across positions. Specifically, alongside the attention score vector computation for each token, the algorithm also calculates an adaptive threshold for voting. If an attention score value is below this threshold, it is marked as unimportant, and a value of 1 is set in the corresponding position of the vote count vector. This process updates the vote count vector continuously, and when the cache reaches its predefined size, the KV vector with the highest vote count is evicted. Additionally, a reserved length is maintained at the beginning, during which no voting occurs, aligning with the attention sink phenomenon [18].

Our evaluation shows that the proposed approach addresses the three above-mentioned limitations. First, it tends to preserve recent tokens; for example, in the step shown in the figure, position 7 of KV is voted for at most twice, whereas position 3 can be voted four times. Second, by applying an adaptive threshold relevant to each specific attention score vector, the criteria bias is effectively mitigated. Third, outlier bias is resolved by assigning a uniform weight of 1 instead of relying on absolute values.

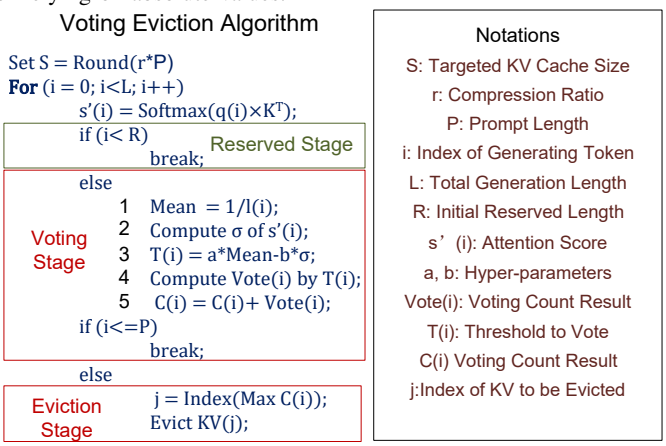


Fig. 3. Voting-based KV cache eviction algorithm with notations on the right.

Details. The complete algorithm is illustrated in Figure 3, with notations on the right, aligned with their order of occurrence. Common operations in LLM inference are omitted for clarity. The algorithm begins with an adjustable parameter r, provided by the user, representing the compression ratio, enabling a trade-off between aggressive compression and precision. The prompt length P is determined once the prompt is input into the model. Based on P and r, the target KV cache size S is calculated. This configuration works well for tasks such as text summarization and question answering, though a fixed target size can also be set for language modeling as the sequence length extends from 1 to the model's maximum capacity.

The algorithm operates through both the prefilling phase and the generation phase. The prefilling phase is divided into the reserved stage and the voting stage. Voting does not occur during the reserved stage, with its size R set to 32, establishing the lower bound for the KV cache. Line 3 of the voting stage defines our adaptive threshold, as a linear combination of the mean and standard deviation of the attention scores. A sparse attention score results in a higher standard deviation, leading to a lower threshold, while a more evenly distributed score has a smaller standard deviation, producing a higher threshold—thus dynamically addressing the criteria bias. The hyper-parameters a and b can be fine-tuned through model-specific calibration without affecting deployment efficiency; values of $a = 1$ and $b = 0.2$ generally prove effective. Notably, the threshold may theoretically drop below zero, in which case the algorithm identifies the minimum attention score and votes accordingly. After processing the entire prompt, the algorithm transitions to the generation phase, which includes voting and eviction stages. Voting continues for each current token, and the algorithm evicts the kv vector with the highest vote count. In cases where multiple positions have identical vote counts, the earliest position is selected for eviction. This strategy significantly accelerates the generation phase due to the reduced cache size.

IV. Flexible Dataflow Optimization

A. Flexible-product Dataflow

Motivations. While dataflow optimization [3] and dataflow-optimized accelerator [17] for the LLM prefilling phase (or encoder-only Transformers) have been studied, a gap remains for the LLM generation phase. As mentioned in Section I, the wide range of dimension sizes, the dynamic inference process, and irregular memory access during transposition hinder efficiency. Notably, the current KV cache compression developed by the algorithm community significantly reduces storage overhead but is less effective in accelerating inference. Specifically, the GEMV of the attention process can be represented as $q \times K^T = s$ with the size $(1,d) \times (d,l) = (1,l)$ and $s' \times V = o$ with $(1,l) \times (l,d) = (1,d)$. With l varying due to token growth during the inference process or KV cache reduction, existing architectures lack the efficiency to handle such variability. To address this, we return to mathematical interpretation for inspiration. **Mathematical Definition:** There are two ways to

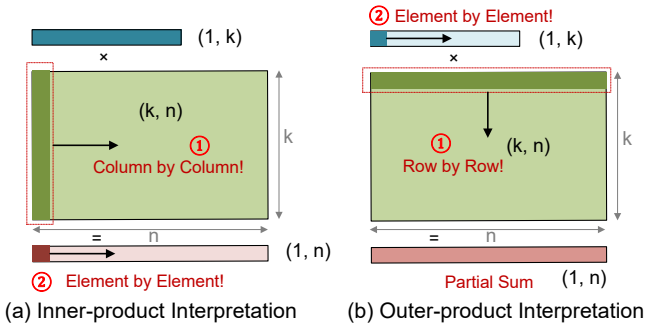


Fig. 4. Two mathematical interpretations of GEMV.

interpret GEMV, as shown in Figure 4. Given a general-purpose notation $(1,k) \times (k,n) = (1,n)$, for the inner-product method, the entire input vector $(1,k)$ is multiplied by the weight matrix (k,n), column by column, generating the output vector $(1,n)$ incrementally. This interpretation is widely used, especially in adder-tree-based implementations. For the outer-product method, one element of the input vector is multiplied element-wise by each row of the weight matrix sequentially, generating a partial output vector $(1,n)$ of the same size as the final output. This partial output is accumulated into the result until all rows of the weight matrix are processed. **Flexible-product Inspiration:** For efficient memory access, denoted by ① in Figure 4, the weight memory access patterns of the two interpretations differ: one operates column-wise, while the other operates row-wise. Interestingly, it inspires us that storing the matrix uniformly, instead of performing transposition, a hybrid implementation on the computation engine can have the equivalent effect to the transpose operation. Specifically, in the attention process, the KV matrix can be uniformly stored in (l,d) format, where l denotes the address, and d elements share the same address. In each step, a $(1,d)$ vector of the K or V matrix can be accessed with a single memory operation, allowing the computation of $q \times K^T = s$ with $(1,d) \times (d,1)$ segment of K^T per step, l steps to realize $(1,d) \times (d,l) = (1,l)$ in an inner-product approach. Similarly, to compute $s' \times V = o$, element-wise multiplication is performed between $(1,1)$ and $(1,d)$ segment of V per step, l steps to realize $(1,l) \times (l,d) = (1,d)$, in an outer-product approach. This strategy avoids the costly transpose operation, ensuring a consistent storage format between the K and V matrices.

For efficient computation, denoted by ② in Figure 4, we observe that either the input or the output is accessed in an element-by-element manner, which offers flexibility in the corresponding dimension, indicating that arbitrary size in that dimension can be temporarily mapped to computation cycles. Specifically, for $q \times K^T = s$ with size $(1,d) \times (d,l) = (1,l)$, let l correspond to n and d correspond to k in Figure 4 (a); the variation in sequence length l is effectively mapped to computation cycles. Similarly, for $s' \times V = o$ with $(1,l) \times (l,d) = (1,d)$, let l and d correspond to k and n, respectively in Figure 4 (b), l variations can be translated into computation cycles variation. In summary, analyzing the problem from both computational and storage perspectives led to a consistent conclusion: using the inner product for $q \times K^T$ and the outer product for $s' \times V$ is the optimal choice, leaving the primary challenge of designing a runtime-reconfigurable computation engine.

Runtime Reconfigurable PE Array: Reconfigurable PE. As illustrated in Figure 5 (a), each PE is governed by a 2-bit control signal, allowing it to perform local partial sum accumulation, transmit partial sums, clear registers, or disable. The multiplexers enable different modes to specify whether the partial sum is accumulated locally or received from other PEs. The dotted section, present only in type-B PEs, indicates that both adder inputs are sourced from other PEs. Outer-Product Configured PE Array. PEs are structured as shown in Figure 5 (b) to perform an outer-product operation. Each PE operates independently with distinct weights, without spatial partial sum transmission, while sharing a broadcast input scalar. The PE array is set as 8×8, a typical finest dimension in modern LLMs. The structure and mapping of the PE are depicted on the right. For $s' \times V$, l is temporally mapped to ensure flexibility, meaning that in each cycle, one scalar from s' is broadcast to the PE array. The parameter d is spatially assigned to each PE, with local accumulation of the partial sum. Inner-Product Configured PE Array. In Figure 5 (c), weights and inputs are independently multiplied within each PE in parallel during each cycle, with partial sums transmitted across PEs. The PE adders are arranged into a two-level adder tree structure. The L1 adder tree receives multiplication results and performs accumulation at the row level, while the L2 adder tree aggregates the results from the L1 adder tree. For $q \times K^T$, l is mapped spatially, allowing one output element of s to be calculated per cycle, with the entire computation completed in l cycles. The parameter d is spatially mapped across the PE array. Hierarchical Adder Tree Implementation. As shown in Figure 5 (d), the red annotations indicate the type and location of each PE in the row. The adder tree is configured with PEs $1,3,5,7$ designated as type-A, where one input to the adder is the local multiplication result, and the second input is the multiplication output from PEs $2,4,6,8$, respectively. PEs $2,4,6,8$ are type-B, with both operands sourced from other PEs. The L2 adder tree follows similar design principles as the L1 adder trees, with the 8_{th} PE functioning as the local adder within a PE, compared to the L1 adder tree.

B. Element-serial Scheduling

Motivations. Nonlinear operations such as softmax and layernorm impose significant data dependencies, as computation cannot proceed until all relevant elements are mathematically ready. These data dependencies impede parallel processing, and the inherent latency delays subsequent operations. As shown in Figure 6, conventional scheduling causes the PE array to remain idle during the computation of softmax or layernorm in the Special Function Unit (SFU). Although increasing the computation units can reduce latency, it leads to higher overhead. Previous research [14], [20] suggests using approximate units to mitigate costs. In contrast, we innovatively address the issue by co-designing dataflow scheduling and architecture, aiming to minimize the number of SFUs while eliminating latency, as illustrated in Figure 6 (a). **Element-serial**

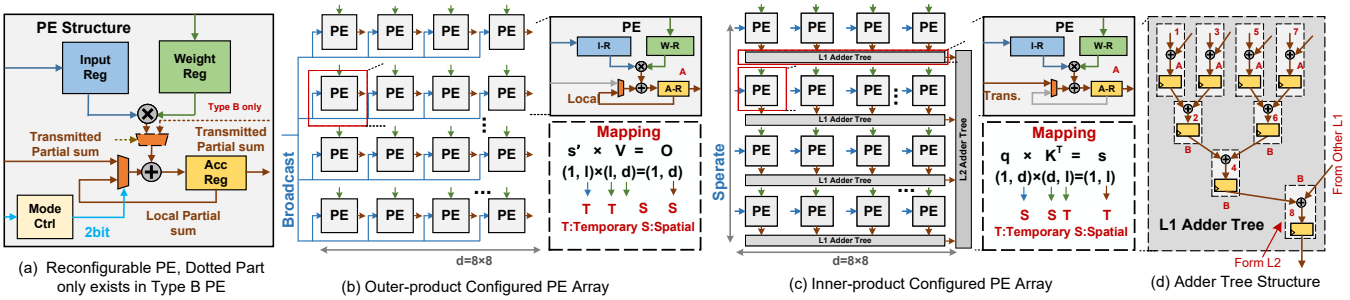


Fig. 5. Runtime reconfigurable hardware architecture.

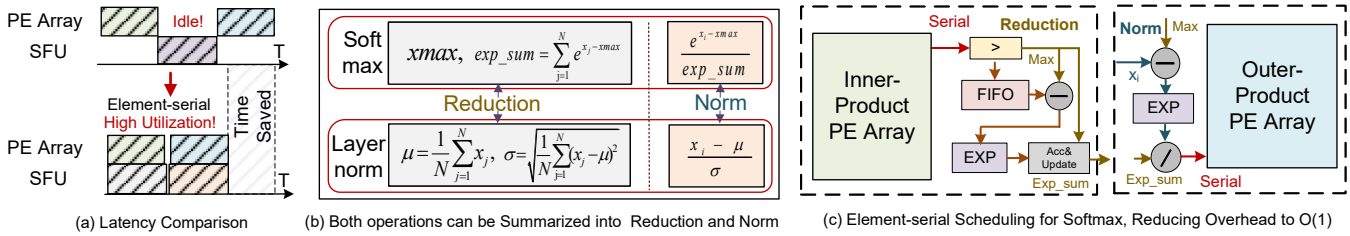


Fig. 6. Element-serial scheduling.

Scheduling. We summarize both softmax and layernorm operations into two stages: reduction and normalization (norm). In the reduction stage, elements are condensed into two numerical characteristics: maximum and exponent summation (exp_sum) for softmax, and mean and standard deviation for layernorm. The normalization stage involves element-wise computations—subtraction, exponentiation, and division for softmax; subtraction and division for layernorm. Drawing from the inspiration of the element-by-element manner discussed in Section IV-A, we propose to schedule the reduction stage element-by-element, running in parallel with PE array computations, which performs operations on the serial output of an inner-product-configured PE array; and scheduling the normalization on the serial input of an outer-product-configured PE array, concurrently with PE array computations as well. **A specific example** of softmax is demonstrated in Figure 6 (c). During the computation of $q \times K = s$ using the inner product in the PE array, the reduction unit in the SFU receives the serial, element-by-element output. It then identifies the maximum value for each tile, updates the maximum, and stores the tile in a FIFO. Once a tile is stored, its maximum is obtained, and this current maximum is used to update the exp_sum, the process is similar to [10]. When the computation of all l elements is completed, the final exp_sum is determined in just a few cycles. For $s' \times V = o$, continuing with the inner product would costly require the concurrent preparation of $(1, l)$ softmax normalization results. Therefore, the outer product configuration facilitates element-by-element serial input into the PE array, enabling parallel processing in SFU. Notably, with a PE array of size d, a single SFU suffices to remove latency in *element-serial scheduling* as shown in Figure 6 (a), and reduce the SFU cost from $O(N)$ to $O(1)$. The same technique applies to layernorm: inner-product implemented GEMV before reduction, the sum of element and the sum of element square is computed simultaneously through the element-serial output to obtain mean and variance; and outer-product implemented GEMV after normalization, the subtraction, and division are performed through element-serial input. Since both softmax and layernorm are positioned between GEMV operations, they effectively define the optimal dataflow for the entire LLM: Please refer to Figure 1, where green indicates the inner product and blue represents the outer product.

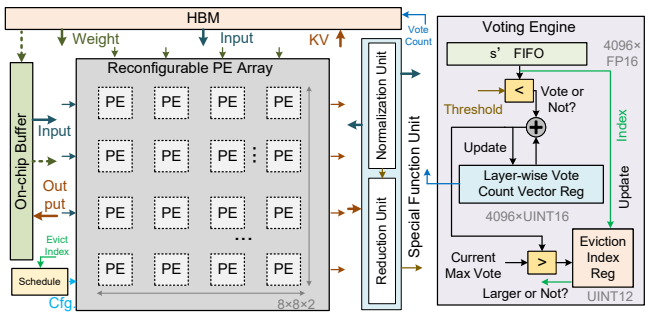


Fig. 7. Hardware architecture of *VEDA*, with blue, green, and red denote input weight and output.

V. Hardware Architecture

Combining dataflow, hardware, and algorithm optimizations, we designed *VEDA*, as depicted in Figure 7. The architecture comprises a PE array, Special Function Unit, voting engine, scheduler, and HBM serving as the off-chip memory. Detailed hardware specifications are provided in Table I. *VEDA* supports both prefilling and generation phases. For storage, weights are fetched to the on-chip buffer and reused across tokens to minimize off-chip memory access for the prefilling phase; however, during the generation phase, they are directly input into the PE array. Thanks to the effective transposition elimination through flexible dataflow, the KV cache is uniformly stored in (l, d), ensuring the high utilization of HBM bandwidth, they need to be stored off-chip, and they can be regarded as weight for the PE array during the generation phase. Additionally, the vote count also needs to be stored and accessed from off-chip to reduce the on-chip overhead. For computation, the GEMM of the prefilling phase is achieved through multiple GEMVs, and weight is reused across GEMVs through buffering. Another significant benefit of GEMV implementation is that, compared with GEMM kernel computation, the effective operations of the attention process are reduced to half due to the causality, our flexible PE array can directly skip computing the upper triangle half of the attention score.

Since the primary modules have been introduced, we will now focus on the voting engine's mechanism. It receives the softmax result s' (which also serves as input for $s'V$) and stores it in the FIFO. Simultaneously, s' is internally input into the reduction unit to compute the mean and standard deviation. Once stored in the FIFO, the reduction unit also finishes the computation of mean and standard deviation, providing a threshold. Each element is then sequentially output from the FIFO to determine voting eligibility, update the previous vote counts, and adjust the registered vote count (hardware

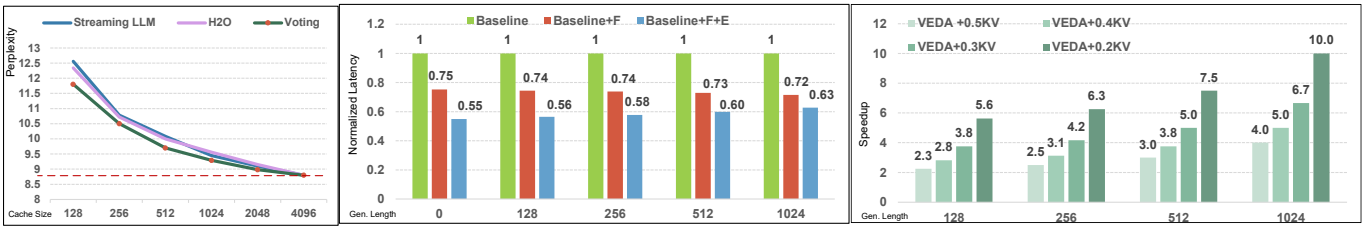


Fig. 8. Experiments. Left: Language modeling evaluation. Center: Ablation study of dataflow optimization. Right: Speedup by voting-based cache eviction.

implemented by buffer) accordingly. During the prefilling phase, no eviction is performed. However, in the generation phase, the vote count is compared with the current maximum vote to update the eviction index. Voting operates layer-wise, meaning that all heads are aggregated and averaged. Once s' across all heads is processed, the final eviction index is established, indicating that a particular address of kv vectors will no longer be accessed. The voting engine consistently operates in parallel, ensuring it does not affect the computations of other modules.

VI. Evaluation

Experiment Setup. Hardware: We utilize SystemVerilog to implement the *VEDA* architecture, using FP16 as the default arithmetic format. The design is synthesized using the Design Compiler under the TSMC 28nm CMOS library with 1GHz to evaluate area and power consumption. A 256GB/s HBM is employed through Ramulator [7] to model the off-chip memory. For the on-chip buffer and FIFO, CACTI [11] is used to estimate energy consumption and area. A cycle-accurate performance model is employed to measure latency, which is cross-validated with RTL simulations. The hardware metrics are detailed in Table I, demonstrating that PE and buffer dominate the area and power, SFU consumes less than 3% due to *element-serial scheduling*, and the voting engine incurs a small 6.5% of overhead. Software: We choose the influential Llama-2 7B model, with a maximum sequence length of 4096, and use the language modeling task—a standard benchmark for assessing LLM performance—to validate our algorithm's feasibility. The primary evaluation metric is perplexity, with lower values indicating better performance.

TABLE I
HARDWARE DETAILS, WITH AREA AND POWER BREAKDOWN OF VEDA

Modules	Parameters	Area/mm^2	Power/mW
PE Array	8*8*2 Reconfigurable PEs	0.493	175.64
Voting Engine	4096*16bit FIFO 4096*16bit Vote Buffer & Others	0.069	26.41
Special Function Unit	2 EXP, 2Divider, 1Sqrt & 2Multiplier and 4Adder 32×16bit FIFO	0.029	13.19
Schedule	System Control & PE Array Config	0.041	11.20
On-chip Buffer	256KB SRAM	0.426	148.82
Off-chip DRAM	HBM, Bandwidth=256GB/s		
Total	TSMC 28nm: Area=1.058mm^2 Power = 375.26mW		

Algorithm Evaluation: We report perplexity results using 1000 samples from the PG-19 dataset [12], a widely-used benchmark for LLM language modeling evaluation. We compare our *voting-based* cache eviction strategy with the notable Streaming LLM [18] and H2O [21] across various cache sizes. As depicted in Figure 8 (left), the *voting-based* eviction consistently outperforms both H2O and Streaming LLM, owing to the voting mechanism's ability to adaptively eliminate three types of bias and select the optimal key-value vector for eviction. Furthermore, even when the KV cache is reduced to 0.1 of the original length (4096), the accuracy loss remains negligible, demonstrating the feasibility of our algorithm.

Ablation Study. For dataflow and reconfigurable hardware gain, to assess the impact of our flexible architecture (F) and *element-serial scheduling* (E), we use a conventional adder-tree-based architecture as the baseline, similar to A^3 [6] Transformer accelerator, in which the softmax operation functions as a pipeline stage with inherent latency. The conventional adder-tree-based design is constrained to a fixed dataflow for inner product computations, lacking flexibility. To ensure a fair comparison, accelerators are configured to have identical peak throughput and the same number of SFUs as *VEDA*. We employ the Llama-2 7B model with a prompt length of 512 for the prefilling phase and generate sequences ranging from 0 to 1024 tokens in the generation phase. The latency of the attention process is averaged over tokens during the generation phase, and depicted in Figure 8 (center), the *flexible-product* approach consistently achieves over a 25% latency reduction due to higher utilization across varying lengths. The *element-serial scheduling* further reduces latency to approximately 60%. For algorithmic gain, we use the same 512-length input and vary the maximum generated sequence length between 128 and 1024, averaging the latency of the attention process over tokens. The baseline in this evaluation is *VEDA* without cache eviction. The *voting-based* approach effectively maintains a constant KV length ($512\times$ ratio) throughout the generation phase, in contrast, the baseline's KV cache grows with each step. This approach achieves a $2.3 \sim 10.0\times$ speedup over the baseline.

Comparison. Table II shows that, compared with SOTA accelerators, thanks to the high utilization powered by flexible dataflow, and low-cost SFU and voting, *VEDA* attains the smallest area cost, and the highest energy efficiency (it remains true after technology scaling [13]). Compared with the SOTA edge GPU, *VEDA* achieves $38.8\times$ energy efficiency. One *VEDA* has a throughput of 18.6 tokens/s. Compensating for the resource discrepancy, $8-VEDA$ will have a $2.86\times$ throughput improvement over GPU. The evaluations demonstrate the feasibility of LLMs edge deployment through *VEDA*.

TABLE II
COMPARISON WITH RELATED ACCELERATORS

Accelerator Comparison	Sanger	Spatten	*VEDA*
Support	Attention	Transformer	**LLM**
Technology[nm]	55	40	**28**
Area[mm^2]	16.9	1.55	**1.06**
Throughput[$GOPS$]	529	360	**245**
Energy Efficiency[$GOPS/W$]	192	382	**653**
End-to-end Comparison with NVIDIA 4090 GPU			
Average Energy Efficiency: 38.8× (Core+Off-chip HBM)			
Average Throughput: 1-*VEDA*: 18.6 tokens/s, 8-*VEDA*:2.86× over GPU			

VII. Conclusion

In conclusion, this work addresses the computational challenges of LLM inference on edge devices through *voting-based* KV cache eviction, *flexible-product* dataflow, *element-serial scheduling*, a pioneering algorithm-hardware-dataflow tri-optimizations. The resulting accelerator, *VEDA*, demonstrates significant performance improvements, enabling efficient LLM inference on resource-constrained platforms, and promoting real-time processing and data privacy.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] M. Adnan, A. Arunkumar, G. Jain, P. Nair, I. Soloveychik, and P. Kamath, "Keyformer: Kv cache reduction through key tokens selection for efficient generative inference," *Proceedings of Machine Learning and Systems*, vol. 6, pp. 114–127, 2024.

[3] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[6] T. J. Ham, S. J. Jung, S. Kim, Y. H. Oh, Y. Park, Y. Song, J.-H. Park, S. Lee, K. Park, J. W. Lee *et al.*, "Aˆ 3: Accelerating attention mechanisms in neural networks with approximation," in *2020 IEEE International Symposium on High Performance Computer Architecture*. IEEE, 2020, pp. 328–341.

[7] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[8] Z. Liu, A. Desai, F. Liao, W. Wang, V. Xie, Z. Xu, A. Kyrillidis, and A. Shrivastava, "Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[9] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[10] M. Milakov and N. Gimelshein, "Online normalizer calculation for softmax," *arXiv preprint arXiv:1805.02867*, 2018.

[11] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[12] J. W. Rae, A. Potapenko, S. M. Jayakumar, and T. P. Lillicrap, "Compressive transformers for long-range sequence modelling," *arXiv preprint arXiv:1911.05507*, 2019.

[13] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[14] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, "Softermax: Hardware/software co-design of an efficient softmax for transformers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 469–474.

[15] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," 2023. [Online]. Available: https://arxiv.org/abs/2307.09288

[16] H. Wang, Z. Zhang, and S. Han, "Spatten: Efficient sparse attention architecture with cascade token and head pruning," in *2021 IEEE International Symposium on High-Performance Computer Architecture*. IEEE, 2021, pp. 97–110.

[17] Z. Wang, G. Wang, H. Jiang, N. Xu, and G. He, "Cosa: Co-operative systolic arrays for multi-head attention mechanism in neural network using hybrid data reuse and fusion methodologies," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[18] G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis, "Efficient streaming language models with attention sinks," *arXiv preprint arXiv:2309.17453*, 2023.

[19] G.-I. Yu, J. S. Jeong, G.-W. Kim, S. Kim, and B.-G. Chun, "Orca: A distributed serving system for {Transformer-Based} generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 521–538.

[20] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: Neural approximation of non-linear operations for efficient transformer inference," 2021. [Online]. Available: https://arxiv.org/abs/2112.02191

[21] Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett *et al.*, "H2o: Heavy-hitter oracle for efficient generative inference of large language models," *Advances in Neural Information Processing Systems*, vol. 36, pp. 34 661–34 710, 2023.

PacTrain: Pruning and Adaptive Sparse Gradient Compression for Efficient Collective Communication in Distributed Deep Learning

Yisu Wang, Ruilong Wu, Xinjiao Li, Dirk Kutscher*
The Hong Kong University of Science and Technology (Guangzhou)
* Corresponding author is Dirk Kutscher, dku@hkust-gz.edu.cn

***Abstract*—Large-scale deep neural networks (DNN) exhibit excellent performance for various tasks. As DNNs and datasets grow, distributed training becomes extremely time-consuming and demands larger clusters. A main bottleneck is the resulting gradient aggregation overhead. While gradient compression and sparse collective communication techniques are commonly employed to alleviate network load, many gradient compression schemes do not achieve acceleration of the training process while also preserving accuracy. This paper introduces PacTrain, a novel framework that accelerates distributed training by combining pruning with sparse gradient compression. Active pruning of the neural network makes the model weights and gradients sparse. By ensuring the global knowledge of the gradient sparsity among all distributed training workers, we can perform lightweight compression communication without harming accuracy. We show that the PacTrain compression scheme achieves a near-optimal compression strategy while remaining compatible with the all-reduce primitive. Experimental evaluations show that PacTrain improves training throughput by 1.25 to 8.72× compared to state-of-the-art compression-enabled systems for representative vision and language models training tasks under bandwidth-constrained conditions.**

I. Introduction

Over the past decade, deep learning (DL) has seen significant growth driven by the development of increasingly complex large-scale models and massive datasets [1]–[3]. These DL models exhibit excellent performance in various computer vision (CV) and neutral language processing (NLP) tasks [4]–[7]. However, their immense scale makes large-scale distributed training (DT) time-consuming and expensive [8].

Large-scale DT incurs a high communication overhead, which is a challenge that spans large and small-scale setups. In large clusters with tens of thousands of GPUs provided by cloud providers (*e.g.*, AWS, Azure, Alibaba Cloud) [9], [10], communication overhead increases linearly as the number of workers increases. Similarly, non-hyperscaler deployments, such as single- or multi-compute clusters in university labs with consumer-grade GPUs, often face communication bottlenecks owing to the constrained and highly variable bandwidths in such systems [8].

Gradient compression (GC) has emerged as a promising solution for mitigating this communication bottleneck by reducing the volume of the exchanged gradient data [11]–[18]. Most compression schemes (*e.g.,* , TopK [15], TernGrad [13], THC [14], and DGC [16]) are lossy and introduce some compression errors, meaning they often compromise both the convergence rate and attainable accuracy, and thus fail to accelerate the training process.

Another potential approach for addressing communication bottlenecks in DDL is sparse collective communication (SCC). In recent years, practitioners in the deep learning community have observed that gradient sparsity often occurs in DNN training and have sought to exploit it [19]–[22]. For instance, OmniReduce [19] introduces a streaming aggregation to transfer sparse gradient blocks, and Zen [20] presents a hashing algorithm that leverages parallel GPU computing to achieve balanced communications. However, these approaches assume that gradients are sparse, which may not always be the case in practice.

Nevertheless, as pointed out by prior studies [11], [18], a common problem of GC and SCC is that most schemes (*e.g.*, DGC, OmniReduce, and Zen) are not compatible with all-reduce. All-reduce [23], [24] is a highly scalable collective communication operation used to aggregate gradients among a set of DT workers. Another widely used gradient collection method is the parameter server (PS) that performs gradient collection on a centralized server [25]. All-reduce is inherently more parallel and scalable than PS based aggregation [11].

To address these challenges, we present PacTrain, a communication-efficient distributed training framework that combines model pruning and adaptive gradient compression. PacTrain can actively change the training gradient distribution through an unstructured pruning technique and leverages an efficient, non-lossy gradient compression scheme that is compatible with the all-reduce primitive to exploit gradient sparsity.

However, transforming this high-level idea into an existing DT framework, such as PyTorch Distributed Data Parallel (DDP) [26] is not straightforward because of the abstracted communication interface. DDP is a widely used framework that accelerates training by distributing model replicas across multiple GPUs, synchronizing gradients, and averaging them to ensure consistency. Most DT frameworks, including DDP, offer communication hook functions that enable developers to customize gradient synchronization algorithms. However, these hooks typically expose only a reformatted one-

TABLE 1: Impact of acceleration methods on training metrics.

Method	Conv. Speed	Compatibility	TTA
PacTrain	✓	✓	✓
THC	✓	✗	✓
Terngrad	✗	✓	?
DGC	✗	✓	?
OmniReduce	✓	✗	✓
Zen	✓	✗	✓

Each method may positively (marked ✓) or negatively (marked ✗) affect performance. The performance of the methods marked with ? will vary according to the choice of the model architecture and training dataset. Compatibility denotes whether the method is compatible with all-reduce operation.

dimensional gradient tensor, where gradient names are removed and gradient orders are rearranged. This abstraction complicates the establishment of a direct correspondence between the reformatted gradient tensor and the original model weights. Re-implementing the distributed training mechanism from scratch risks loses the performance optimizations inherent to the native frameworks. To address this, we propose a mask tracker mechanism that captures sparsity patterns within the reformatted gradient tensor, enabling efficient alignment and customization.

By ensuring that all distributed training workers share global knowledge of gradient sparsity, we can use a pruning mask to reformat sparse gradients into low-dimensional dense gradient tensors. This approach enables lightweight gradient compression without compromising accuracy during the training process. Tab. 1 summarizes how the different methods of GC and SCC impact one or both factors that determine the Time-To-Accuracy (TTA): the iteration time and convergence speed compared with PacTrain. Compared to these methods, PacTrain can provide both fast convergence and compatibility with all-reduce, and it can improve TTA performance.

We conducted extensive experiments on two datasets: CIFAR-10, CIFAR-100 [27], using VGG19 [28], ResNet18, ResNet152 [29], and ViT-Base-16 [6] models. Our evaluations include specific link speeds of 100 Mbps, 500 Mbps, and 1 Gbps, chosen to reflect the WAN bandwidth in small multicluster scenarios. Our evaluation shows that PacTrain effectively minimizes TTA under bandwidth-constrained conditions. Our implementation demonstrates that PacTrain successfully adapts the compression scheme to prune-aware-distributed training scenarios and reduces TTA by up to 8.72× compared with native PyTorch DDP with variable-constrained network bandwidth.

This study makes the following contributions.

Contribution 1. We propose that pruning can be used to enhance the gradient compression, thereby accelerating the process. Traditional pruning after training drops a subset of weights to accelerate inference and save memory size, but we show that the gradient sparsity enforcement (GSE) method based on pruning is crucial for realizing efficient distributed deep learning.

Contribution 2. We propose that all distributed training workers share global knowledge of gradient sparsity and propose a highly efficient, non-lossy compression scheme based on this property that is compatible with all-reduce operations. Our framework can be easily integrated into existing DT frameworks with minimal effort.

Contribution 3. Our comprehensive experiments on large-scale DL training tasks demonstrate that PacTrain outperforms traditional GC methods and achieves state-of-the-art TTA performance.

The rest of this paper is structured as follows: Section II discusses related work. Section III describes the system design and how to change the gradient distribution via pruning, and Section IV provides a detailed evaluation of different machine learning models in a real-world network, and Section V concludes the paper with a summary of our insights and a perspective on future work.

II. Related Work

A. Gradient Aggregation

Data parallelism is widely used for training large-scale DL models. In data-parallel DDL, each GPU worker holds a replica of the DNN model, and the training dataset is partitioned among the workers. Each GPU worker independently performs *forward and backward propagation* on its partition, computing gradients that are aggregated among GPUs.

To mitigate the communication tension with training distributed DDL, many GC and SCC algorithms have been proposed in the community. Techniques such as quantization [14], sparsification [16], and sparse collective communication [19], [20] compress gradients to reduce the amount of data transmitted. However, as previously mentioned, existing GC and SCC methods often lack compatibility with all-reduce collectives. The primary challenge lies in avoiding decompression and recompression at each worker, as this could introduce substantial computational overhead or, in some cases, accumulate compression-induced errors, potentially impacting the training process.

B. Neural Network Pruning

Neural network pruning is a sparsification technique that reduces the memory cost and computation load by removing less important weights. The importance of model weights can be evaluated using various criteria, such as the absolute value of the weights [30] and the L1/L2 norm of the layers [31]. Conventional pruning techniques are mainly applied in a train-prune-fine-tune manner [32] to improve efficiency at the inference stage.

Pruning-Aware-Training. Conventional exploration of pruning during the training reduced training computation cost at each iteration but suffered noticeable performance degradation. Recently, researchers have proposed several policies to prune as early as possible during training without hurting performance such as the Lottery Ticket Hypothesis (LTH) [33], [34] and Early Pruning Indicator [35].

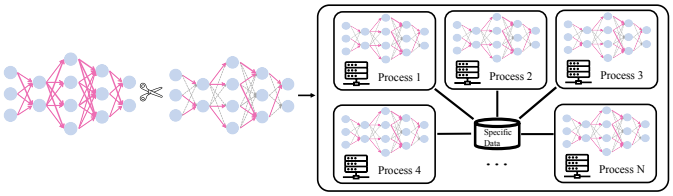


Fig. 1: Pruning and Fine-tune on specific data.

LTH shows that the existence of sparse sub-networks at initialization can achieve almost the same performance as the dense model when trained alone. By identifying these optimal sparse sub-networks after several training epochs, we can prune most of the model weights and reduce both the computation and communication costs during training.

The effectiveness of these strategies is also based on the empirical observation that network representations remain similar after the early phase of training [36]. Once the model stabilizes, only a small portion of the weights in the network is critical to its performance. Therefore, we can perform targeted training without the need to compute the gradients for all neurons [37].

Pruning-Aware-Distributed-Training. When we know in advance that the model weights are sparse, we can leverage this sparsity to customize our specific compression algorithm based on the sparse distribution of the model, which leads us to optimize the precision of the compression techniques and further reduce the gradient synchronization overhead.

Recent studies [38]–[42] accelerate distributed learning through dynamic sparse training. For example, DeepZero [38] employs pruning in zeroth-order optimization to induce gradient sparsity and improve training efficiency. FedMef [39], FedDST [41], and FedTiny [42] perform pruning and regrowth on devices to expedite federated learning (FL). However, pruning typically can achieve only 30% to 80% sparsity, which is insufficient for OmniReduce's compression algorithm which requires approximately 1% sparsity. Using compression schemes like COO (Coordinate List)) also increases heavy computational overhead. Unfortunately, these studies do not detail how to efficiently compress and transmit the sparse gradients.

III. PacTrain Design

This section first outlines the problem setup and describes the system design of PacTrain. We then explain how pruning changes the gradient distribution and how the gradient binarization technique preserves the accuracy impacted by pruning.

A. Problem Setup

Researchers typically start with a model that has been pre-trained on a general-purpose dataset such as ImageNet [43]. Then they will re-train this model on a domain-specific dataset to meet the requirements of a specific task. To speed up training, the dataset is distributed across GPUs, with DDP used to enhance training efficiency. However, due to network bottlenecks, gradient aggregation consumes a significant portion of the total training time.

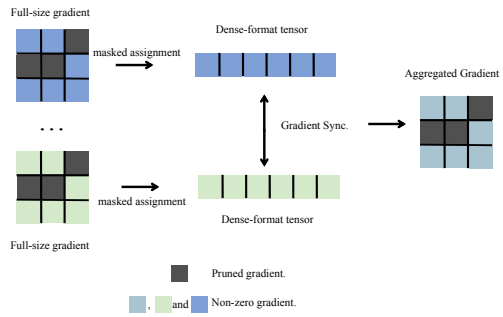


Fig. 2: Acceleration with masked assignment.

Formally, we consider gradient aggregation phase as:

$$X^{t+1} = X^t - \eta \cdot \frac{1}{n} \sum_{i=1}^{n} \Delta(X^t, D_i^t) \tag{1}$$

where X^t represents the model weights at iteration t, n is the number of training workers, η represents the learning rate, D_i^t denotes the dataset partition assigned to the i-th GPU, and $\Delta(X^t, D_i^t)$ are the gradients computed by the i-th GPU based on its local data. Gradient aggregation ensures all GPU workers have the same updated model weights X^{t+1} for the next iteration.

Our objective is to derive a specialized sparse model with a mask m during the training process, aiming to optimize the TTA in DDP. Pruning a pre-trained model can lead to some accuracy loss, which depends on the pruning amount. The challenge in our optimization is how to enhance the communication speed in the gradient aggregation phase while maintaining the final test accuracy.

B. Design Overview

The design of PacTrain is motivated by the limitations of re-training and post-finetuning large models within bottleneck network link scenarios. Previous gradient sparsification techniques [15], [16] have two main limitations: computational overhead and incompatibility with all-reduce operations, limiting efficient gradient synchronization [11].

PacTrain provides less computational overhead and is compatible with all-reduce to preserve accuracy. To that end, we first train the model on a generic, smaller dataset like CIFAR-10 or TinyImageNet [44], or use an existing pre-trained model. The pre-trained model contains important weight information for the task, allowing us to perform pruning on the large model, as shown in Fig. 1. We then conduct distributed training on our specific dataset to accelerate convergence.

Fig. 2 shows how PacTrain implements a local sparsification technique. Instead of selecting TopK gradient coordinates, PacTrain selects non-zero gradient elements based on the local weight magnitudes, which remain the same across the distributed workers. The sparse gradients are reformatted into dense tensors to enable all-reduce synchronization, achieving

significant compression while maintaining computational efficiency.

C. Adaptive Compression Based on Pruning-Aware-Distributed-Training

As illustrated in Fig.1 and Fig.2, the design of PacTrain involves: a pre-trained model is provided to each distributed worker, which then performs task-specific fine-tuning on its subset of data. An unstructured pruning step ensures that the model weights are sparse, optimizing memory and computation efficiency. Subsequently, each worker engages in model update aggregation through the all-reduce operation.

Gradient Sparsity Enforcement (GSE). To reduce non-zero values in the neural network tensor, a pruning operation is necessary. However, during training, these weights can be updated to non-zero values through non-zero gradients. In other words, the distribution of the gradients is different from the distribution of the model weights. We can explicitly set the corresponding gradients to zero for model weights that are zero during the optimization process. This method ensures gradient sparsity. Specifically, this can be expressed using the following GSE formula:

$$\text{Gradient} = (\text{Weight} \neq 0) \odot \text{Gradient} \tag{2}$$

where $\odot$ denotes element-wise multiplication. By repeating this process in every training epoch, the model can progressively adapt to the task while maintaining synchronization in distributed environments.

Our algorithm leverages both sparse gradients and gradient compression during synchronization to significantly reduce communication and computation overhead. Unlike methods that rely solely on gradient magnitudes for compression, this algorithm incorporates the importance of model weights. It effectively eliminates low-magnitude gradients and aligns the gradient update process with both the pruning strategy and intrinsic gradient properties. This design improves data efficiency in constrained networks, accelerates convergence, and enhances the robustness of the model.

The detailed steps of the algorithm are as follows:

1) **Initialization**: Start with a pre-trained model and apply pruning using the specified pruning ratio to create a sparse network.
2) **Training**: During each training epoch, apply GSE to ensure that only gradients corresponding to non-zero weights are updated.
3) **Sparse Mask Update**: Use a mask tracker to monitor the distribution of sparse weights. When the mask stabilizes, compress the gradient updates to include only non-zero elements; otherwise, perform full synchronization to ensure correctness.
4) **Synchronization**: In the distributed environment, communicate only the non-zero elements of the sparse tensor to reduce communication overhead. For unstable masks, use full synchronization to maintain accuracy.

Algorithm 1: Worker Algorithms: Pruning Neural Networks and Send Sparse Tensor during AllReduce Operation

Data: Pre-trained model: The initial neural network model,
Number of epochs n: Total number of training epochs,
$pruning_ratio$: Ratio used for pruning the model,
$Configuration$: Configuration settings for distributed training,
Result: Trained sparse model

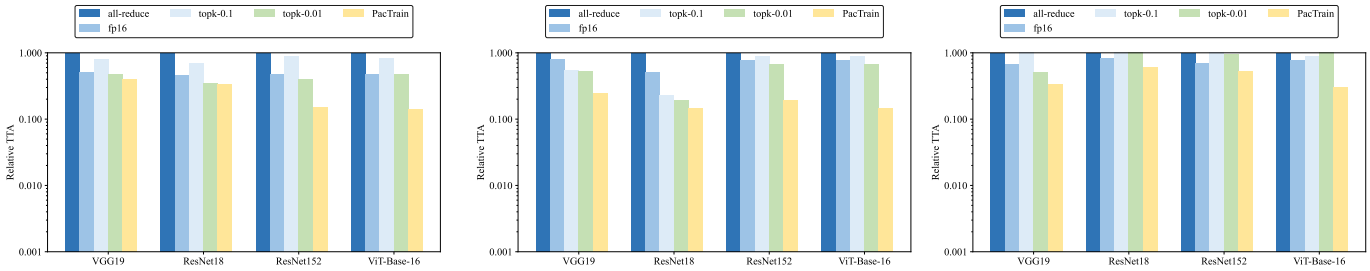
```
Initialize Configuration to find peers and aggregator;
Prune(model, pruning_ratio) ;
for i = 0 to n epochs do
    Perform training on the model;
    Apply GSE to the model gradient;
    foreach update ∈ model_updates do
        Update sparse mask using Mask Tracker;
        if mask is stable then
            compressed_update ← compress(update);
            Communicate only non-zero elements in compressed_update;
        else
            Perform All-reduce on update;
```

Mask Tracker mechanism. For each training epoch, the worker performs training on the sparse model. We introduce Mask Tracker (MT) to capture the sparsity patterns of the reformatted one-dimensional gradient tensor by maintaining the mapping of the gradient and the model weights. Existing DDP frameworks abstract the communication interface to the bucket and gradients are transformed into one-dimensional tensors during synchronization. This transformation removes the original names and order information of the gradients, making it challenging to establish a correspondence between the reformatted gradient tensors and the model weights.

The Mask Tracker monitors the sparsity patterns of the gradients without the need to redesign the underlying distributed training mechanism. By comparing this mask with the one from the previous iteration, we detect any changes in the sparsity pattern. If the mask remains unchanged over multiple iterations, MT deems the sparsity pattern stable. Once a pattern is considered stable, we communicate only the non-zero (non-masked) portions of the gradient tensor during synchronization. Because the MT operates directly on the flattened gradient tensors without modifying their values, it remains fully compatible with native floating-point AllReduce operations. This compatibility allows us to utilize existing optimizations and communication protocols without alteration.

D. Pruning with Gradient Quantization

To further reduce communication overhead while preserving final test accuracy, we introduce ternary gradient quantization



(a) WAN bandwidth = 100 Mbps (b) WAN bandwidth = 500 Mbps (c) WAN bandwidth = 1 Gbps

Fig. 3: End-to-End TTA speedup with different WAN bandwidths (relative to native all-reduce, log scale).

on top of importance-based pruning during DT. Our insight is applying Terngrad quantization to a pruned model can still direct optimization toward a desirable solution, as the pruning step already retains only the most important weight parameters. Specifically, in PacTrain, we use $ternarize(\cdot)$ to randomly quantize the gradient g_t into a ternary vector with values $\in \{-1, 0, +1\}$.

Pruning establishes a fixed mask that permanently zeros out gradients for less important parameters, thus confining the optimization process to a reduced parameter subspace. Although this can potentially slow down convergence or yield suboptimal solutions in non-convex neural networks, ternary quantization introduces controlled randomness. This stochastic element aids the model in escaping local minima.

Convergence Analysis. For each gradient component, the expected value of the quantized gradient is:

$$E(\tilde{g}_t) = E[s \cdot \text{sign}(g_t)] = s \cdot E(\text{sign}(g_t)) \tag{3}$$

where s is a scaling factor ensuring that the quantized gradient remains unbiased in expectation. However, quantization increases variance, which may cause fluctuations in the optimization direction and impact convergence speed.

We employ GraSP [45] to compute pruning scores that reflect each parameter's contribution to the gradient flow. Specifically, the pruning score S is defined as:

$$S = -\theta \odot (H \nabla l(\theta)) \tag{4}$$

where $\nabla l(\theta)$ is the gradient of the loss function, and H is its Hessian matrix. This score allows us to retain parameters critical for maintaining essential gradient directions, thereby enhancing robustness.

In practice, determining the pruning mask based on weight ranking reduces sensitivity to NMSE (Normalized Mean Squared Error, $\text{NMSE}(x, \hat{x}) = \frac{\|x-\hat{x}\|^2}{\|x\|^2}$), further improving robustness in distributed training. This ensures that the remaining parameters retain the key gradients necessary for stable optimization.

IV. Evaluation

PacTrain is designed to operate in arbitrary network settings, ranging from datacenter networks to the public Internet. We implemented and evaluated PacTrain using a virtual testbed hosted in our laboratory.

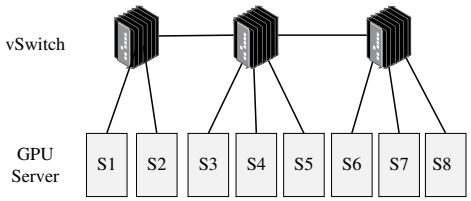


Fig. 4: Evaluation topology.

A. Setup and workloads

We built our test bed using the ESXI virtualization platform with a server equipped with eight A40 GPUs and an Intel Xeon Platinum 8358P processor with 64 CPU cores. To simulate the network environment shown in Fig. 4, we added three virtual switches, each connected to virtual machines with GPUs.

We developed the PacTrain prototype on top of the PyTorch distributed framework [26], implementing distributed machine learning communication, where each GPU performs a learning task and communicates with others using NCCL [46] over TCP for gradient aggregation. In our implementation, we utilized PyTorch's *DistributedDataParallel* communication hook to override the default all-reduce operation, allowing fine-grained control over how gradients are communicated across workers.

Workloads. We evaluated our PacTrain prototype by training popular computer vision models on an eight-worker testbed. The models and datasets used in our evaluation included VGG19 [28], ResNet18, ResNet152 [29] and Vit-Base-16 [6], all of which were trained on the CIFAR-100 and CIFAR-10 dataset for 100 epochs. Unless otherwise noted, we set the per-GPU batch size to 32.

Metrics. We evaluate time-to-accuracy (TTA) across all training tasks under varying bandwidth conditions. We define convergence time as the duration required for the model to reach a target accuracy threshold, signifying that the model has fully converged.

B. Evaluation Scenarios

We conducted experiments under three different scenarios using the topology shown in Fig. 4, creating network perfor-

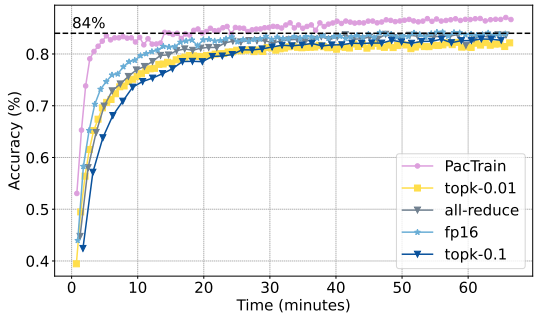


Fig. 5: Time-to-accuracy Comparison of performance for the CIFAR-10 classification task using ResNet152.

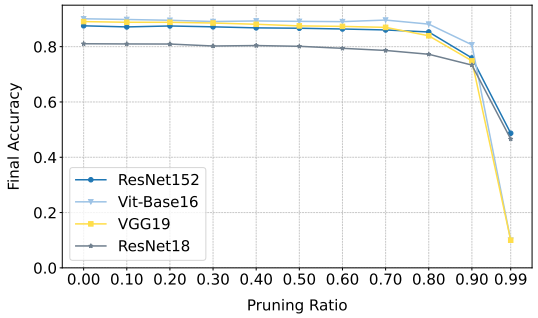


Fig. 6: Impact of Pruning Ratio on Final Accuracy for CIFAR-10 classification task.

mance bottlenecks by adjusting the link bandwidths of the two connections to the switch.

We configured different bottleneck bandwidths of 100 Mbps, 500 Mbps, and 1 Gbps, representing typical network conditions in cloud environments and small datacenters.

C. End-to-End Performance

1) Time-to-accuracy: We evaluated PacTrain against FP16 quantization, TopK compression (0.1 rate and 0.01), and AllReduce for VGG19, ResNet18, ResNet152, and Vit-Base-16 models in bandwidth-constrained environments (100 Mbps, 500 Mbps, and 1 Gbps). As most gradient compression algorithms perform similarly to FP16 [11], we selected FP16 and the commonly used TopK algorithms for comparison. Under extremely low bandwidth conditions, PacTrain demonstrates superior convergence speed compared to FP16 and TopK. Unless otherwise specified, we use a compression scheme with a pruning ratio of 0.5.

Fig. 3 shows the end-to-end training time required to reach the same target accuracy in multiple bandwidth-constrained environments, normalized to that of the native all-reduce operation. The experimental results show that, even under constrained network bandwidth, PacTrain can continue training effectively, leading to faster convergence. Compared with methods that do not involve compression, PacTrain achieves a higher TTA reduction by up to 8.72×. Compared to solutions that support gradient compression and quantization (FP16 and TopK), PacTrain achieves TTA reductions of 1.25-7.05×.

TopK's gradient exchange relies on all-gather, causing network congestion at a 0.1 compression rate, while a 0.01 rate results in too much information loss, making convergence difficult.

2) Iteration and convergence speed: We compare ResNet152 on CIFAR-10 under a 1 Gbps bandwidth due to its representative slow convergence. Fig. 5 shows that for ResNet152, PacTrain reaches the 84% target accuracy 5.64× faster than all-reduce baseline and 3.28× faster than fp16. For models where gradient components are evenly distributed, such as ResNet152, PacTrain tends to achieve a better convergence rate due to its utilization of pruning and quantization strategies.

We observed that model compression algorithms like TopK, which are incompatible with all-reduce, can slow convergence under good network conditions. This is because TopK requires more epochs to reach the target accuracy. When gradient aggregation is not a bottleneck, the benefits of compression are minimal, potentially hindering convergence. TopK relies on all-gather for gradient aggregation, which can lead to accuracy loss with low compression ratios or excessive communication overhead with high compression ratios in distributed training. In contrast, PacTrain, being compatible with all-reduce, ensures communication cost scales are proportional to the pruning ratio.

3) Impact of Pruning Ratio on Final Accuracy: Fig. 6 shows the impact of pruning ratios on final accuracy, based on our testing of VGG19, ResNet18, ResNet152, Vit-Base-16 models on the CIFAR-10 dataset. The relationship between the pruning ratio and final accuracy reflects a trade-off: as the pruning ratio increases, the final accuracy typically decreases due to the removal of important parameters. Although pruning introduces accuracy degradation, the impact remains minimal when the pruning ratio is kept below 80%. Taking ResNet152 as an example, as long as the pruning ratio does not exceed 80%, the final accuracy drops by less than 2%.

V. Conclusion

This paper presents PacTrain, a framework that leverages sparse communication to accelerate pruning-aware distributed learning, offering a novel solution to the communication bottlenecks faced in the distributed training of large models. To demonstrate generalizability, we conducted several experiments to demonstrate that our method can effectively reduce communication costs and maintain the model accuracy across various training scenarios. Testbed experiments using eight GPU workers under various bandwidth-constrained network conditions show that PacTrain can achieve up to an 8.72× improvement.

VI. Acknowledgements

This work is supported by the Guangdong Provincial Project 2023QN10X048, the Guangzhou Key Laboratory on Future Networked Systems 2024A03J0623, and the Guangzhou Science and Technology Project 2023A03J0011.

REFERENCES

[1] R. Thoppilan, D. D. Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, and et al., "Lamda: Language models for dialog applications," 2022. [Online]. Available: https://arxiv.org/abs/2201.08239

[2] H. Chen, Y. Miao, L. Chen, H. Sun, H. Xu, L. Liu, and et al., "Software-defined network assimilation: bridging the last mile towards centralized network configuration management with nassim," ser. SIGCOMM '22, 2022, p. 281–297.

[3] D. Wu, X. Wang, Y. Qiao, Z. Wang, J. Jiang, S. Cui, and F. Wang, "Netllm: Adapting large language models for networking," ser. ACM SIGCOMM '24, 2024, p. 661–678.

[4] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, and et al., "Llama: Open and efficient foundation language models," 2023.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, and et al., "Language models are few-shot learners," ser. NIPS '20. Curran Associates Inc., 2020.

[6] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, and et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *ICLR*, 2021.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016, pp. 779–788.

[8] H. Lim, J. Ye, S. Abdu Jyothi, and D. Han, "Accelerating model training in multi-cluster environments with consumer-grade gpus," ser. ACM SIGCOMM '24, 2024, p. 707–720.

[9] J. Cao, Y. Guan, K. Qian, J. Gao, W. Xiao, J. Dong, and et al., "Crux: Gpu-efficient communication scheduling for deep learning training," ser. ACM SIGCOMM '24. Association for Computing Machinery, 2024.

[10] Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, and et al., "MegaScale: Scaling large language model training to more than 10,000 GPUs," in *NSDI 24*. USENIX Association, Apr. 2024, pp. 745–760.

[11] W. Han, S. Vargaftik, M. Mitzenmacher, B. Karp, and R. B. Basat, "Beyond throughput and compression ratios: Towards high end-to-end utility of gradient compression," 2024. [Online]. Available: https://arxiv.org/abs/2407.01378

[12] R. Ben-Basat, Y. Ben-Itzhak, M. Mitzenmacher, and S. Vargaftik, "Optimal and near-optimal adaptive vector quantization," 2024. [Online]. Available: https://arxiv.org/abs/2402.03158

[13] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li, "Terngrad: Ternary gradients to reduce communication in distributed deep learning," 2017. [Online]. Available: https://arxiv.org/abs/1705.07878

[14] M. Li, R. B. Basat, S. Vargaftik, C. Lao, K. Xu, M. Mitzenmacher, and M. Yu, "Thc: Accelerating distributed deep learning using tensor homomorphic compression," 2024. [Online]. Available: https://arxiv.org/abs/2302.08545

[15] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, M. Palmer, R. Hwa, and S. Riedel, Eds. Association for Computational Linguistics, Sep. 2017, pp. 440–445.

[16] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," 2018, the full paper is available at https://openreview.net/forum?id=rkxoQg-CZ.

[17] T. Vogels, S. P. Karimireddy, and M. Jaggi, "Powersgd: Practical low-rank gradient compression for distributed optimization," 2020. [Online]. Available: https://arxiv.org/abs/1905.13727

[18] H. Xu, C.-Y. Ho, A. M. Abdelmoniem, A. Dutta, E. H. Bergou, K. Karatsenidis, and et al., "Grace: A compressed communication framework for distributed machine learning," in *ICDCS*, 2021.

[19] J. Fei, C.-Y. Ho, A. N. Sahu, M. Canini, and A. Sapio, "Efficient sparse collective communication and its application to accelerate distributed deep learning," ser. SIGCOMM '21. Association for Computing Machinery, 2021, p. 676–691.

[20] Z. Wang, Z. Xu, A. Shrivastava, and T. S. E. Ng, "Zen: Near-optimal sparse tensor synchronization for distributed dnn training," 2023. [Online]. Available: https://arxiv.org/abs/2309.13254

[21] S. Li, Z. Lai, D. Li, Y. Zhang, X. Ye, and Y. Duan, "Embrace: Accelerating sparse communication for distributed training of deep neural networks," ser. ICPP '22. Association for Computing Machinery, 2023.

[22] Z. Wang, H. Lin, Y. Zhu, and T. S. E. Ng, "Hi-speed dnn training with espresso: Unleashing the full potential of gradient compression with near-optimal usage strategies," ser. EuroSys '23. Association for Computing Machinery, 2023, p. 867–882.

[23] Y. Wu, Y. Xu, J. Chen, Z. Wang, Y. Zhang, M. Lentz, and D. Zhuo, "Mccs: A service-based approach to collective communication for multi-tenant cloud," ser. ACM SIGCOMM '24. Association for Computing Machinery, 2024, p. 679–690.

[24] D. D. Sensi, T. Bonato, D. Saam, and T. Hoefler, "Swing: Short-cutting rings for higher bandwidth allreduce," in *NSDI 24*. USENIX Association, Apr. 2024, pp. 1445–1462.

[25] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, and et al., "Scaling distributed machine learning with the parameter server," in *OSDI 14*. USENIX Association, Oct. 2014, pp. 583–598.

[26] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, and et al., "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," ser. ASPLOS '24. Association for Computing Machinery, 2024, p. 929–947.

[27] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep. 0, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[30] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015.

[31] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," 2017.

[32] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," 2017.

[33] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," 2019.

[34] X. Chen, Y. Cheng, S. Wang, Z. Gan, Z. Wang, and J. Liu, "Earlybert: Efficient bert training via early-bird lottery tickets," 2021.

[35] M. Shen, P. Molchanov, H. Yin, and J. M. Alvarez, "When to prune? a policy towards early structural pruning," in *CVPR*, 2022.

[36] S. Kornblith, M. Norouzi, H. Lee, and G. Hinton, "Similarity of neural network representations revisited," 2019. [Online]. Available: https://arxiv.org/abs/1905.00414

[37] J. Jia, J. Liu, P. Ram, Y. Yao, G. Liu, Y. Liu, P. Sharma, and S. Liu, "Model sparsity can simplify machine unlearning," 2024. [Online]. Available: https://arxiv.org/abs/2304.04934

[38] A. Chen, Y. Zhang, J. Jia, J. Diffenderfer, J. Liu, K. Parasyris, Y. Zhang, Z. Zhang, B. Kailkhura, and S. Liu, "Deepzero: Scaling up zeroth-order optimization for deep model training," 2024. [Online]. Available: https://arxiv.org/abs/2310.02025

[39] H. Huang, W. Zhuang, C. Chen, and L. Lyu, "Fedmef: Towards memory-efficient federated dynamic pruning," 2024. [Online]. Available: https://arxiv.org/abs/2403.14737

[40] X. Qiu, J. Fernandez-Marques, P. P. Gusmao, Y. Gao, T. Parcollet, and N. D. Lane, "Zerofl: Efficient on-device training for federated learning with local sparsity," 2022. [Online]. Available: https://arxiv.org/abs/2208.02507

[41] S. Bibikar, H. Vikalo, Z. Wang, and X. Chen, "Federated dynamic sparse training: Computing less, communicating less, yet learning better," 2021. [Online]. Available: https://arxiv.org/abs/2112.09824

[42] H. Huang, L. Zhang, C. Sun, R. Fang, X. Yuan, and D. Wu, "Distributed pruning towards tiny neural networks in federated learning," 2023. [Online]. Available: https://arxiv.org/abs/2212.01977

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," 2015. [Online]. Available: https://arxiv.org/abs/1409.0575

[44] Y. Le and X. S. Yang, "Tiny imagenet visual recognition challenge," 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:16664790

[45] C. Wang, G. Zhang, and R. Grosse, "Picking winning tickets before training by preserving gradient flow," 2020. [Online]. Available: https://arxiv.org/abs/2002.07376

[46] NVIDIA, "Nvidia collective communications library (nccl)," https://developer.nvidia.com/nccl., 2024.

Speculative Decoding for Verilog: Speed and Quality, All in One

Changran Xu[1,3,†], Yi Liu[1,3,†], Yunhao Zhou[1,3], Shan Huang[2,3], Ningyi Xu[2], and Qiang Xu[1,3]
[1]*The Chinese University of Hong Kong*, Shatin, Hong Kong S.A.R.
[2]*Shanghai Jiao Tong University*, Shanghai, China
[3]*National Technology Innovation Center for EDA*, Nanjing, Jiangsu, China

***Abstract*—The rapid advancement of large language models (LLMs) has revolutionized code generation tasks across various programming languages. However, the unique characteristics of programming languages, particularly those like Verilog with specific syntax and lower representation in training datasets, pose significant challenges for conventional tokenization and decoding approaches. In this paper, we introduce a novel application of speculative decoding for Verilog code generation, showing that it can improve both inference speed and output quality, effectively achieving speed and quality all in one.**

Unlike standard LLM tokenization schemes, which often fragment meaningful code structures, our approach aligns decoding stops with syntactically significant tokens, making it easier for models to learn the token distribution. This refinement addresses inherent tokenization issues and enhances the model's ability to capture Verilog's logical constructs more effectively. Our experimental results show that our method achieves up to a 5.05× speedup in Verilog code generation and increases pass@10 functional accuracy on RTLLM by up to 17.19% compared to conventional training strategies. These findings highlight speculative decoding as a promising approach to bridge the quality gap in code generation for specialized programming languages.

***Index Terms*—Verilog code generation, speculative decoding**

I. Introduction

The rapid advancement in large language models (LLMs) has transformed code generation across various programming languages [1]–[4]. These models, driven by advanced pre-training techniques, have achieved notable success in generating syntactically and semantically correct code for widely used languages like Python and C++. However, their application to specialized languages such as Verilog, which is critical for hardware design and verification, remains limited due to the unique challenges posed by Verilog's syntax intricacies and its underrepresentation in training datasets.

Unlike natural languages, programming languages like Verilog are governed by strict syntactic rules, where even minor deviations can lead to significant compilation failures or functional errors. Traditional tokenization methods, such as Byte Pair Encoding (BPE) [5], fragment meaningful code structures into subword units, obscuring the logical relationships inherent in Verilog. Consequently, models often struggle to accurately capture the syntax and semantics of Verilog code. Grammar-based approaches attempt to mitigate this issue by representing code as sequences of grammar rules [6]–[11]. While effective for smaller models and datasets, these methods face scalability issues when applied to LLMs, due to vocabulary explosion and distribution shifts. Moreover, these methods remain largely unexplored for underrepresented languages like Verilog, which presents an additional layer of complexity.

To address these challenges, we propose a novel application of speculative decoding [12]–[15] for Verilog code generation. While speculative decoding has traditionally been employed to accelerate LLM inference by predicting multiple tokens simultaneously to reduce latency, our key insight is that *speculative decoding, when carefully aligned with syntactically significant tokens, can also improve the quality of code generation*. By structuring decoding stops around meaningful fragments in Verilog, such as identifiers, keywords, and operators, we enable the model to better capture the logical and structural relationships inherent in the language. This alignment simplifies the learning of Verilog's token distribution, mitigating issues caused by conventional tokenization schemes and enhancing the syntactic and functional correctness of the generated code. Specifically, the main contributions of our work include:

- To the best of our knowledge, this is the first application of speculative decoding aimed not only at improving inference speed but also enhancing output quality.
- We propose a simple yet effective method for identifying syntactically significant tokens in Verilog using abstract syntax trees (ASTs) and modify the speculative decoding scheme by incorporating syntax-enriched labels to align decoding stops with these tokens.
- Our decoding scheme further unlocks the potential of the original MEDUSA method [15] by training it with dynamic labels in a flexible manner, increasing the number of effective heads and further accelerating the speedup.

We conduct experiments with two models, CodeT5p-220m-bimodal [2] and CodeLlama-7b [3]. The results show that our method achieves up to a 5.05× speedup in Verilog code generation and increases pass@10 functional accuracy on RTLLM [16] by up to 17.19% compared to conventional training strategies. Impressively,

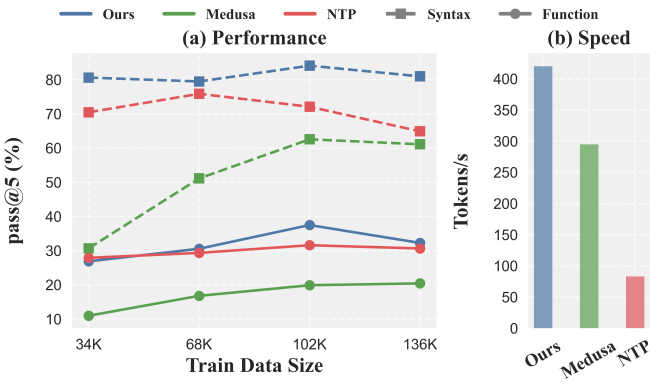


Fig. 1. A brief comparison of the performance and speed of our method against the MEDUSA method and the conventional next token prediction (NTP) approach. The experiments are conducted using the CodeLlama-7b model, with performance metrics evaluated on the RTLLM benchmark.

† These authors contributed equally.

our approach also accelerates model inference by 1.42–2.29× over the original MEDUSA method, highlighting its ability to deliver both speed and quality simultaneously.

II. RELATED WORK

A. LLM for Verilog Generation

Recent advancements in LLMs have shown significant potential for automating the generation of Verilog code from high-level prompts [17]–[21], demonstrating their ability to address the design challenges faced by hardware developers. While some works [22]–[27] have fine-tuned open-source models to improve Verilog code generation, they often treat Verilog code as if it were natural language, using tokenizers originally designed for natural language models without modification. This approach often leads to the fragmentation of meaningful Verilog code structures, which hinders the model's ability to accurately capture the syntactic structure of Verilog and generate syntactically correct code. Furthermore, the scarcity of Verilog code datasets exacerbates the challenge of effectively learning Verilog's syntax. To address this issue, we propose a novel application of speculative decoding for Verilog code generation. By integrating conventional tokenization with Verilog's syntactical structure, our method allows the model to generate more accurate Verilog code more efficiently.

B. Syntax-Aware Tokenization for Code

Driven by recent developments in LLMs, a variety of pretrained models for programming languages have emerged [1]–[4], [22]–[27]. However, most of these models represent code as token sequences using the BPE algorithm for pre-training, which often hinders their ability to capture code's syntactic structure and fails to ensure the syntactic correctness of the generated output. To address this limitation, some models [28]–[30] have attempted to explicitly incorporate syntax by representing code as AST sequences, which are obtained by traversing the AST in pre-order and recording the symbol of each node. However, to maintain the tree structure in the node sequences, these approaches introduce additional nodes, which significantly increase sequence length and GPU memory usage, potentially compromising model performance. Furthermore, these models only leverage the AST representation in the encoder and do not ensure syntactic correctness during code generation. Meanwhile, several non-pretrained code generation models [6]–[10] have used grammar rule sequences to represent code. Specifically, they traverse the AST in pre-order and record the grammar rules used to expand each non-terminal. The integration of grammar rules has been shown to improve performance. However, these models have only been tested in non-pretrained settings and do not involve LLMs. When applied to pre-training with large code corpora, these models face a significant challenge due to the *big vocabulary* problem, which arises from the large number of user-defined identifiers and could degrade the model's performance. In contrast, BPE is designed to find a relatively small set of subtokens whose concatenations could represent a large token set. To integrate BPE with grammars, GrammarT5 [11] uses a variant of grammar rule sequence to represent code, called Tokenized Grammar Rule Sequence (TGRS). Despite this innovation, GrammarT5 still faces generalization issues when applied to token sequences and has only demonstrated marginal improvements in smaller models, leaving its effectiveness on larger models unexplored. Moreover, all these methods have ignored programming languages like Verilog, which are underrepresented in training datasets, making these challenges even more significant. To overcome these limitations, we propose a more natural way to integrates BPE with grammar without significantly altering the distribution of code tokens. By introducing a novel application of speculative decoding for Verilog, our method can enhance both the speed and quality of Verilog code generation.

C. Speculative Decoding

As the size of language models continues to grow, inference latency has become a critical challenge for practical applications. To alleviate this issue, speculative decoding [12]–[14] has been proposed to reduce the number of decoding steps. Traditional speculative decoding employs a smaller draft model to generate an initial token sequence, which the original, larger model then refines to produce an acceptable continuation. However, it remains challenging to acquire and maintain a separate draft model. To address this, MEDUSA [15] introduces additional decoding heads that predict multiple tokens concurrently, enabling seamlessly integration into existing LLM systems. MEDUSA offers two fine-tuning approaches tailored to different use cases: (1) MEDUSA-1, which is fine-tuned on a frozen LLM, ensuring lossless acceleration; and (2) MEDUSA-2, which is jointly fine-tuned with the backbone LLM, achieving higher prediction accuracy and greater speedups. While originally developed for accelerating LLM inference, we find that speculative decoding can also preserve the syntactic structure of code by modifying its decoding mechanisms. By aligning decoding stops with syntactically significant tokens, speculative decoding enhances both inference speed and output quality in Verilog code generation. Fig. 1 presents a brief comparison of the performance and speed of our method against the original MEDUSA method and the conventional next token prediction (NTP) approach.

III. METHODOLOGY

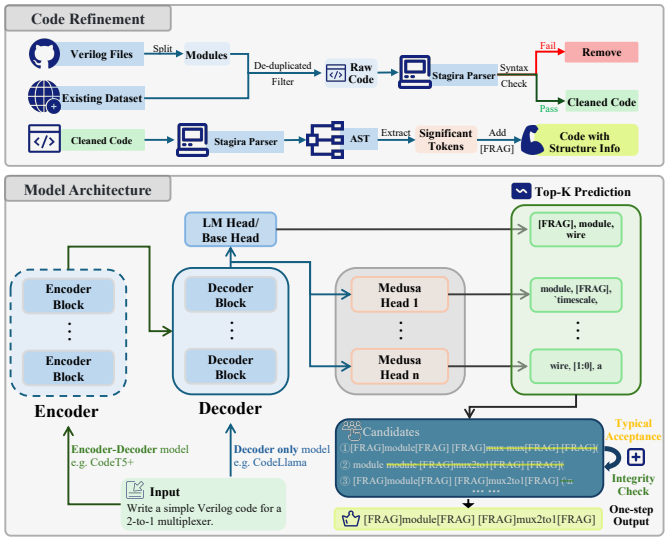


Fig. 2. The overview of the data refinement process and model architecture.

A. Dataset Construction

Our dataset comprises `.v` files collected from Github using `Verilog` as the search keyword. Each file is segmented into functional Verilog modules, and duplicates are removed using MinHash and Jaccard similarity metrics [31]. We also filter out files lacking complete `module` and `endmodule` structures or primarily consisting of comments. Additionally, we supplement our dataset with data from the open-source projects MG-Verilog [25] and RTLCoder [23], resulting a total of 13,6134 data items. To ensure the quality of the

dataset, we apply the Stagira Verilog parser [32] to perform syntax checks on all code samples and retain only those that pass (*i.e.*, cleaned code). For these cleaned code samples, we use the parser to generate their corresponding ASTs, from which the syntactically significant tokens are extracted. The entire code refinement process is illustrated in Fig. 2. For the cleaned code collected from Github, we leverage GPT-4 [33] to generate functional descriptions, while the data from MG-Verilog and RTLCoder already include code summaries.

B. Model Architecture

In this work, we evaluate our approach using two base models: CodeT5p-220m-bimodal [2] and CodeLlama-7b [3], which differ in architecture and size. Specifically, CodeT5p employs an encoder-decoder architecture while CodeLlama is a decoder-only model. As shown in Fig. 2, we augment the base models with additional heads attached to their last hidden states to predict multiple tokens concurrently, following the original MEDUSA method [15]. To improve both inference speed and the output quality of Verilog code generation, we modify MEDUSA's decoding mechanism to align decoding stops with syntactically significant tokens, which ensures that the model maintains a complete syntactic structure at each decoding step, enabling it to explicitly capture the Verilog syntax. For detailed explanations, please refer to Section III-C.

During the decoding process, given that the model is currently at position t, the base model will predict the token at $t+1$, while the i-th head will predict the token at $t+i+1$. Each token generated by the heads is first evaluated by the typical acceptance strategy [15]:

$$p_{\text{base}}(x_{t+i+1}|x_1, x_2, \ldots, x_{t+i}) > \min\left(\epsilon, \delta \exp\left(-H\left(p_{\text{base}}(\cdot|x_1, x_2, \ldots, x_{t+i})\right)\right)\right) \quad (1)$$

where p_{base} represents the prediction probability of the base model, $H(\cdot)$ denotes the entropy function, and ϵ, δ are hyper-parameters. A token is accepted only if (1) holds for it and all preceding tokens. At this decoding step, suppose only the tokens predicted by the first u heads are accepted. These tokens, spanning positions $t+1$ to $t+u+1$, are then re-evaluated to ensure they form a complete syntactic structure. Any extraneous tokens that compromise the integrity of the code fragment are discarded. For instance, if the tokens from $t+1$ to $t+v+1$ (where $v < u$) already constitute a complete code fragment, such as an identifier or keyword, the outputs from the remaining heads, corresponding to $t+v+2$ to $t+u+1$, are discarded. During decoding, we maintain several candidates comprising the top-k predictions from the base model and additional heads. The final prediction for the current step is the *longest accepted prefix* among all candidates.

C. Syntax-Enriched Labels

In this section, we explain how syntactically significant tokens are identified and present a novel method to construct syntax-enriched labels for incorporating syntax into the speculative decoding process. To identify syntactically significant tokens, we first parse Verilog code into ASTs, from which we extract leaf nodes and non-terminal nodes that contains critical information as keywords. Additionally, we supplement these keywords with commonly used Verilog constructs, such as `negedge` and `endmodule`. Together, these keywords form the syntactically significant tokens. We then use regular expressions (regex) to segment the code into meaningful fragments that maintain syntax integrity based on these keywords. At each segmentation point, we insert a special token `[FRAG]` to prepare for constructing syntax-enriched labels. Fig. 3 illustrates this process with an example.

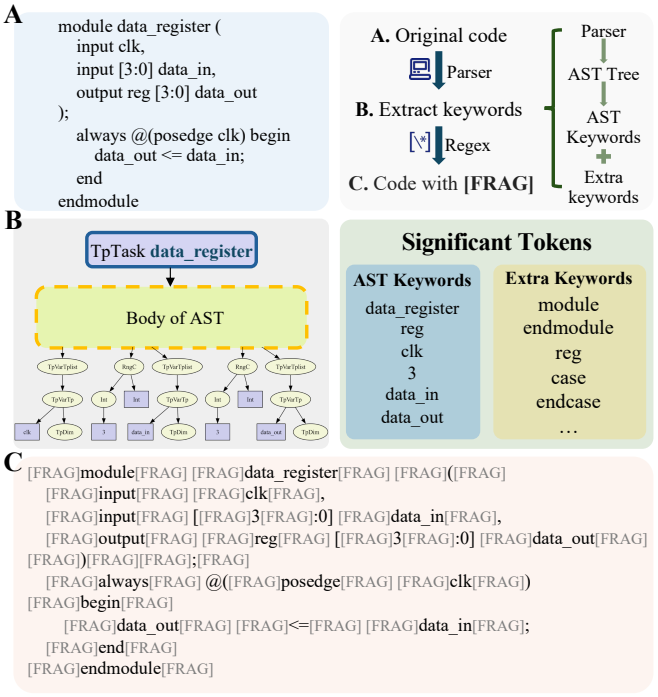


Fig. 3. An example demonstrating the identification and extraction of syntactically significant tokens from Verilog code.

To incorporate syntax information into the speculative decoding process, we modify the decoding mechanism by introducing syntax-enriched labels. Specifically, the label for the base model, denoted as L_0 with a length of $sequence_length$, is the tokenized version of the Verilog code filled with `[FRAG]` tokens. And the label for the i-th head is first derived by left-shifting the base model's label by i, resulting in $L_i = L_0[i:]$. `[PAD]` tokens are then appended to ensure that the head labels align in length with the base model's label. The labels for all heads are then concatenated with the base model's label, forming the combined label $Labels$ with size $(num_heads+1) \times sequence_length$. As shown in Fig. 4, for each sequence position, we identify the position of the last `[FRAG]` token along the decoding heads. Labels beyond this position are replaced with `[IGNORE]` tokens, excluding them from loss computation. This ensures that, for any given sequence position s, the base model's label and the corresponding head labels, $Labels[:, s]$, represent syntactically meaningful fragments by excluding incomplete information. The syntax-enriched labels align decoding stops with syntactically significant tokens, enabling the model to better capture the syntactic structure of Verilog code. Additionally, the progressive increase of the proportion of `[IGNORE]` tokens in the labels of later heads reduces their prediction difficulty, improving the model's training efficiency and allowing us to train more robust heads than the original MEDUSA method, which further accelerates inference. To optimize performance, we design a parallel algorithm that fully parallelizes the procedure, significantly speeding up the process and minimizing computational overhead during training. The pseudo-code for this algorithm is provided in Fig. 4.

IV. EXPERIMENTS

A. Experimental Setup

We select CodeLlama-7b-Instruct-hf [3] (CodeLlama) and CodeT5p-220m-bimodal [2] (CodeT5p) as our base models. For

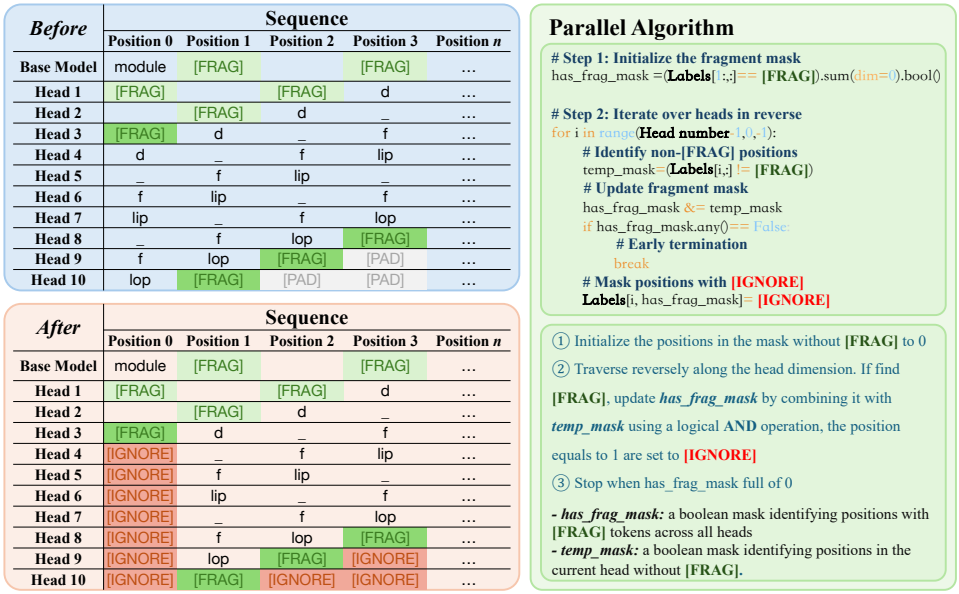


Fig. 4. The construction of syntax-enriched labels for aligning decoding stops with syntactically significant tokens. The top-left panel illustrates the initial labels of Verilog code filled with `[FRAG]` tokens, while the bottom-left panel depicts the final syntax-enriched labels used for training. The right panel presents the parallel algorithm for accelerating the label construction process.

each base model, we append 10 additional heads and fine-tune it for Verilog code generation, comparing our syntax-enriched training method (Ours) with the original MEDUSA-2 method [15] (Medusa) as one baseline. We also consider the conventional NTP scheme (NTP) as another baseline.

1) Training Data: The training data is constructed using the dataset introduced in Section III-A. Specifically, it is formatted into the Alpaca style [34], with the natural language description as input and the corresponding Verilog code as output, resulting in 136K samples for fine-tuning. Due to the 2048-token limit of the CodeT5p model, we exclude examples exceeding this threshold, resulting in 128K training samples for models based on the CodeT5p architecture. The data used for the two baseline methods is identical to ours, except that it does not incorporate the syntax-enriched labels. To evaluate model performance across varying training data sizes, we fine-tune the models not only on the full dataset but also on random subsets comprising 1/4, 1/2, and 3/4 of the original data.

2) Model Training: All model training is conducted on four NVIDIA A800-SXM4-80GB GPUs using the Distributed Data Parallel (DDP) module from PyTorch.

We utilize the Axolotl framework [35] to fine-tune models based on CodeLlama with QLoRA [36], using consistent hyperparameters: a LoRA adapter rank of 32, α set to 16, and a dropout rate of 0.05. Additional hyperparameters are also kept constant, including a micro-batch size of 1, an 8-bit AdamW optimizer with a cosine learning rate scheduler, an initial learning rate of $5e^{-4}$ for the base model, a warmup period of 40 steps, and a maximum sequence length of 8192 tokens. To improve training throughput, we employ the multipack method, which combines multiple short sequences into a single batch. For models based on CodeT5p, we fine-tune directly using a batch size of 2, the AdamW optimizer, a learning rate of $5e^{-4}$ for the base model, a warmup ratio of 0.1, and a maximum sequence length of 2048 tokens.

For models with additional heads, the learning rate for the decoding heads is set to four times that of the base model. The overall loss for these multi-head models is computed using the method proposed in MEDUSA [15]:

$$\text{Loss} = Loss_{base} + \lambda \cdot \sum_{i=1}^{n} (Loss_{\text{head}_i} \cdot \gamma^i) \tag{2}$$

In this equation, λ is a dynamic weighting factor that adjusts the influence of the heads' losses on the overall loss. During training, λ follows a sine growth pattern, increasing from 0 to 0.2 as training progresses. The parameter γ, set to 0.8 in our experiments, serves as a decay coefficient to differentially weight each head's loss. Here, n represents the number of heads, fixed at 10 in our experiments.

3) Model Inference: During inference, all models are loaded in float16 format. CodeLlama-based models are configured with a maximum token length of 8192, while CodeT5p-based models are limited to 2048 tokens. For speed evaluation, each prompt is processed using two decoding methods: greedy decoding and sampling decoding at a temperature of 0.8. For quality evaluation of the generated Verilog code, 20 responses are sampled per prompt at temperatures of 0.2, 0.4, 0.6, and 0.8. The final result for each prompt is determined by selecting the output with the highest accuracy across all temperatures.

B. Evaluation Benchmark and Metric

We use RTLLM [16] and VGen [22] as evaluation benchmarks. Specifically, we employ low-level prompts from VGen that align with the format of our training data. These prompts describe the module's function along with its header, including the module name and the input and output types, which are the most challenging cases.

1) Speed Evaluation: In addition to prompts from RTLLM and VGen, we utilize GPT-4 to generate additional prompts for the Verilog code generation task based on the input prompt formats of RTLLM and VGen, aiming to enhance testing accuracy by increasing the diversity of prompts used during evaluation. Ultimately, the generation speed of the fine-tuned models is assessed using a total of 575 input prompts. For each prompt, the model generate outputs using both greedy decoding and sampling decoding methods, with

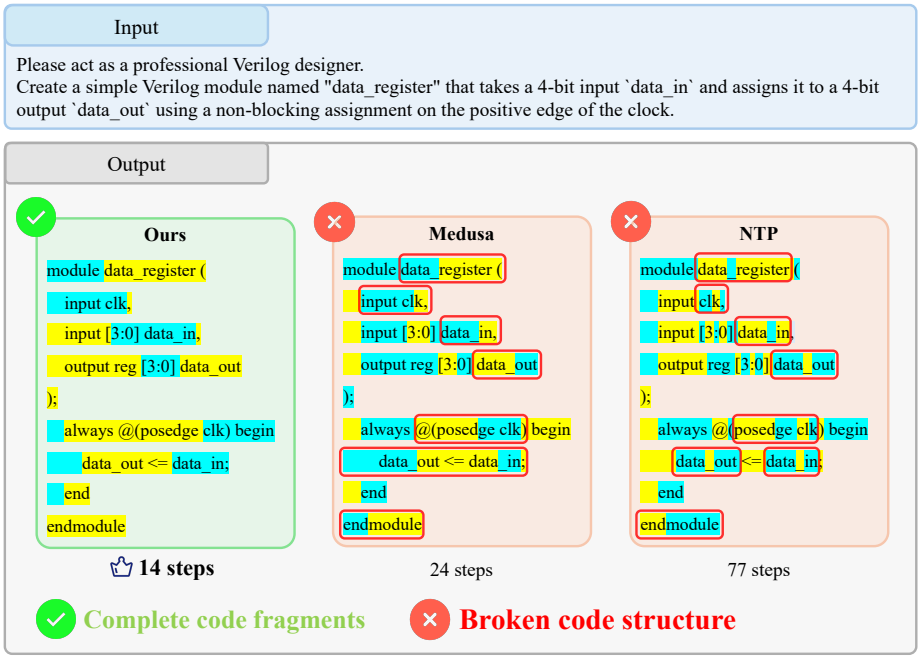


Fig. 5. The comparison of the decoding processes for a specific example using our method, Medusa, and NTP. Remarkably, our method generates the output in significantly fewer steps while preserving the integrity of syntactic structure at each decoding step.

the inference time recorded separately for each method. We then calculate the model's generation speed using the following formula:

$$Speed = \frac{1}{n}\sum_{i=1}^{n}\frac{\text{Output Token Length}_i}{\text{Inference Time}_i} \tag{3}$$

where n represents the total number of outputs generated (*i.e.*, 575 outputs for each decoding method, resulting in 575×2 for the two decoding strategies).

The speedup of each fine-tuned model is calculated relative to its counterpart fine-tuned with NTP, which serves as the baseline, and is defined as follows:

$$Speedup = \frac{\text{Speed of Fine-tuned Model}}{\text{Speed of Fine-tuned Model with NTP}} \tag{4}$$

2) Quality Evaluation: For syntax evaluation, a design is considered syntactically correct if both the design and its testbench successfully compile together using iverilog [37]. For functionality evaluation, a design is deemed functionally correct if its output matches the expected results for all testbench-provided stimuli. We use the $pass@k$ metric, introduced in VerilogEval [20], to evaluate both the functional and syntactic correctness of Verilog code generated by LLMs. For a specific prompt i, $pass@k$ reflects the likelihood of at least one correct solution among k randomly selected attempts:

$$pass@k = \mathop{\mathbb{E}}_{prompt_i}\left[\frac{1-\binom{n-c}{k}}{\binom{n}{k}}\right] \tag{5}$$

Here, n denotes the total number of samples generated by the model for each prompt, and c represents the count of outputs that pass the functional check. To ensure a comprehensive assessment while maintaining experimental efficiency, we set $n = 20$ for all prompts and evaluate k at values 1, 5, and 10.

We also use an additional evaluation criterion, *Pass Rate*, to further assess model performance. For each example in the benchmark, the model is considered successful if any of the 20 generated attempts

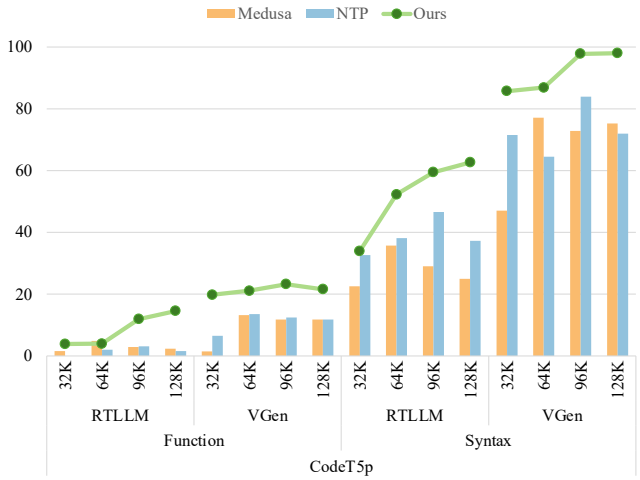


Fig. 6. The comparison of pass@5 output quality between our method and baselines using the CodeT5p architecture.

passes validation. Given m successful cases, the overall *Pass Rate* is calculated as:

$$Pass\ Rate = \frac{m}{len(\text{Benchmark})} \tag{6}$$

C. Experimental Result

Table II highlights the superior inference speed achieved by models fine-tuned with our method compared to those trained using Medusa and NTP. The most significant improvement is observed with the CodeLlama model, which achieves a speed of 420.13 tokens per second, corresponding to a 5.05× speedup over the NTP baseline. For CodeT5p, our method delivers a 2.66× speedup, outperforming the model trained with Medusa, which achieves a 1.16× speedup. To

TABLE I
EVALUATION RESULTS FOR THE QUALITY OF GENERATED VERILOG CODE

Test	Model	Data Size	Benchmark	pass@1 (%)			pass@5 (%)			pass@10 (%)			Pass Rate (%)		
				Ours	Medusa	NTP	Ours	Medusa	NTP	Ours	Medusa	NTP	Ours	Medusa	NTP
Function	CodeLlama	34K	RTLLM	16.21	4.66	16.72	26.87	10.96	27.89	33.85	13.07	32.62	41.38	13.79	37.93
			VGen	30.59	27.06	29.12	49.17	34.48	47.65	56.78	38.04	55.16	64.71	41.18	58.82
		68K	RTLLM	18.28	13.28	18.79	30.55	16.78	29.34	35.52	19.87	34.24	41.38	24.14	37.93
			VGen	32.06	25.59	24.12	47.01	32.88	44.12	53.61	36.59	51.81	58.82	41.18	58.82
		102K	RTLLM	20.52	13.10	17.07	**37.48**	19.88	31.58	**46.29**	23.50	38.25	**55.17**	27.59	41.38
			VGen	31.18	26.76	32.35	53.42	34.36	53.82	63.17	36.81	61.72	70.59	41.18	64.71
		136K	RTLLM	**21.55**	13.79	12.24	32.25	20.42	30.63	38.56	25.04	37.55	44.83	34.48	37.93
			VGen	**34.12**	22.35	31.76	**55.47**	32.78	51.64	**65.51**	35.01	63.21	**76.47**	35.29	**76.47**
	CodeT5p	32K	RTLLM	1.21	0.34	0.00	3.87	1.54	0.00	5.15	2.63	0.00	6.90	3.45	0.00
			VGen	14.41	0.29	1.76	19.79	1.47	6.48	19.97	2.94	11.15	23.53	5.88	17.65
		64K	RTLLM	0.86	1.90	0.52	3.95	4.89	2.07	6.99	6.08	3.09	10.34	6.90	3.45
			VGen	**15.88**	10.59	7.06	21.09	13.23	13.54	22.91	14.71	16.22	23.53	17.65	17.65
		96K	RTLLM	5.00	0.69	0.69	11.91	2.93	3.09	15.46	5.17	5.26	17.24	10.34	6.90
			VGen	14.71	9.71	7.65	**23.24**	11.76	12.47	**29.47**	11.76	14.67	**35.29**	11.76	17.65
		128K	RTLLM	**5.52**	0.52	0.34	**14.58**	2.40	1.54	**19.82**	4.36	2.63	**27.59**	6.90	3.45
			VGen	15.29	9.71	9.41	21.57	11.76	11.74	23.34	11.76	11.76	23.53	11.76	11.76
Syntax	CodeLlama	34K	RTLLM	60.52	14.31	40.69	80.61	30.67	70.47	84.52	39.28	77.85	86.21	44.83	82.76
			VGen	86.76	69.71	88.82	99.14	80.61	99.48	99.97	82.10	99.97	**100.00**	88.24	**100.00**
		68K	RTLLM	60.69	26.90	53.45	79.48	51.12	75.87	84.87	62.85	81.10	**89.66**	68.97	82.76
			VGen	97.65	71.76	60.88	100.00	82.40	89.90	100.00	86.48	93.73	**100.00**	88.24	94.12
		102K	RTLLM	**66.55**	36.72	45.52	**84.10**	62.56	72.08	**88.82**	72.48	78.80	**89.66**	79.31	82.76
			VGen	96.47	66.18	75.59	**100.00**	78.82	97.14	**100.00**	81.64	99.80	**100.00**	88.24	**100.00**
		136K	RTLLM	66.38	39.48	33.28	80.97	61.11	64.90	84.46	67.88	74.84	86.21	72.41	79.31
			VGen	**99.12**	67.65	73.53	**100.00**	79.89	96.94	**100.00**	82.65	99.78	**100.00**	88.24	**100.00**
	CodeT5p	32K	RTLLM	12.59	9.66	10.86	33.97	22.50	32.60	46.19	28.52	45.12	58.62	34.48	51.72
			VGen	63.53	17.06	34.12	85.74	47.09	71.55	90.86	59.36	83.75	94.12	70.59	94.12
		64K	RTLLM	28.28	16.21	15.17	52.27	35.71	38.19	61.96	43.35	51.73	68.97	51.72	65.52
			VGen	74.12	46.76	26.47	86.85	77.14	64.50	88.14	84.70	79.40	88.24	88.24	88.24
		96K	RTLLM	33.28	12.76	24.83	59.49	29.05	46.58	67.72	38.94	59.44	72.41	48.28	68.97
			VGen	**82.35**	42.94	52.65	97.79	72.81	83.97	**99.93**	81.68	92.84	**100.00**	88.24	94.12
		128K	RTLLM	**44.83**	10.69	19.48	**62.67**	24.93	37.30	**72.54**	33.22	46.31	**79.31**	41.38	55.17
			VGen	77.35	43.82	35.88	**98.00**	75.25	71.91	99.89	84.38	82.89	**100.00**	94.12	88.24

TABLE II
EVALUATION RESULTS FOR THE SPEED OF GENERATING VERILOG CODE

Method	CodeLlama		CodeT5p	
	Speed (tokens/s)	Speedup	Speed (tokens/s)	Speedup
Ours	**420.13**	**5.05**	**243.70**	**2.66**
Medusa	294.99	3.55	106.33	1.16
NTP	83.13	1	91.65	1

further illustrate the effectiveness of our method, Fig. 5 compares the decoding process of our approach to those of Medusa and NTP for a specific example. Notably, our method generates the output in significantly fewer steps while maintaining the integrity of the syntactic structure at each decoding step.

The output quality of models fine-tuned with varying amounts of training data on the RTLLM [16] and VGen [22] benchmarks is summarized in Table I. The highest values for each benchmark, within the same model architecture and across different training data sizes and methods, are highlighted in bold. For clearer visualization, Fig. 6 highlights the pass@5 results of our method compared to the baselines using the CodeT5p architecture.

The models trained with our method show significant improvements in both functional and syntactic accuracy compared to those trained with Medusa. On the VGen benchmark, our method achieves a maximum functional accuracy increase of 30.5% using the pass@10 metric. For syntactic accuracy, it delivers a substantial improvement of 49.94% in the pass@5 metric on the RTLLM benchmark. When compared to the models fine-tuned with NTP, our approach achieves a functional accuracy improvement of up to 17.19% using the pass@10 metric on the RTLLM benchmark and a syntactic accuracy gain of up to 47.65% in the pass@1 metric on the VGen benchmark. Averaged across all benchmarks and evaluation metrics, models trained with our method show a 13.03% improvement in functional accuracy over Medusa and a 5.91% improvement over NTP. For syntactic accuracy, our method demonstrates an average enhancement of 22.9% over Medusa and 11.8% over NTP. Additionally, our approach's strong performance on small datasets highlights its effectiveness, achieving competitive results without requiring extensive additional data.

V. CONCLUSION

In this work, we introduce a novel application of speculative decoding for Verilog code generation, demonstrating its potential to enhance both inference speed and output quality. By aligning decoding stops with syntactically significant tokens extracted from ASTs, our method addresses the limitations of conventional tokenization and grammar-based approaches, enabling models to more effectively capture Verilog's structural and semantic intricacies. The proposed method achieves significant advancements in generating both syntactically and functionally correct code while substantially accelerating inference, delivering up to a 17.19% improvement in pass@10 functional accuracy on the RTLLM benchmark and achieves up to a 5.05× speedup in Verilog code generation compared to the conventional NTP scheme. Furthermore, our approach enhances model inference speed by 1.42–2.29× over the original MEDUSA method. These results underscore the efficacy of leveraging syntax-enriched speculative decoding to unlock new possibilities for applying LLMs to specialized programming languages, paving the way for more efficient and reliable Verilog code generation.

VI. ACKNOWLEDGMENTS

This work was supported in part by the General Research Fund of the Hong Kong Research Grants Council (RGC) under Grant No. 14212422 and 14202824, and in part by National Technology Innovation Center for EDA.

References

[1] Y. Wang, W. Wang, S. Joty, and S. C. Hoi, "Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 8696–8708.

[2] Y. Wang, H. Le, A. Gotmare, N. Bui, J. Li, and S. Hoi, "Codet5+: Open code large language models for code understanding and generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023, pp. 1069–1088.

[3] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, "Code llama: Open foundation models for code," *arXiv preprint arXiv:2308.12950*, 2023.

[4] Q. Zhu, D. Guo, Z. Shao, D. Yang, P. Wang, R. Xu, Y. Wu, Y. Li, H. Gao, S. Ma *et al.*, "Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence," *arXiv preprint arXiv:2406.11931*, 2024.

[5] R. Sennrich, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[6] M. Rabinovich, M. Stern, and D. Klein, "Abstract syntax networks for code generation and semantic parsing," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017, pp. 1139–1149.

[7] Z. Sun, Q. Zhu, L. Mou, Y. Xiong, G. Li, and L. Zhang, "A grammar-based structural cnn decoder for code generation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, no. 01, 2019, pp. 7055–7062.

[8] Z. Sun, Q. Zhu, Y. Xiong, Y. Sun, L. Mou, and L. Zhang, "Treegen: A tree-based transformer architecture for code generation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 05, 2020, pp. 8984–8991.

[9] Y. Xiong and B. Wang, "L2s: A framework for synthesizing the most probable program under a specification," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 3, pp. 1–45, 2022.

[10] Q. Zhu, Z. Sun, W. Zhang, Y. Xiong, and L. Zhang, "Grape: Grammar-preserving rule embedding." in *IJCAI*, 2022, pp. 4545–4551.

[11] Q. Zhu, Q. Liang, Z. Sun, Y. Xiong, L. Zhang, and S. Cheng, "Grammart5: Grammar-integrated pretrained encoder-decoder neural model for code," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 2024, pp. 1–13.

[12] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 274–19 286.

[13] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," *arXiv preprint arXiv:2302.01318*, 2023.

[14] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi *et al.*, "Specinfer: Accelerating large language model serving with tree-based speculative inference and verification," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 932–949.

[15] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple llm inference acceleration framework with multiple decoding heads," *arXiv preprint arXiv:2401.10774*, 2024.

[16] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[17] H. Pearce, B. Tan, and R. Karri, "Dave: Deriving automatically verilog from english," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 2020, pp. 27–32.

[18] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[19] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[20] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[21] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[22] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[23] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[24] K. Chang, K. Wang, N. Yang, Y. Wang, D. Jin, W. Zhu, Z. Chen, C. Li, H. Yan, Y. Zhou *et al.*, "Data is all you need: Finetuning llms for chip design via an automated design-data augmentation framework," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[25] Y. Zhang, Z. Yu, Y. Fu, C. Wan, and Y. C. Lin, "Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[26] F. Cui, C. Yin, K. Zhou, Y. Xiao, G. Sun, Q. Xu, Q. Guo, D. Song, D. Lin, X. Zhang *et al.*, "Origen: Enhancing rtl code generation with code-to-code augmentation and self-reflection," *arXiv preprint arXiv:2407.16237*, 2024.

[27] Y. Zhao, D. Huang, C. Li, P. Jin, Z. Nan, T. Ma, L. Qi, Y. Pan, Z. Zhang, R. Zhang *et al.*, "Codev: Empowering llms for verilog generation through multi-level summarization," *arXiv preprint arXiv:2407.10424*, 2024.

[28] X. Wang, Y. Wang, F. Mi, P. Zhou, Y. Wan, X. Liu, L. Li, H. Wu, J. Liu, and X. Jiang, "Syncobert: Syntax-guided multi-modal contrastive pre-training for code representation," *arXiv preprint arXiv:2108.04556*, 2021.

[29] X. Jiang, Z. Zheng, C. Lyu, L. Li, and L. Lyu, "Treebert: A tree-based pre-trained model for programming language," in *Uncertainty in Artificial Intelligence*. PMLR, 2021, pp. 54–63.

[30] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "Unixcoder: Unified cross-modal pre-training for code representation," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022, pp. 7212–7225.

[31] Z. Yan, J. Liu, G. Li, Z. Han, and S. Qiu, "Privmin: Differentially private minhash for jaccard similarity computation," *arXiv preprint arXiv:1705.07258*, 2017.

[32] X. Chen, Y. Meng, and G. Chen, "Incremental verilog parser," in *2023 International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2023, pp. 236–240.

[33] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[34] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford alpaca: An instruction-following llama model," https://github.com/tatsu-lab/stanford_alpaca, 2023.

[35] Axolotl, "Axolotl," https://github.com/OpenAccess-AI-Collective/axolotl, 2023.

[36] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023.

[37] S. Williams, "Icarus verilog," http://iverilog.icarus.com, 2024, accessed: 2024-11-19.

EPIC: Error PredIction and Correction for Power-Efficient Voltage Underscaling Multiply-Accumulate Unit

Tongjing Wu*, Xiaolu Hu*, Tong Li*, Siting Liu†, Hui Wang*, Weifeng He*, Zhigang Mao*, Honglan Jiang*
*Department of Micro-Nano Electronics, Shanghai Jiao Tong University, Shanghai, China
†School of Information Science and Technology, ShanghaiTech University, Shanghai, China
{tongjing_wu, huxiaolu, tongli_licu, ihuiwang, hewf, maozhigang, honglan}@sjtu.edu.cn and liust@shanghaitech.edu.cn

***Abstract*—Matrix multiplication dominates the power consumption in compute-intensive applications such as deep neural networks (DNNs), spurring intensive investigations into power-efficient multiply-accumulate (MAC) units. Among the mainstream low-power design methodologies, voltage underscaling can achieve effective power savings yet induce timing errors that may lead to catastrophic accuracy loss. In this paper, we propose an error prediction and correction framework (denoted as EPIC) for arbitrary MAC unit under voltage underscaling, which predicts the timing errors and samples the correct output by using a delay-tunable clock. A prediction bits searching algorithm is proposed to enhance the prediction accuracy with low hardware cost, resulting in up to 100% accuracy. While preserving the accuracy, EPIC achieves up to 52% power savings over the corresponding MAC operating at nominal voltage. With transistor-level optimizations, EPIC incurs only 8% area and 1% power overheads, achieving 100% error correction under a voltage underscaling ratio of 0.74. Compared to state-of-the-art error resilient circuit designs, EPIC consumes 60%-88% less area. Additionally, to achieve the accuracy performance of EPIC in error-resilient applications, we propose a simulation workflow involving precise timing features, enabling an accurate simulation of voltage underscaling MAC in large-scale applications. The experimental results show that, under voltage-underscaling, the MAC with EPIC consumes 11% less power than the one without EPIC, when a same accuracy as exact implementation is required in multi-layer perceptron (MLP).**

***Index Terms*—Power-efficient, timing error resilient circuit, voltage underscaling**

I. Introduction

With a rapid development, DNNs have been proven highly effective across various machine learning applications, including classification, object detection, and generation [1], [2]. To enable efficient hardware deployment, computing hardware like graphics processing unit (GPU) has been tailored to support dominate computations in DNNs. Matrix multiplication is the most basic and key operation for DNNs, especially for the Transformer-based architectures that have emerged as backbone networks in many large language models [3]–[6]. Thus, the implementation of matrix multiplication significantly affects the power efficiency of the entire computing core. While the scale of DNNs continually evolves, the demand on power-efficient matrix multiplication is increasingly high.

In general, matrix multiplication is implemented by using a systolic array consisting of a large number of MAC units [7]. Thus, power-efficient MAC unit has been of particular interest. Numerous design methods have been proposed to reduce the power consumption of MAC from various perspectives, including approximate computing [8], algorithm optimization [9], and voltage underscaling [10]–[12]. Among them, voltage underscaling is promising in reducing power dissipation as it is a quadratic function of the supply voltage. However, aggressive voltage underscaling would incur serious timing errors due to the slow switching of transistor states in low voltage, which may lead to catastrophic accuracy loss in applications [13]. To address this issue, some schemes have been proposed to speed up circuits under low voltage, including redesigning the standard cells [14], leveraging body bias [15] and adopting dual-VDD [16]. Nevertheless, these techniques are tailored to specific circuits, with limited portability and high implementation complexity. A more feasible technique based on error detection and correction (EDaC) has widely been investigated and implemented, which aims to detect the timing errors due to voltage underscaling and recover them.

Many previous works have demonstrated that EDaC and EDaC-like designs can effectively recover timing errors or mitigate the impact of timing errors in DNN accelerators under voltage underscaling [13], [17]–[25]. In [13], the outputs of the processing element (PE) with timing errors are dropped from the sum of product result; this is valid due to the fact that lacking several multiplication results leads to smaller impact than a wrong partial sum. Based on this scheme, [20] adds a compensation circuit to further enhance the accuracy of DNN accelerators under voltage underscaling. [19], [21] propose designs tailored to the characteristics of the inference network. [19] predicts whether errors will occur based on the input activations and raises the voltage if necessary, while [21] identifies weights that are more sensitive to voltage underscaling and maps them to more robust MACs. In [18], a simple razor flip-flop is used to detect and correct the errors due to timing violations. However, using Razor for EDaC, the designs mentioned above fail to detect or recover all the output bits in a MAC due to the area-accuracy trade-off [26]. In [23], rather than Razor, transition detector (TD) is devised to detect timing errors; latches are used for error correction, resulting in a relatively high area overhead.

In this paper, we propose EPIC for arbitrary MAC unit under voltage underscaling, which predicts and corrects the timing errors with negligible area and power overheads. Specifically, the contributions of this work are as follows.

- The EPIC framework is devised for voltage underscaling MAC, where several internal signals (referred to as prediction bits) in an MAC are utilized to predict the timing errors, rather than comparing the outputs of MAC as traditional EDaC does. The errors are then recovered by sampling the correct output by using a delay-tunable clock. Instead of designing the error resilient circuit at gate-level with Razor, we optimize the EPIC-equipped circuit at transistor-level, resulting in 60%-88% less area consumption compared to state-of-the-art designs.
- We propose a searching algorithm for the combination form of prediction bits, aiming to enhance the prediction accuracy with low hardware cost. While maintaining accuracy, EPIC saves up to 52% power for the MAC under aggressive voltage underscaling, compared with the corresponding MAC operating at nominal voltage.
- We propose a timing-accurate simulation workflow to assess the accuracy of voltage underscaling MAC at application level. Different from error injections, the proposed framework involves precise timing features of circuits, achieving the same results as

This work was supported in part by the National Key Research and Development Program of China under grant 2022YFB4500200 and in part by the National Natural Science Foundation of China under grant numbers 62204155 and 62374108.

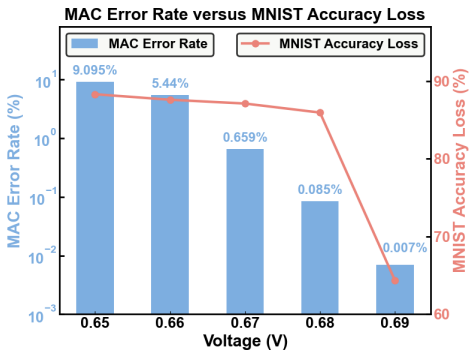


Fig. 1: Error rate in a single low-voltage MAC versus accuracy loss on MNIST classification.

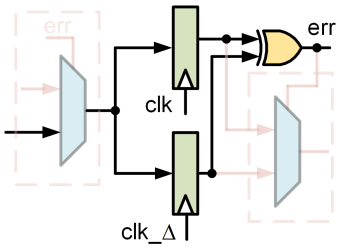


Fig. 2: Schematic of Razor flip-flop.

the corresponding analog/mixed-signal (AMS) simulation while saving time by more than three orders of magnitude. To the best of our knowledge, this is the first work to perform accurate simulation in large-scale applications for voltage underscaling MAC equipped with transistor-level resilient circuits.

II. Background and Motivation

A. The Impacts of Timing Errors in MAC on Applications

To investigate the relationship between the timing errors of MACs and the application accuracy, we utilize a commonly used systolic array to perform MLP inference on the MNIST dataset. Each MAC is synthesized with CMOS 28nm standard cell library using multiplier and adder from Synopsys DesignWare [27], [28]. The error rate of the MAC under each voltage is obtained by providing 300,000 combinations of random numbers as the inputs using VCS simulation. The nominal voltage of the library is 0.9V, corresponding to an MLP inference accuracy of 98.3%.

As shown in Fig. 1, the accuracy loss reaches up to 64.34% even though the error rate in the MAC is only 0.007%. A possible reason can be that, the rarely occurred timing errors lead to significant large error distances, e.g., causing a correct negative result to be sampled as a large positive value, which severely impacts the final outcome. As the voltage continues to decrease, the error rate of MAC rises significantly, while accuracy loss of MLP remains around 87%. At this point, the systolic array can no longer perform reliable computation, which necessities the EDaC for MAC.

B. Razor flip-flop

Razor flip-flop has widely been utilized for EDaC due to its ease of implementation [13], [17]–[21]. As shown in Fig. 2, the key of Razor is borrowing time from the next cycle by double-sampling with two flip-flops, one driven by a normal clock and the other by a delayed clock. The interval between the normal clock and the delayed clock is defined as the detection window. An XOR gate is utilized to compare the outputs of the two flip-flops, generating an error signal as a symbol of timing error. This error signal can then be further used for error processing, e.g., [13] deploys a multiplexor (MUX) before the output register to determine whether to bypass the multiplier, while [18] puts a MUX after the output register to directly obtain the correct result. However, the area overhead caused by the extra flip-flop, MUX and XOR gate will be excessively high if Razor is employed for each bit of MAC outputs, leading to significant area and power consumption for the systolic array. Given this area-accuracy trade-off, a prevalent method is employing Razor for significant bits of MAC outputs instead of all bits [13], [18], [20], which compromises the effectiveness of EDaC.

C. Limitations of Error Injection-Based Simulations

Different from those at nominal voltage, timing specifications of the standard cells under aggressive voltage underscaling are not provided in the library; thus, application-level timing simulation by importing standard cell library for low-voltage circuits is not available. A common methodology for simulating low-voltage circuits at application-level is inserting errors to nominal implementations with specific error rate that is extracted from HSPICE simulation [13], [20], [21]. However, the error due to timing violations highly depends on the current and previous input patterns. Thus, the injected errors as per simple error rate can hardly reflect the timing performance of the circuits under voltage underscaling, as clearly illustrated in Fig. 3.

Fig. 3 diagrams how a timing error happens in a 4×4 array multiplier, where pp1 to pp4 denote the partial product vectors generating by AND gates. The initial inputs are assumed to be 0011 and 1101, resulting in the corresponding accurate output of 00100111. Subsequently, the inputs change to 0101 and 0010; the partial products change accordingly after an AND operation at time t_1. Then, after the time for completing a full addition (FA), at time t_2, sum1-3 and carry1-3 are updated. If the output is sampled between t_1 and t_2, the obtained multiplication result for 0101 and 0010 becomes 00100110. At this time, the error due to timing violation is 28. Similarly, the timing error is 32 when the output is sampled between t_2 and t_3. These indicate that the error distance caused by timing violations is closely related to the internal signal states of a circuit and may persist until the timing margin is satisfied. Computing 0101×0010 requires additional time due to the logic 1 left in the fifth output bit from the previous operation. Thus, timing errors and sampled values depend on both current and previous input patterns as well as prior outputs.

III. EPIC Design

A. Design Overview

Fig. 4 illustrates the MAC unit equipped with the proposed EPIC, where the inputs are 8-bit and the output is 24-bit. In this design, TD is utilized to predict the timing errors by detecting the flips of the logical combination of prediction bits that are selected by a searching algorithm. When a timing error occurs, a delay-tunable clock "clk_Δ" switched by the MUX is used to sample the adder result; otherwise, the nominal clock "clk" is utilized. When correcting the timing errors, the inputs of the adder, including the output from the previous MAC and all bits of the multiplier output (assuming the multiplier completes computation before the rising edge of the clock), are maintained by the transmission gates (TGs) to pad the short paths, preventing the correct data from being overwritten. Unlike EDaC designs using Razor, the timing error correction in this design covers all output bits of the MAC.

B. Prediction Bits Searching for Error Detection

The motivation for error prediction in EPIC lies in the high correlation between the flips of certain internal signals in MAC and timing errors. While sampling the inputs and output at rising edge of the clock, the flips of the more significant output bits in the multiplier and the carry bits in the adder during the negative clock phase are more likely to indicate a long critical path delay in the MAC. However, it is of great difficulty in identifying proper combination of signals for timing error prediction, with low hardware overhead. In this work, we propose

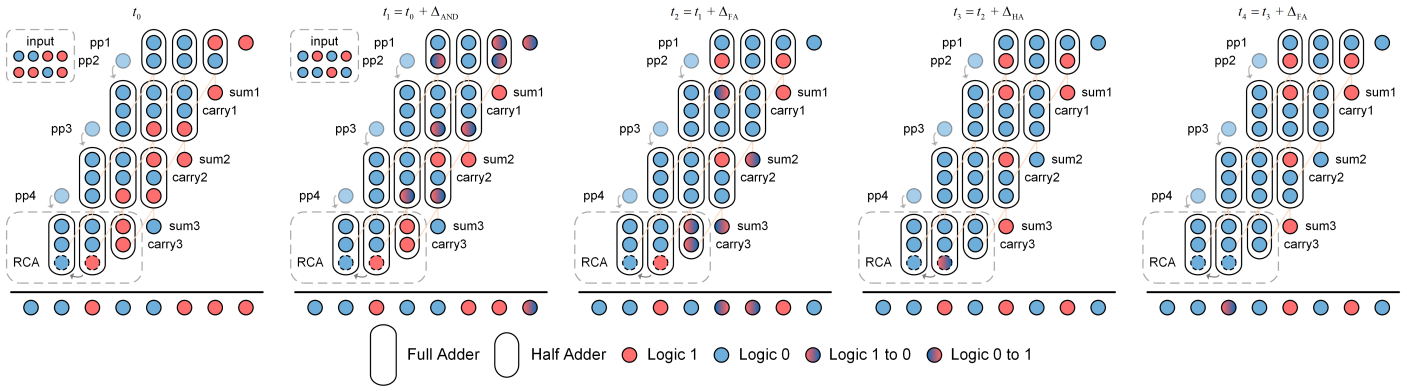


Fig. 3: Diagrams of how a timing error happens in a 4 × 4 array multiplier.

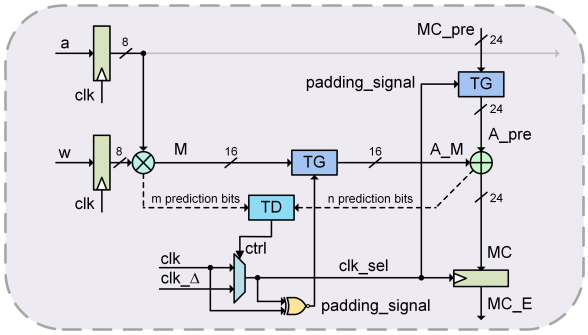


Fig. 4: Schematic of MAC equipped with EPIC.

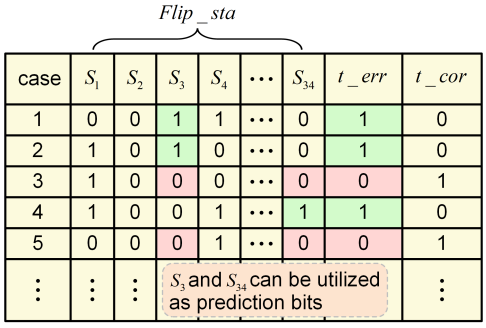


Fig. 5: An example for prediction bits searching.

a searching algorithm to identify the optimal combination form with the fewest internal signals in an arbitrary MAC design, while achieving high prediction accuracy for timing errors.

As shown in Algorithm 1, the inputs include the flip statistics ($Flip_sta$) of internal signals on the critical path and the corresponding labels for timing errors (t_err), and timing correctness (t_cor), which are obtained based on Synopsys VCS simulations with precise timing features inserted. $Flip_sta$ records the flipping behaviors of each selected signal in the negative clock phase, based on 300,000 random input patterns. t_err and t_cor label the output state for each input pattern, denoting the occurrence and non-occurrence of timing errors, respectively.

First, the number of signals num to be used is restricted within the range of 1 to max_comb. To shorten the computation time and guarantee a low hardware cost, max_comb is set to 6 in this work. Next, num signals are selected from the candidate signals recorded in $Flip_sta$; all possible logical combinations of these signals (using only AND and OR gates) are then generated. If a logical combination can predict all timing errors, its prediction accuracy for timing correctness is calculated based on the misprediction rate for t_cor, as shown in line 7. Through iterative loops, the best combination of signals S_{opt},

Algorithm 1: Prediction Bits Search

Input: $Flip_sta$, t_err, t_cor
Output: S_{opt}, C_{opt}, acc_opt
1 **for** $num \in [1, max_comb]$ **do**
2 *// Generate all combinations of num bits in Flip_sta*
3 **for** *each selected signal set* S_{curr} **do**
4 *// Generate all logical combinations for* S_{curr}
5 **for** *each logical combination* C_{curr} **do**
6 **if** $t_err \subseteq pred_err$ **then**
7 $acc_curr = 1 - (|pred_err| - |t_err|)/|t_cor|$
8 **if** $acc_curr > acc_opt$ **then**
9 $S_{opt} = S_{curr}$
10 $C_{opt} = C_{curr}$
11 $acc_opt = acc_curr$
12 **end**
13 **end**
14 **end**
15 **end**
16 **end**

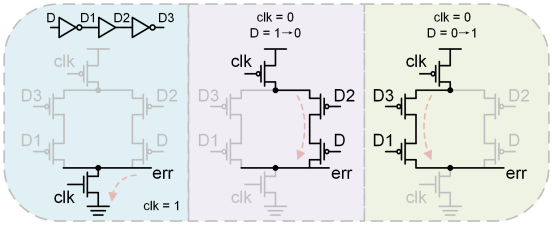
along with the corresponding logical combination C_{opt} and prediction accuracy acc_opt are obtained. Fig. 5 gives a simple example of how prediction bits are selected based on the algorithm. Among the five hypothetical cases, the OR operation of S_3 and S_{34} can predict all timing errors and timing correctness. In contrast, other combinations either fail to predict all timing errors or exhibit a high rate of prediction errors for timing correctness. Fig. 7b illustrates how EPIC predict and correct the timing errors utilizing the combination of prediction bits.

C. Transition Detector Design

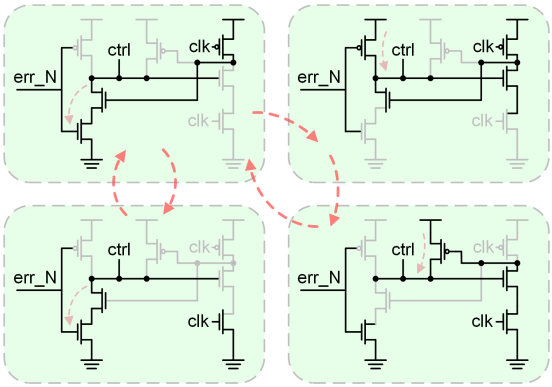
To monitor the flips of the selected prediction bits, TDs are inserted before the adder. Inspired by [23], we propose a power-efficient and low-latency TD consisted of 14 transistors with its detection window positioned in the negative clock phase to enhance the correlation between prediction bits and timing errors.

As shown in the top left of Fig. 6a, the signal D to be monitored is fed to an inverter to generate an inverted signal D1; D1 is then delayed by a buffer, generating D2; a delayed D is finally achieved by inverting D2, denoted as D3. During the positive clock phase, the node "err" is discharged; thus, the signal transition is not monitored. Fig. 6c shows the timing diagram of the related signals, regardless of the input value or whether it changes, the "err" signal remains 0 during the positive clock phase.

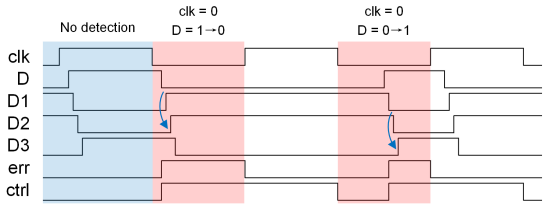
As shown in the middle of Fig. 6a, during the negative clock phase, if D changes from 1 to 0, there is a delay for D2 to change from 0 to 1, causing the transistors on the right to turn on simultaneously, thus creating a charge path from VDD to the node "err". In this case,



(a) The schematic and working principle of TD.



(b) The Schematic and working principle of the control signal generation circuit.



(c) Timing diagram of TD and the control signal generation circuit.

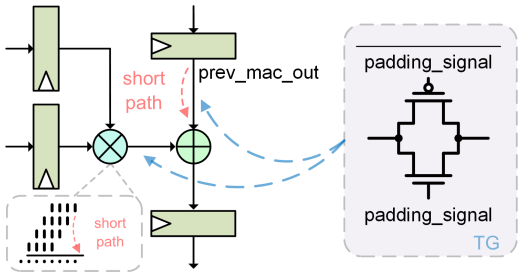
Fig. 6: Schematics, working principle, and timing diagram of TD and the control signal generation circuit.

TD generates a positive pulse to indicate an input transition in the detection window, as shown in Fig. 6c. Similarly, the right of Fig. 6a shows the detection of D changing from 0 to 1 during the negative clock phase, another charge path is activated on the left due to the delay between D3 and D1.

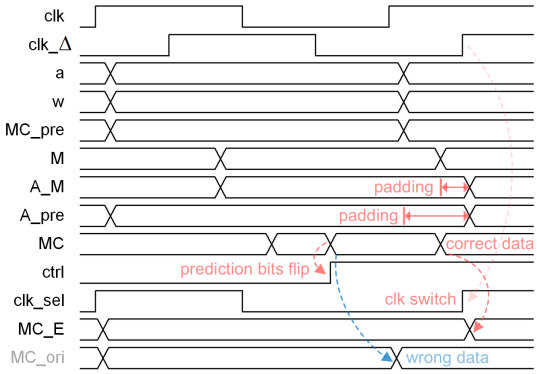
When a timing error is predicted to occur, a delayed clock is selected to sample the correct multiplication result. Thus, the prediction result must be available until the next falling edge of the clock to ensure a proper clock switching. However, as the "err" is reset to 0 at the rising edge of the clock, the logical combination of the "err" signals for the prediction bits will disappear at the rising edge of the clock. Thus, a control signal generation circuit is devised to extend the "err" signal until the next clock's falling edge. As shown in Fig. 6b and Fig. 6c, at the beginning of the negative clock phase, the "err" is 0, resulting in the activation of the discharge path from node "ctrl" to the ground. The "err_N" signal, which is the inverted "err" signal, will change to 0 when the prediction bit flips, resulting in the charging of node "ctrl". During the positive clock phase, the feedback path is activated, maintaining "ctrl" at 1 until the falling edge of the clock (bottom right of Fig. 6b). Another possible scenario is that, during the negative clock phase, if the prediction bits do not flip, the "ctrl" signal remains 0 during the positive clock phase, which is shown in the bottom left of Fig. 6b.

D. Short-path padding

The short-path problem is severe in EDaC designs, which is resulted from the activation of short path within the detection window. As shown in Fig. 7a, when the output registers are ready to sample the results at the rising edge of the delayed clock due to a predicted timing error, an early arrival of the adder inputs can happen before the correct data is sampled because the clock of the input registers remains unchanged. The possible early arrived inputs include the output from the previous MAC and the lower bits of the multiplier output.



(a) Illustration of TG for short-path padding.



(b) Timing diagram of TG and EPIC.

Fig. 7: Illustration and timing diagram of TG.

The short-path problem is intrinsically a hold-time violation, which can be solved by imposing a hold constraint that matches the width of the error detection window on all monitored paths [22]. As EDaC designs based on Razor are commonly devised at gate-level, the traditional method for hold-time violation correction involves inserting buffers. However, this introduces a trade-off between area and accuracy, as larger detection window requires more buffers. Additionally, since the number of padding buffers is hard to adjust in post-fabrication, the detection window can hardly be modified in these designs.

Benefiting from the transistor-level design of EPIC, we adopt TGs for short-path padding [29], with extremely low area overhead and high compatibility with the delay-tunable clock. As shown in Fig. 7a, TG is controlled by a padding signal, which is inserted between the adder and its inputs to prevent the early arrival of the inputs. For the input from the previous MAC, the padding signal is the "clk_sel" generated by the clock switching MUX. This ensures that the output from the previous MAC will be blocked until the rising edge of the delayed clock, while it passes through normally when the normal clock is used. For the input from the multiplier, we insert TG for each bit as we assume that all the multiplications can complete before the rising edge of the clock. In this case, the padding signal is generated by performing an XNOR operation on "clk_sel" and "clk", ensuring a normal propagation of the multiplier output during the negative clock phase. As shown in Fig. 7b, the inputs of the adder are extended to the rising edge of the delayed clock when an error is predicted to happen. As the short-path padding signal is derived from "clk_sel" and "clk", the detection window in EPIC can be easily adjusted by tuning the delayed clock.

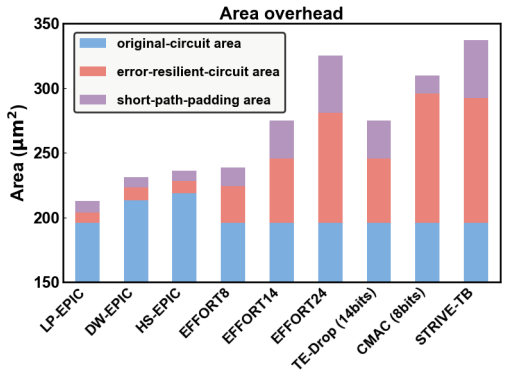


Fig. 8: Area overhead comparison of EPIC-based MAC variants, EFFORT [18], TE-Drop [13], CMAC [20], STRIVE-TB [17].

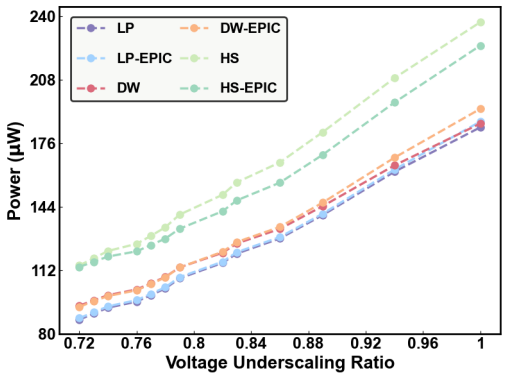


Fig. 9: Power of MACs under different voltage underscaling ratio.

E. Variants of EPIC-Based MAC

To demonstrate the generality of EPIC, we design three MAC structures along with their EPIC-equipped counterparts.

- Low power design (LP): The MAC is completely flattened by EDA tool to minimize power and area, with the multiplication and addition operations merged.
- Design ware design (DW): The multiplier and adder are from Synopsys DesignWare [27], [28], which retains the original multiplier output.
- High speed design (HS): The ripple carry adder (RCA) for the final addition in the multiplier is replaced with a parallel prefix adder based on BK-tree [30], which shortens critical path of the multiplier.

IV. Experiments

In this section, we evaluate EPIC in a standalone MAC unit and in large-scale applications. In Section IV-A, we compare the area, power consumption, and prediction accuracy of MAC units with and without the prediction bits searching algorithm. Additionally, we compare the area of MAC units using EPIC with those employing EDaC from other works. Finally, we evaluate the error correction performance and power overhead of EPIC under different corners. In Section IV-B, we expound upon the detailed work flow for simulation on large-scale application and the corresponding results.

A. Evaluation of MAC Unit

To demonstrate the efficiency of our proposed prediction bits searching algorithm, we evaluate the area, power, and prediction accuracy of the EPIC-based MAC variants, both with and without using the algorithm. For DW and HS, the versions without using algorithm utilize the 6 most significant bits (MSBs) of the multiplier output as the prediction bits while 6 MSBs of the adder carry are used

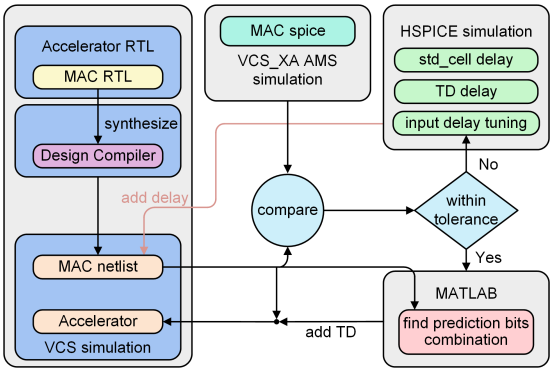


Fig. 10: Timing-accurate simulation workflow for applications.

in LP. Befitting from the searching algorithm, LP utilizes the AND result of two internal signals, DW uses the OR result of the MSB of the multiplier output and two significant bits of adder carries, while HS employs the AND result of the 11th and 12th output bits of the multiplier, followed by an OR operation with the 11th carry bit of the adder. With a 1GHz clock, both LP and DW are tested at 0.67V while HS is tested at 0.66V, with corresponding voltage underscaling ratios of 0.73 and 0.74, respectively.

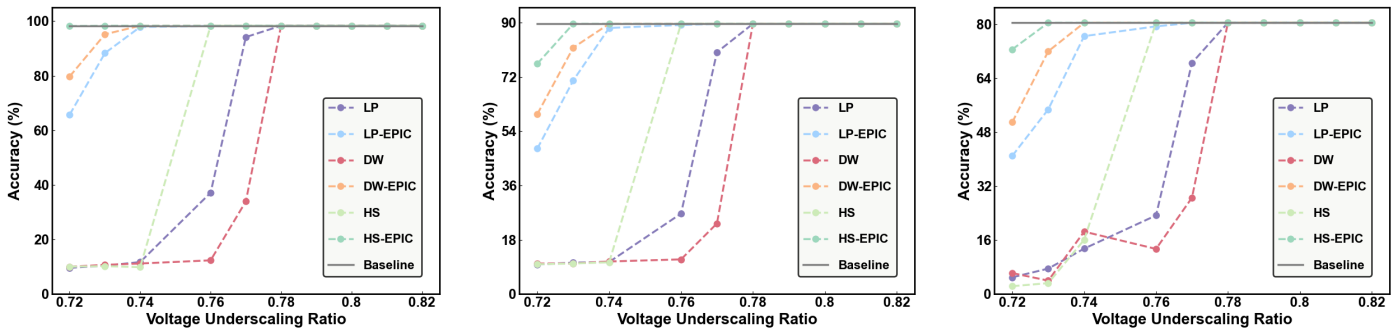
Table I shows the comparison results, where te and tc represent the probabilities of accurately predicting the timing errors and error-free timing, respectively, p denotes the power consumption. It can be seen that EPIC-based MAC variants utilizing the prediction bits searching algorithm achieve 100% prediction accuracy for timing errors, with relatively low power dissipation. This indicates that, benefiting from the algorithm, the EPIC can detect all timing errors, and some predicted timing errors do not happen, resulting in $tc < 100\%$. In contrast, EPIC-based MACs without using the algorithm fail to guarantee 100% prediction accuracy for timing errors, while exhibiting higher power consumption.

To investigate the area overhead of EPIC, we synthesize EPIC-based MAC variants and calculate the area based on CMOS 28nm standard cell library. For designs utilizing Razor, we insert buffers for short-path padding using PrimeTime and report the area overhead. Since these Razor-based designs achieve EDaC by comparing only the outputs of the MAC without requiring the multiplier results, they are implemented using LP to minimize area. Note that the synthesis clock is 1GHz for all the considered designs. As shown in Fig. 8, the area overhead caused by short-path padding increases with the number of protected bits, accounting for up to 14% of the total area. Similarly, the area overhead introduced by Razor-based EDaC circuits is also significant, reaching up to 32% of the total area. In contrast, for circuits utilizing EPIC, the timing-resilient circuits account for up to 4.2% of the total area, while the circuits for short-path padding account for up to 4.3%. Compared to the resilient circuit designs in other works, EPIC consumes 60%-88% less area, with only 8% area overhead in total.

To demonstrate the robustness of EPIC, we also measure the performance of EPIC-based MAC variants at different CMOS technology corners, namely ff, tt, and ss. As shown in Table II, V_{EPIC} and V_{ori} represent the minimum voltages achievable by the circuit with EPIC and the original circuit, respectively, under the condition of no timing errors. P_{overhead} denotes the power overhead of the circuit with EPIC

TABLE I: Comparisons of EPIC-based MAC variants with and without using prediction bits searching algorithm

EPIC variants	with algorithm			without algorithm		
	te (%)	tc (%)	p (μW)	te (%)	tc (%)	p (μW)
LP	100	99.5	93.99	98	81.8	99.43
DW	100	87.3	99.2	98.7	92.1	103.9
HS	100	66.1	116.4	93.8	50.3	121.5



(a) MNIST. (b) FMNIST. (c) Reuters.

Fig. 11: Inference accuracy of systolic array on three MLP benchmarks with voltage scaling.

TABLE II: Comparisons of EPIC-based MAC variants at different CMOS technology corners (clk=1GHz).

EPIC Variants	corner	V_{EPIC} (V)	V_{ori} (V)	P_{overhead} (%)
LP	ff	0.57	0.6	1.52
	tt	0.67	0.7	0.70
	ss	0.76	0.8	0.45
DW	ff	0.56	0.6	0.90
	tt	0.67	0.7	-0.44
	ss	0.76	0.81	-1.16
HS	ff	0.56	0.59	-0.50
	tt	0.66	0.69	-1.52
	ss	0.76	0.80	-2.94

at V_{EPIC}. At different corners, EPIC maintains stable error correction performance across all EPIC-based MACs, with a maximum power overhead of only 1.52%. Note that, in some cases, the power overhead of EPIC is negative because the use of short-path padding may reduce adder glitches, resulting in less power than the original circuits. We measure the power consumption of all considered MACs versus voltage underscaling ratio using AMS simulation, as shown in Fig. 9. While preserving the accuracy, EPIC achieves up to 52% power savings over the corresponding MAC operating at nominal voltage.

B. Application

To demonstrate the potential of EPIC in large-scale applications, we deploy three MLP models on the weight-stationary systolic array consisting of 16x16 EPIC-based MACs mentioned in Section III-A. To enable an accurate simulation for voltage underscaling MAC equipped with transistor-level resilient circuits in large-scale applications, we propose a workflow for timing-accurate simulation, as illustrated in Fig. 10. As we focus only on timing errors within MAC, the key to enable an accurate and fast simulation for large-scale applications is building an efficient timing-accurate model for MAC. To this end, the Synopsys Design Compiler is first used to synthesize the RTL implementation of a MAC for constructing accelerator, resulting in the netlist of a MAC. Next, HSPICE is utilized to extract the timing features of standard cells and TDs under voltage underscaling; these timing features are then inserted into the synthesized MAC netlist using Verilog specify statements. Synopsys VCS is then used to simulate the MAC, and the simulation results are compared with those from Synopsys VCS_XA AMS simulation based on real timing features. If significant discrepancies are observed, the input delay of the test stimuli is fine-tuned in HSPICE, as variations in the input delay can lead to differences in the extracted timing features. The comparison is then repeated until the results are nearly identical. Subsequently, the flips of the selected internal signals in MAC and their corresponding timing error occurrence information are recorded and processed in MATLAB, where the prediction bits searching algorithm identifies the optimal prediction bits and their logic combination for TD. Using Verilog primitives, TDs and TGs are modeled and inserted into the MAC netlist. Finally, three pre-trained and quantized MLP models shown in Table III are deployed on the systolic array, and VCS is employed to simulate the inference of the MLP at a clock frequency of 1 GHz. Note that the considered MLP models are quantized into 8-bit integer format. By avoiding numerical simulations with transistor-level SPICE models, the proposed workflow achieves the same accuracy as AMS simulations while saving over three orders of magnitude in simulation time.

TABLE III: MLP models and their error-free accuracy.

Benchmarks		Accuracy (%)
Dataset	Architectures	
MNIST [31]	FC: 784×256×256×10	98.30
FMNIST [32]	FC: 784×256×256×10	89.56
Reuters [33]	FC: 2048×256×256×46	80.45

As shown in Fig. 11, EPIC demonstrates stable error correction performance across all three MAC variants on the three MLP benchmarks. According to Fig. 9, compared to MAC units operating at the minimum voltage required to maintain accuracy, EPIC achieves up to an 11% reduction in power consumption while preserving accuracy. Considering that MACs operating at low voltages encounter IR drop issues, a 10% voltage margin is needed to ensure MACs can maintain accuracy on three MLP benchmarks even with a 10% voltage drop. In this scenario, the operating voltages for the three MAC variants, LP, DW, and HS, should be 0.77V, 0.77V, and 0.75V, respectively. According to Fig. 9, equipped with EPIC, the power consumption of three MAC variants can be reduced by up to 15% while maintaining a 10% voltage margin.

V. Conclusion

In this paper, we propose EPIC, an efficient error prediction and correction framework for MAC units under voltage underscaling. EPIC uses several internal signals of the MAC to predict the timing errors and correct the errors by using a delay-tunable clock to borrow the time margin of computation during the next clock. Moreover, a searching algorithm is proposed to find the optimal prediction bits and their logic combinations for the timing error prediction; it achieves 100% prediction accuracy on timing errors with minimal hardware overhead. Overall, EPIC delivers up to 52% power savings compared to MACs without EPIC at nominal voltage, with only 8% area overheads while ensuring 100% error correction under a voltage underscaling ratio of 0.74. Compared to state-of-the-art designs, EPIC consumes 60%-88% less area. Furthermore, a timing-accurate simulation workflow incorporating precise timing features of circuits is proposed, enabling efficient evaluation in large-scale applications. Experimental results demonstrate an 11% power reduction in MLP inference with EPIC-based MACs under equivalent accuracy constraints.

References

[1] Z. Li, F. Liu, W. Yang, S. Peng, and J. Zhou, "A survey of convolutional neural networks: Analysis, applications, and prospects," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 12, pp. 6999–7019, 2022.

[2] M. Shafiq and Z. Gu, "Deep residual learning for image recognition: A survey," *Applied Sciences*, vol. 12, no. 18, p. 8972, 2022.

[3] M. A. K. Raiaan, M. S. H. Mukta, K. Fatema, N. M. Fahad, S. Sakib, M. M. J. Mim, J. Ahmad, M. E. Ali, and S. Azam, "A review on large language models: Architectures, applications, taxonomies, open issues and challenges," *IEEE Access*, vol. 12, pp. 26 839–26 874, 2024.

[4] S. Islam, H. Elmekki, A. Elsebai, J. Bentahar, N. Drawel, G. Rjoub, and W. Pedrycz, "A comprehensive survey on applications of transformers for deep learning tasks," *Expert Systems with Applications*, p. 122666, 2023.

[5] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[6] S. Kim, C. Hooper, T. Wattanawong, M. Kang, R. Yan, H. Genc, G. Dinh, Q. Huang, K. Keutzer, M. W. Mahoney, Y. S. Shao, and A. Gholami, "Full stack optimization of transformer inference: a survey," 2023. [Online]. Available: https://arxiv.org/abs/2302.14017

[7] B. Asgari, R. Hadidi, and H. Kim, "Meissa: Multiplying matrices efficiently in a scalable systolic architecture," in *2020 IEEE 38th International Conference on Computer Design (ICCD)*, 2020, pp. 130–137.

[8] X. Hu, A. Liu, X. Geng, Z. Wei, K. Jiang, and H. Jiang, "A configurable approximate multiplier for cnns using partial product speculation," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[9] M. H. Haider and S.-B. Ko, "Booth encoding-based energy efficient multipliers for deep learning systems," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 70, no. 6, pp. 2241–2245, 2023.

[10] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[11] P. N. Whatmough, S. K. Lee, D. Brooks, and G.-Y. Wei, "Dnn engine: A 28-nm timing-error tolerant sparse deep neural network processor for iot applications," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.

[12] B. Reagen, P. Whatmough, R. Adolf, S. Rama, H. Lee, S. K. Lee, J. M. Hernández-Lobato, G.-Y. Wei, and D. Brooks, "Minerva: Enabling low-power, highly-accurate deep neural network accelerators," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 267–278, 2016.

[13] J. Zhang, K. Rangineni, Z. Ghodsi, and S. Garg, "Thundervolt: enabling aggressive voltage underscaling and timing error resilience for energy efficient deep learning accelerators," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[14] J. Jun, J. Song, and C. Kim, "A near-threshold voltage oriented digital cell library for high-energy efficiency and optimized performance in 65nm cmos process," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 5, pp. 1567–1580, 2018.

[15] M. R. Kakoee and L. Benini, "Fine-grained power and body-bias control for near-threshold deep sub-micron cmos circuits," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, pp. 131–140, 2011.

[16] K. Kim and V. D. Agrawal, "Minimum energy cmos design with dual subthreshold supply and multiple logic-level gates," in *2011 12th International Symposium on Quality Electronic Design*, 2011, pp. 1–6.

[17] N. D. Gundi, Z. M. Mowri, A. Chamberlin, S. Roy, and K. Chakraborty, "Strive: Enabling choke point detection and timing error resilience in a low-power tensor processing unit," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[18] N. D. Gundi, T. Shabanian, P. Basu, P. Pandey, S. Roy, and K. Chakraborty, "Effort: A comprehensive technique to tackle timing violations and improve energy efficiency of near-threshold tensor processing units," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 10, pp. 1790–1799, 2021.

[19] P. Pandey, P. Basu, K. Chakraborty, and S. Roy, "Greentpu: Predictive design paradigm for improving timing error resilience of a near-threshold tensor processing unit," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 7, pp. 1557–1566, 2020.

[20] D. Ji, D. Shin, and J. Park, "An error compensation technique for low-voltage dnn accelerators," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 2, pp. 397–408, 2020.

[21] D. Shin, W. Choi, J. Park, and S. Ghosh, "Sensitivity-based error resilient techniques with heterogeneous multiply–accumulate unit for voltage scalable deep neural network accelerators," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 3, pp. 520–531, 2019.

[22] R. Uytterhoeven and W. Dehaene, "Completion detection-based timing error detection and correction in a near-threshold risc-v microprocessor in fdsoi 28 nm," *IEEE Solid-State Circuits Letters*, vol. 3, pp. 230–233, 2020.

[23] Z. Shen, W. Shan, Y. Du, Z. Li, and J. Yang, "Beyond eliminating timing margin: An efficient and reliable negative margin timing error detection for neural network accelerator without accuracy loss," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 5, pp. 1462–1471, 2022.

[24] Y. Du, J. Qian, Z. Shen, C. Wu, W. Shan, and X. Wang, "Dsc-trcp: Dynamically self-calibrating tunable replica critical paths based timing monitoring for variation resilient circuits," *IEEE Journal of Solid-State Circuits*, 2024.

[25] M. Safarpour, R. Inanlou, and O. Silvén, "Algorithm level error detection in low voltage systolic array," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 69, no. 2, pp. 569–573, 2021.

[26] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner *et al.*, "Razor: A low-power pipeline based on circuit-level timing speculation," in *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.* IEEE, 2003, pp. 7–18.

[27] Synopsys, "DesignWare DW02_mult," https://www.synopsys.com/dw/ipdir.php?c=DW02_mult, 2024.

[28] Synopsys, "DesignWare DW01_add," https://www.synopsys.com/dw/ipdir.php?c=DW01_add, 2024.

[29] W. Shan, W. Dai, C. Zhang, H. Cai, P. Liu, J. Yang, and L. Shi, "Tg-spp: A one-transmission-gate short-path padding for wide-voltage-range resilient circuits in 28-nm cmos," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 5, pp. 1422–1436, 2020.

[30] Abhilash and S. Hussain, "Comparative analysis of parallel prefix adders," in *2024 International Conference on Intelligent Algorithms for Computational Intelligence Systems (IACIS)*, 2024, pp. 1–5.

[31] Y. LeCun and C. Cortes, "Mnist handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010.

[32] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[33] "Reuters-21578 dataset," http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html, 2021.

To Tackle Cost-Skew Tradeoff: An Adaptive Learning Approach for Hub Node Selection

Guowei Sun*[2], Lin Chen*[1], Qiming Huang[3], Hu Ding†[1]
[1]School of Computer Science and Technology, [2]School of Data Science, [3]School of the Gifted Young
University of Science and Technology of China, Hefei 230026, China
{gwsun1998, ischenlin, chnhqm}@mail.ustc.edu.cn, huding@ustc.edu.cn

***Abstract*—In chip design, *skew* is a pivotal factor that significantly influences the overall performance for routing. A major challenge is how to achieve an appropriate trade-off between the total wire-length cost and skew. Selecting hub nodes is an effective method to improve this cost-skew trade-off. In this paper, we propose a novel reinforcement learning-based method for hub node selection, where our key idea is leveraging an effective adaptive learning strategy. Moreover, our approach is particularly suitable for solving large-scale routing instances. The empirical results suggest that our method can achieve promising performance on both small-scale and large-scale clock nets, implying its potential practical significance in EDA.**

***Index Terms*—Skew, wirelength, MSPD.**

I. Introduction

In the past decades, the problem of designing efficient algorithms for various VLSI routing applications has attracted substantial attention [1]–[6]. The routing quality are closely related to several crucial metrics in chip design, which are the so-called "*PPA*", i.e., *performance*, *power*, and *area* [7]. The design procedure of interconnect trees relies on several widely studied optimization objectives, among which the total tree **cost** is the most prevalent, typically measured by wire length. Additionally, the **skew**, defined as the difference in signal travel time between the longest and shortest root-leaf paths, is also a crucial factor, as illustrated in Fig. 1. For different applications, these objectives may have different priorities.

In this paper, we focus on designing an effective routing algorithm for tackling the tradeoff between the wire-length and skew. For chip design, the "clock frequency" is an important performance indicator, as it determines the number of instructions that the processor can execute per second [8]. The clock frequency is the reciprocal of the clock cycle, which can be affected by several different factors in practice. In particular, the clock skew often seriously influences the clock cycle; as noted in [9], it can account for even more than 10% of the cycle time. Consequently, how to effectively address the clock skew issue is an imperative problem that we are confronted with for designing high-quality chips.

Actually, the skew issue is a long historical topic in the research of EDA. Several classical methods were proposed before, such as the methods of *Zero-Skew Tree (ZST)* and *Bounded-Skew Tree (BST)* [8]. The goal of ZST is to build a routing tree where the signal from the root node can reach all the terminals at the same time, so that the skew is zero. Boese and Kahng [10] presented the Deferred-Merge Embedding (DME) algorithm for building a ZST that can accomplish precise zero skew. However, a drawback of ZST is that it often results in long wire-length and it is difficult to be realized due to the limitations of the manufacturing process [8]. This motivates researchers to consider an alternative solution, Bounded-Skew Tree [11]–[13], which is a clock tree with a maximum skew bounded by a given threshold $UB > 0$. It is worth noting that when there is no skew restriction (i.e., $UB = \infty$), the BST problem is simply equivalent to the classical Rectilinear Steiner Minimum Tree (RSMT) problem [14]. Several heuristic methods were developed to address the bounded skew Steiner tree construction problem in the past decades [12], [13]. Cong et al. [11] provided an improved DME algorithm called "BST-DME", which is specifically designed for the BST problem. Additionally, several elegant approximation algorithms were also proposed for the skew problem [15], [16]. More recently, Han et al. [17] studied the tradeoff between cost and skew and formulated the bounded skew problem as a flow-based integer linear programming.

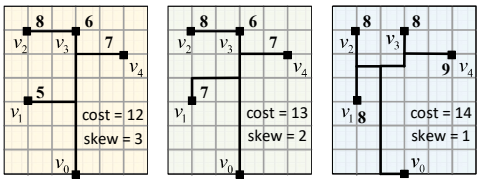


Fig. 1: This illustration shows that different routing priorities can lead to various topologies; v_0 is designated as the root node, and the number marked alongside each vertex is the distance to v_0 in the tree. Figure (a) depicts a design focusing on minimizing wire length [8], while Figure (b) illustrates a trade-off between cost and skew. Figure (c) primarily emphasizes skew, aiming to minimize the time difference for signals traveling from the root node to other nodes.

Recently, building upon the classic Prim-Dijkstra (PD) algorithm [18], Kahng et al. [19] introduced the **Multi-Source Prim-Dijkstra (MSPD)** algorithm for the cost-skew tradeoff problem, which will be detailed in Section II-B. The main insight of the MSPD approach is to select several centrally located "hub" nodes (termed "sources" in their paper) to serve as root connections and enforce a constraint that prevents other nodes from connecting directly to the root. Subsequently, the

*First two authors contributed equally. †Corresponding author. This work was supported in part by the Fundamental Research Funds for the Central Universities through Grant WK2150110025.

modified PD algorithm (i.e., "MSPD") is employed to generate the tree topology. From the examples illustrated in Fig. 2, we can see the advantage of utilizing hub nodes.

However, there is still a gap for efficiently identifying the hub nodes. **The first challenge** comes from the high computational cost, since an optimal algorithm would require an exhaustive search for the best combination of hub nodes. Although the method proposed in [19] could reduce the complexity to a certain extent by only considering those centrally located node candidates, the resulting smaller solution space could affect the final performance of routing. Moreover, **the second challenge** is that there is currently no effective method capable of adaptively deciding how many hub nodes should be chosen. The existing algorithms require specifying a maximum quantity of hub nodes, and then enumerating different combinations that have the sizes within such an upper bound. Due to the importance of hub nodes in the skew problem, the open-source project *OpenROAD* ("Foundations and Realization of Open, Accessible Design") in collaboration with TILOS AI Institute recently launched an innovation-driven competition in June 2023, whose objective is to devise efficient hub node selection algorithms to achieve favorable cost-skew tradeoff.

To tackle those challenges in addressing the cost-skew tradeoff problem, we propose an adaptive learning-based framework. In particular, our framework takes account of the global space for determining the number of chosen hub nodes and their locations. **The main contributions** of our paper are outlined as follows:

1) We propose a novel "Selector $\rightleftarrows$ Conductor" idea that can be seamlessly integrated with the existing reinforcement learning method. The "Selector" network is for selecting hub nodes, and the "Conductor" network is used for monitoring the current environment and actions to determine when to halt operations. This cooperation mechanism between them yields an adaptive learning process, and also enhances our model's ability to manage larger-scale instances.
2) To evaluate the performance of our proposed method, we conducted a set of experiments on both small-scale and large-scale problems using diverse datasets sourced from synthetic data and the OpenROAD competitions. In the experiments, our learning-based method achieved the best Pareto frontier in terms of cost and skew across various test cases and demonstrated satisfying efficiency.

II. Preliminaries

A. Problem Formulation

An instance of a clock routing problem comprises n nodes, denoted as $V = \{v_0, v_1, \cdots, v_{n-1}\}$, with v_0 acting as the **root node** and other nodes serving as the **sinks**. The goal is to construct a routing tree such that the signal from the root, v_0, can be transmitted to the other nodes. Suppose a routing tree $T = (V, E)$ is constructed to connect all the nodes in V. In this routing tree T, let $e_{i,j}$ denote the edge linking v_i with v_j; its edge cost is defined as the Manhattan distance $|x_j - x_i| + |y_j - y_i|$ (suppose $v_i = (x_i, y_i)$ and $v_j = (x_j, y_j)$). Consequently, the ***cost*** of the clock tree, denoted as cost(T), is the sum of all edge costs, i.e.,

$$\text{cost}(T) = \sum_{e_{ij} \in E} |x_j - x_i| + |y_j - y_i|. \tag{1}$$

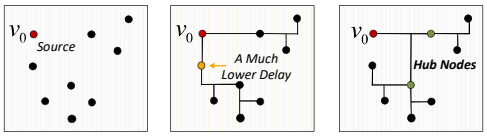
The clock ***skew*** refers to the difference between the highest and lowest arrival times of the clock signal across the $n-1$ sinks. Formally, we define:

$$\text{skew}(T) = \max_{v_i, v_j \in V} |t(v_0, v_i) - t(v_0, v_j)|, \tag{2}$$

where $t(v_0, v_i)$ denotes the signal delay between the nodes v_0 and v_i, estimated by using either the *linear delay* or the *Elmore delay* model [8]. The linear delay model equates delay directly with the path length between v_0 and v_i, ignoring other details in the circuit, whereas the Elmore model is more complex that also takes the resistance and capacitance into account. Due to its simplicity for calculation, the linear model remains popular in commercial tools [8].

As discussed in Section I, a central problem for practical clock routing is designing an effective algorithm for achieving a pleasant **cost-skew tradeoff**. Formally, given a node set V and a parameter $\alpha \in [0, 1]$, our objective is to construct a routing tree T to minimize:

$$\alpha * \text{cost}(T) + (1 - \alpha) * \text{skew}(T). \tag{3}$$



(a) (b) (c)

Fig. 2: This figure illustrates the role of hub nodes. Figure (a) shows the locations of all nodes. Figure (b) depicts a routing tree without adding hub nodes, leading to significant skew; the orange sink experiences a much lower delay than the other black sinks. Figure (c) presents a routing tree generated by the MSPD algorithm, demonstrating a reduction in skew through the aid of hub nodes.

B. Multi-Source Prim-Dijkstra (MSPD)

Multi-Source Prim-Dijkstra (MSPD) [19] is a variant of the PD algorithm [18]. Given a clock routing instance as shown in Fig. 2(a), a major difference of MSPD is that the algorithm selects a set of hub nodes and only these designated nodes can link directly to the root. The purpose of this constraint is to avoid large skew (see the comparison between Fig.2(b) and Fig.2(c)). It is noteworthy that the authors of [19] proposed two algorithms for hub node selection. The first one is an optimal algorithm; however, it incurs a significantly high computational cost due to its exhaustive search for the best combination of hub nodes. To avoid such a large combination space, the authors also introduced an alternative approach that only enumerates the combinations of the potential hub nodes located centrally to the root; the "centrality" is measured by several pre-specified parameters (we refer the reader to

https://github.com/TILOS-AI-Institute/multi-source-Prim-Dijkstra

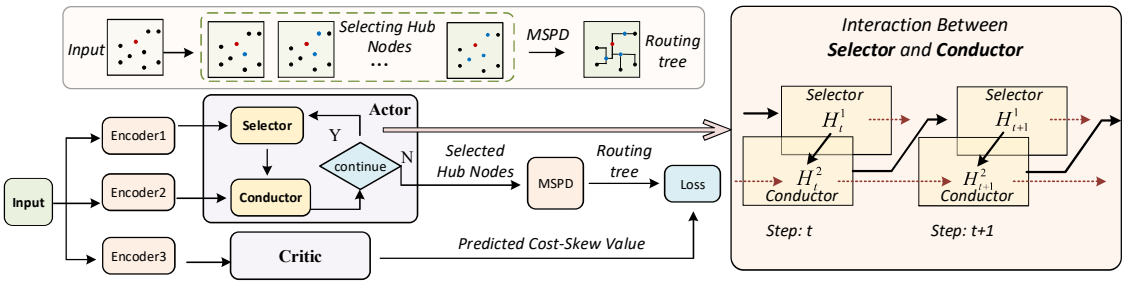


Fig. 3: Our proposed RL-based framework, whose core neural components are the Selector and Conductor networks. The Selector network determines the hub node in each step, while the Conductor network monitors the environment and existing actions, assessing the need for further actions. The right figure illustrates the detailed interaction between these networks; the black arrow indicates the execution order, while the dotted brown arrow represents information passed from previous steps. A running example is also shown in this figure (top left): given an input node set, the red node is the root, and the hub nodes (labeled in blue) can be determined by the Selector and Conductor networks. Then, the instance is processed by the MSPD algorithm and eventually, the routing tree with cost-skew tradeoff is obtained.

their article [19] for more details). Although this method reduces the size of the candidate set, it results in a narrower solution space that could downgrade the obtained solution quality. Moreover, the prior methods usually require specifying a maximum quantity of hub nodes, which may not be easy to determine and thus also restricts the capability to optimize the objective of (3). These practical issues motivate us to develop an efficient and adaptive method for hub nodes selection in Section III.

C. Reinforcement Learning

Our proposed method utilizes Reinforcement learning (RL), which is an important machine learning methodology that has been extensively studied for many different real-world applications [20]. RL uses an **agent** to perform the learning process, where the agent can make decisions by taking actions in a specified environment. This RL process consists of interacting with the environment, evaluating the current state, and receiving rewards based on the outcomes of its actions. In particular, the **Actor-Critic** RL framework [20] has been successfully applied to solve several routing tasks in EDA [4], [5]. The "Actor" network typically involves a sequence of decision-making steps, such as connecting a node to the existing tree; ultimately this network will generate a routing tree. Meanwhile, the "Critic" network is employed to facilitate the training of the Actor network.

For completeness, we introduce the three important terminologies of RL that will be used throughout Section III: 1) **State:** At any given time step, the state includes the locations of all nodes in V, the root node v_0, and the currently selected set of hub nodes; 2) **Action:** At each step t, the agent selects a new hub node that has not been previously chosen; subsequently, the agent must also determine whether to continue selecting additional hub nodes; 3) **Reward:** In our model, the reward is computed at the conclusion of the RL process and is combined with the critic's prediction to assess the quality of the selected hub nodes.

Our method also follows the above overall framework; however, our proposed method needs to select specific hub nodes during the tree construction, necessitating some significant new ideas for this purpose.

III. Our Proposed Framework

In this section, we introduce our proposed approach. First, we analyze the technical challenges and provide an overview of our framework below.

The challenges and our high-level idea. We are aware of several recently proposed RL-based methods for routing problems, such as the method of [4] for minimizing the wirelength and the method of [5] for minimizing the timing-driven objective. Nevertheless, we are still confronted with some new significant challenges in addressing the issue of cost-skew tradeoff. Both of those methods are designed for generating the tree using a Markov chain, with the aid of some clearly defined termination criteria. But in our problem, our aim is to select several key hub nodes. The immediate issues for realizing such a model include how to handle the uncertainty of the hub nodes and how to overcome the absence of a predefined termination criterion (note that even the number of needed hub nodes is also unknown in advance). Thus, our task is to design some flexible and easy-to-implement method that can adaptively select the hub nodes and efficiently construct the routing tree with a qualified cost-skew tradeoff.

Our high-level idea relies on a novel "**Selector** $\rightleftarrows$ **Conductor**" reinforcement learning framework. To identify the most appropriate candidates for hub nodes, we design a "Selector" network that is dedicated to hub node selection. More importantly, we also integrate a "Conductor" module into the classical Actor module, which can help the model monitor the environment and existing actions, so as to determine the necessity of further actions. The architecture of our whole model is built upon this adaptive Selector $\rightleftarrows$ Conductor mechanism, as shown in Fig. 3.

We present the technical details in the following parts. In Section III-A, we introduce the Encoder module that is used for representing the input data. The "Selector" and "Conductor" modules are two key parts to realize the adaptive learning in our framework, which are respectively introduced

in Section III-B and Section III-C. Finally, we describe the Critic network and the training method in Section III-D.

Superscripts of the notations. For ease of presentation, in the following sections, we use the superscripts from $\{0, 1, 2, 3\}$ to differentiate the parameters and variables in different modules. For example, W^0 denotes the parameters in the Encoder modules, W^1 is for the Selector network, W^2 is for the Conductor network, and W^3 is for the Critic network.

A. Encoder

The goal of the Encoder is to embed an input instance V (which contains n vertices in $2D$) into a higher dimensional space $\mathbb{R}^d$ with some appropriately set value d; this new representation will serve the following modules as shown in Fig. 3. Our Encoder model is based on the seminal **attention mechanism** in deep learning [21], which is inspired by human visual attention, enables the model to weigh the importance of different inputs and allows the model to focus on different parts of the input sequence. Note that we have three encoders for the Selector, Conductor, and Critic networks, respectively (as shown in Fig. 3), where they have the same structure but with different trained parameters.

Given the routing instance V containing n nodes, we build the input to our encoder that is the form of a matrix $X_{\texttt{ini}} \in \mathbb{R}^{n\times 3}$. The 3-dimensional information includes the 2D coordinate information (x, y) and a label indicating whether it is a root node (labeled as 1 if it is a root node, and 0 otherwise). Then we embed these nodes into a higher dimensional space for enriching their encoded information: the input vector $X_{\texttt{ini}} \in \mathbb{R}^{n\times 3}$ is processed through a feed-forward neural network layer, obtaining the output $\hat{X}_{\texttt{ini}} \in \mathbb{R}^{n\times d_0}$ (where the integer d_0 is a pre-specified hyperparameter in our model). Formally, our feed-forward neural network can be calculated as:

$$\hat{X}_{\texttt{ini}} = \max\left\{\vec{0}, X_{\texttt{ini}} \times W_1^0 + \vec{b_1^0}\right\} \times W_2^0 + \vec{b_2^0}, \tag{4}$$

where $\vec{0}$ is the zero vector, and $W_1^0 \in \mathbb{R}^{3\times d_0}, W_2^0 \in \mathbb{R}^{d_0\times d_0}, \vec{b_1^0} \in \mathbb{R}^{d_0}, \vec{b_2^0} \in \mathbb{R}^{d_0}$ are the parameters to be trained in this feed-forward layer. Considering the importance of the root node, we need to ensure that the information of the root node v_0 should be accessible to other nodes. For this purpose, we concatenate the embedding of the root node with that of each other node: the obtained new embedding is $X \in \mathbb{R}^d$ ($d = 2d_0$), and each row

$$X[i] = \left[\hat{X}_{\texttt{ini}}[i], \hat{X}_{\texttt{ini}}[0]\right] \tag{5}$$

is the augmented embedding of the node v_i, where $\hat{X}_{\texttt{ini}}[i]$ is the i-th row of $\hat{X}_{\texttt{ini}}$, i.e., the original embedding of v_i, and $\hat{X}_{\texttt{ini}}[0]$ is the original embedding of the root.

Then, the embedding X is fed into the "multi-head" attention layer [21]. The function of this layer is to aggregate and capture information from all the nodes. Through this multi-head attention layer, the information from various nodes is aggregated to create a new representation for each node, and thus an enriched embedding can be obtained. As mentioned at the beginning of this subsection, we have this Encoder network before each of the downstream Selector, Conductor, and Critic networks. We denote the corresponding outputs from these three Encoders as E^1, E^2, and $E^3 \in \mathbb{R}^{n\times d}$, respectively.

B. Selector Network

The function of the Selector network is to identify the nodes that have high potential to become the hubs. The methodology for identifying hub nodes leverages the **Pointer Network** [22], which has shown significant promise in addressing various combinatorial optimization problems with sequence generation tasks, such as the planar Travelling Salesman Problem (TSP) [22] and the Rectilinear Minimum Spanning Tree (RMST) problem [4].

Before diving into the details of our Selector and Conductor networks, we illustrate this **Selector** $\rightleftarrows$ **Conductor** cooperation mechanism in Fig. 3 (right). To assist the Selector and Conductor in making decisions, we define hidden states $H_t^1 \in \mathbb{R}^{d_1}$ and $H_t^2 \in \mathbb{R}^{d_1}$ (initially set to $\vec{0}$) to encode the environment at step t, where d_1 is a hyperparameter. At step t, the Selector network selects a new hub node based on the hidden state H_t^1. The action of selecting the hub node is encoded and can be passed to the Conductor to update its hidden state, resulting in H_t^2, thereby assisting the Conductor in making decisions. If the Conductor chooses to continue selecting hub nodes, the Selector progresses to step $t + 1$. It is important to note that the encoding of the action at step t will be employed to update the hidden state of the Selector, yielding H_{t+1}^1. Subsequently, the Selector will use H_{t+1}^1 to select hub nodes as described above.

Now, we introduce more details for the Selector and leave the details for the Conductor to Section III-C. At step t, leveraging H_t^1, the probabilities of those n nodes being selected as the hub node can be obtained by the following n-dimensional vector:

$$p_t^{\texttt{sel}} = \text{Softmax}\left(\tanh\left(\text{Mask}\left(z_t^1\right)\right)\right), \tag{6}$$

where the value $z_t^1 = {g^1}^\top \tanh\left(W_1^1 {E^1}^\top + W_2^1 H_t^1\right)$ with $W_1^1 \in \mathbb{R}^{d_2\times d}$, $W_2^1 \in \mathbb{R}^{d_2\times d_1}$, and $g^1 \in \mathbb{R}^{d_2}$ being the parameters to be trained (the integer d_2 is a pre-specified hyperparameter), and E^1 represents the node embeddings for the Selector network. The computation mechanism of Eq. 6 is based on the Pointer Network [22], and let us explain it in detail. The value $z_t^1 \in \mathbb{R}^n$ represents the logits for selecting different nodes. For the nodes that have already been selected, we use $\text{Mask}(z_t^1)$ to set their values to $-\infty$, thereby preventing them from being re-selected. The "softmax" function normalizes this vector and outputs the probabilities of those n nodes to be selected, i.e., the vector $p_t^{\texttt{sel}} \in \mathbb{R}^n$. Note that during the training phase, selections are made according to the probability distribution defined by $p_t^{\texttt{sel}}$, whereas in the testing phase, the model directly chooses the node with the highest probability.

Suppose the Selector network selects the i-th node ($1 \le i \le n-1$). It is important to embed this action to assist subsequent decisions. To achieve this, we use $A_t^1 \in \mathbb{R}^{d_1}$ to encode the selection action. Since the selected hub node will

be connected to the root node, the encoding of the i-th node $E^1[i] \in \mathbb{R}^d$ should be combined with the encoding of the root node $E^1[0]$:

$$A_t^1 = W_a^1 E^1[i] + W_r^1 E^1[0], \tag{7}$$

where $W_a^1 \in \mathbb{R}^{d_1 \times d}$ and $W_r^1 \in \mathbb{R}^{d_1 \times d}$ are the parameters to be trained. $W_r^1 E^1[0]$ and $W_a^1 E^1[i]$ represent the transformed embeddings of the root node and the currently selected hub node, respectively. After the action is encoded, the hidden state of the Selector for step $t+1$ is updated as follows:

$$H_{t+1}^1 = \max\left(0, \max\left(H_t^1, W_u^1 A_t^1\right)\right), \tag{8}$$

where $W_u^1 \in \mathbb{R}^{d_1 \times d_1}$ is the parameter to be trained. The new hidden state H_{t+1}^1 is obtained by integrating the previous state H_t^1 with the current action encoding $W_u^1 A_t^1$.

C. Conductor Network

The Conductor network monitors the environment and assesses existing actions to determine whether to select additional hub nodes. Similar to the Selector, it also maintains a hidden state H_t^2 that records information from previous actions. At step t, the Conductor needs to perceive the action made by the Selector; thus, the selection action of the Selector at this step should be encoded to facilitate better decision-making by the Conductor. We employ the same method as illustrated in Eq. 7 and Eq. 8 to encode the action and update the hidden state of the Conductor, but with different trained parameters.

At step t, based on the hidden state H_t^2, we compute:

$$z_t^2 = {g^2}^\top \tanh\left(W_1^2 {E^2}^\top + W_2^2 H_t^2\right), \tag{9}$$

where $W_1^2 \in \mathbb{R}^{d_2 \times d}$, $W_2^2 \in \mathbb{R}^{d_2 \times d_1}$, and $g^2 \in \mathbb{R}^{d_2}$ are parameters to be trained, and E^2 represents the node embeddings for the Conductor network. The term $z_t^2 \in \mathbb{R}^n$ is expected to produce a vector that integrates information from the hidden state H_t^2 and the node embeddings. Based on z_t^2, the probability of whether to continue selecting hub nodes at step t can be computed as:

$$p_t^{\texttt{con}} = \text{Sigmoid}\left(\vec{\mathbf{1}} \cdot \left(\tanh\left(z_t^2\right)\right)\right), \tag{10}$$

where $\vec{\mathbf{1}} \cdot \left(\tanh\left(z_t^2\right)\right)$ represents the sum of the elements of $\tanh\left(z_t^2\right)$. Eq. 10 aggregates information from different nodes to inform the decision on whether to continue selecting a hub node. If $p_t^{\texttt{con}} > 0.5$, the process proceeds to the next step; otherwise, the Selector will stop.

D. Critic and Training Method

In reinforcement learning, particularly in an Actor-Critic model, the Critic module is often used to assist the learning process [20]. Different from the previous RL-based networks [4], [5], our Critic module tries to predict the "cost-skew tradeoff value" of the input instance V, where the predicted value is denoted as $\texttt{Cr}(V)$. Recall that our Critic network receives the input $E^3 \in \mathbb{R}^{n \times d}$ from the Encoder, as described in Section III-A. Initially, we compute an "attention-weighted" representation of E^3: $\text{Attn}(E^3) = \text{softmax}\left(\tanh\left(E^3\right) g^3\right)^\top E^3$, where $g^3 \in \mathbb{R}^d$ is a trainable parameter vector, and $\text{Attn}(E^3)$ represents a weighted sum of the encoded information from different nodes. Then, two fully connected layers are utilized to obtain the predicted cost-skew value:

$$\texttt{Cr}(V) = \max\left(0, \text{Attn}(E^3) W_1^3 + \vec{b_1^3}\right) W_2^3 + b_2^3, \tag{11}$$

where $W_1^3 \in \mathbb{R}^{d \times d_3}$, $\vec{b_1^3} \in \mathbb{R}^{d_3}$, $W_2^3 \in \mathbb{R}^{d_3 \times 1}$, and $b_2^3 \in \mathbb{R}$ are trainable parameters (with d_3 being a hyperparameter). After obtaining $\texttt{Cr}(V)$, we introduce the training method of the Actor and Critic networks.

Let θ_1 and θ_2 denote the parameters for the Actor and Critic, respectively. Suppose the batch size is $N \geq 1$ and the set of sampled instances for training are $\mathcal{V} = \{V_1, V_2, \cdots, V_N\}$. For each $1 \leq i \leq N$, assume $\mathcal{T}_i$ is the trajectory executed by the Actor, which follows a Markov decision process. Each step consists of selecting a hub node and deciding whether to continue. The probability of trajectory $\mathcal{T}_i$ under the Actor's policy is denoted as $p_{\theta_1}(\mathcal{T}_i)$. This probability is calculated as the product of the probabilities of the actions taken, including selecting hub nodes and making continuation decisions. The cost-skew value of the solution produced by the Actor is denoted by $\texttt{Ac}(V_i, \mathcal{T}_i)$ for each i. The loss for the Actor, denoted as $\mathcal{L}_{\text{A}}(\theta_1 \mid \mathcal{V})$, is calculated by:

$$\frac{1}{N} \sum_{i=1}^{N} \left[\texttt{Cr}(V_i) - \texttt{Ac}(V_i, \mathcal{T}_i) + \sum_{t=1}^{|\mathcal{T}_i|} R_t \gamma^t\right] \log p_{\theta_1}(\mathcal{T}_i | V_i), \tag{12}$$

where $|\mathcal{T}_i|$ denotes the number of hub nodes chosen by the Actor in trajectory $\mathcal{T}_i$. R_t and γ respectively represent the intermediate reward and the decay factor, which are used to balance the exploration and exploitation in RL [20].

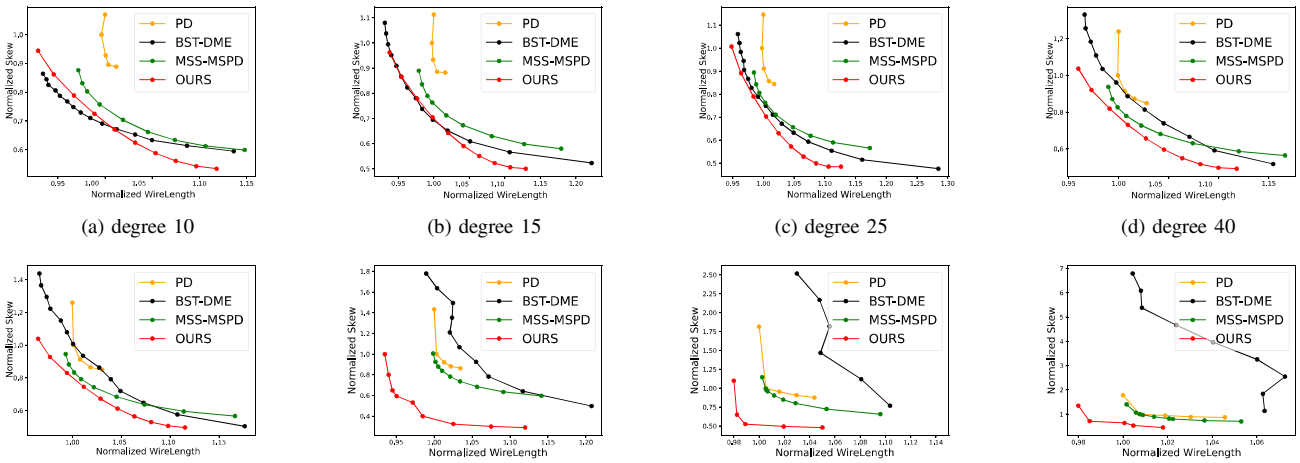
The loss for the Critic $\mathcal{L}_{\text{C}}(\theta_2 | \mathcal{V})$ is based on the Mean Squared Error (MSE) function:

$$\frac{1}{N} \sum_{i=1}^{N} \|\texttt{Cr}(V_i) - \texttt{Ac}(V_i, \mathcal{T}_i)\|_2^2. \tag{13}$$

Both the Actor and Critic networks are trained using the Stochastic Gradient Descent (SGD) method [23].

IV. EXPERIMENTAL RESULTS

We trained our model using PyTorch and conducted our experiments on a 64-bit Linux system equipped with eight NVIDIA RTX A6000 GPUs. We compare the performance of our algorithm with three representative baselines: (1) the "BST-DME" algorithm outlined in [11]; (2) a heuristic **m**ulti-**s**ource **s**election method introduced in [19], denoted as "MSS-MSPD"; (3) the "PD" algorithm [18], [24] introduced in Section II-B. Our algorithm is labeled as "OURS". We present our results separately for small-scale ($n \leq 50$) and large-scale ($n > 50$) instances.



(a) degree 10 (b) degree 15 (c) degree 25 (d) degree 40

(e) degree 50 (f) degree 100 (g) degree 500 (h) degree 1000

Fig. 4: Figures (a) to (e) show the comparison results of our framework with the BST-DME, MSS-MSPD, and PD algorithms for data with degrees of 10, 15, 25, 40, and 50, respectively. Figures (f) to (h) show the comparison results for large degrees of 100, 500, and 1000, respectively.

A. Small-Scale

We trained our models for degrees ranging from 10 to 50 on randomly generated training data. For degree 10, we trained the model with mini-batch size B = 2048 for $100k$ iterations. Subsequently, the batch size was diminished for each ascending degree. In this part, we compare the experimental performance for nets with degrees $\leq$ 50, using the test data (300 instances for each degree) derived from the recent OpenROAD contest [25], which is generated by a well-designed random algorithm, ensuring the diversity of distribution.

The Prim's algorithm and Dijkstra's algorithm are commonly used representative algorithms for optimizing cost and skew [18], so we particularly take their obtained values "$\mathbf{cost}(\mathbf{T_P})$" and "$\mathbf{skew}(\mathbf{T_D})$" as the baseline values, where T_P and T_D are the trees obtained by Prim's and Dijkstra's algorithms, respectively. Suppose T is the tree obtained by an algorithm (either our method or a baseline), then we evaluate this algorithm's performance by the ratios $\frac{cost(T)}{cost(T_P)}$ and $\frac{skew(T)}{skew(T_D)}$, where are respectively referred as "**normalized wirelength**" and "**normalized skew**". The comparative performance of these algorithms is illustrated in Fig. 4 (a)-(e). It can be observed that our algorithm achieves the best Pareto frontiers, and the performance difference between "OURS" and the other algorithms gradually increases as the scale increases. To more clearly illustrate this trend, we also conducted the experiment on larger scale instances in the next section.

B. Large-Scale

We compare the algorithms on randomly generated large-degree networks (degrees of 100, 500, and 1000, with 100 instances for each degree). The results are shown in Fig. 4 (f)-(h). It can be seen that the BST-DME algorithm has relatively poor performance on large-scale instances. Since the BST-DME algorithm has to obey skew constraints, the wirelength cost caused by this constraint accumulates rapidly with the expansion of scale, and our framework maintains a superior performance relative to other algorithms. It is worth noting that we directly use the model trained in degree 100 for degrees 500 and 1000 without fine-tuning, which implies the effectiveness of applying our model to large-scale problems. The runtimes of various algorithms across different degrees are shown in Table I. Since most of the computational load of our learning-based model belongs to the training stage, the running time of solving a new instance in the test stage is quite small. Our runtime is comparable with PD and slightly slower than BST-DME, but our algorithm enjoys significantly better performance than these two baselines as shown in Fig. 4.

TABLE I: Average runtime performance over 1000 samples for small-scale tests and over 100 samples for large-scale tests.

Degree	BST-DME(s)	MSS-MSPD(s)	PD(s)	OURS(s)
10	0.022	0.007	0.001	0.003
15	0.045	0.074	0.005	0.007
25	0.057	0.107	0.009	0.009
40	0.081	0.240	0.014	0.015
50	0.094	0.400	0.018	0.019
100	0.202	2.575	0.103	0.104
500	1.216	136.6	1.387	1.415
1000	2.480	1008	4.392	5.362

V. Conclusion

In this paper, we propose an adaptive learning-based approach to select hub nodes for the cost-skew tradeoff problem. The key idea relies on a novel Selector $\rightleftarrows$ Conductor cooperation mechanism integrated with the RL framework. Our experimental results illustrate the promising performance of our model on small and large-scale nets. In the future, we can consider further speeding up our method through other aspects. For example, it is interesting to consider whether the MSPD algorithm can be implemented on GPUs. It is also worth investigating whether our adaptive "Selector $\rightleftarrows$ Conductor" RL framework can be applied to solve other challenging optimization problems in EDA.

REFERENCES

[1] M. Hanan, "On steiner's problem with rectilinear distance," *SIAM Journal on Applied mathematics*, vol. 14, no. 2, pp. 255–265, 1966.

[2] C. Chu and Y.-C. Wong, "Flute: Fast lookup table based rectilinear steiner minimal tree algorithm for vlsi design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 1, pp. 70–83, 2007.

[3] M. D. Moffitt, "Maizerouter: Engineering an effective global router," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 11, pp. 2017–2026, 2008.

[4] J. Liu, G. Chen, and E. F. Young, "Rest: Constructing rectilinear steiner minimum tree via reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 1135–1140.

[5] L. Yang, G. Sun, and H. Ding, "Towards timing-driven routing: An efficient learning based geometric approach," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[6] R. Cheng, X. Lyu, Y. Li, J. Ye, J. Hao, and J. Yan, "The policy-gradient placement and generative routing neural networks for chip design," *Advances in Neural Information Processing Systems*, vol. 35, pp. 26 350–26 362, 2022.

[7] V. Soteriou, N. Eisley, H. Wang, B. Li, and L.-S. Peh, "Polaris: A system-level roadmap for on-chip interconnection networks," in *2006 International Conference on Computer Design*. IEEE, 2006, pp. 134–141.

[8] A. B. Kahng, J. Lienig, I. L. Markov, and J. Hu, *VLSI physical design: from graph partitioning to timing closure*. Springer, 2011, vol. 312.

[9] M. R. Guthaus, G. Wilke, and R. Reis, "Revisiting automated physical synthesis of high-performance clock networks," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, pp. 1–27, 2013.

[10] K. D. Boese and A. B. Kahng, "Zero-skew clock routing trees with minimum wirelength," in *[1992] Proceedings. Fifth Annual IEEE International ASIC Conference and Exhibit*. IEEE, 1992, pp. 17–21.

[11] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and steiner routing," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 3, no. 3, pp. 341–388, 1998.

[12] D. J. Huang, A. B. Kahng, and C.-W. A. Tsao, "On the bounded-skew clock and steiner routing problems," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*, 1995, pp. 508–513.

[13] A. B. Kahng and C.-W. A. Tsao, "Practical bounded-skew clock routing," *High Performance Clock Distribution Networks*, pp. 87–103, 1997.

[14] M. R. Garey and D. S. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[15] A. Z. Zelikovsky and I. I. Mandoiu, "Practical approximation algorithms for zero-and bounded-skew trees," *SIAM Journal on Discrete Mathematics*, vol. 15, no. 1, pp. 97–111, 2001.

[16] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai, and A. Tomkins, "Minimizing wirelength in zero and bounded skew clock trees," *SIAM Journal on Discrete Mathematics*, vol. 17, no. 4, pp. 582–595, 2004.

[17] K. Han, A. B. Kahng, C. Moyes, and A. Zelikovsky, "A study of optimal cost-skew tradeoff and remaining suboptimality in interconnect tree constructions," in *Proceedings of the 20th System Level Interconnect Prediction Workshop*, 2018, pp. 1–8.

[18] C. J. Alpert, T. C. Hu, J.-H. Huang, A. B. Kahng, and D. Karger, "Prim-dijkstra tradeoffs for improved performance-driven routing tree design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 7, pp. 890–896, 1995.

[19] A. B. Kahng, S. Thumathy, and M. Woo, "An effective cost-skew tradeoff heuristic for vlsi global routing," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2023, pp. 1–8.

[20] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[22] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in neural information processing systems*, vol. 28, 2015.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu, and S. Venkatesh, "Prim-dijkstra revisited: Achieving superior timing-driven routing trees," in *Proceedings of the 2018 International Symposium on Physical Design*, 2018, pp. 10–17.

[25] TILOS-AI-Institute, "Mspd," Github, 2023. [Online]. Available: https://github.com/TILOS-AI-Institute/multi-source-Prim-Dijkstra

HPIM-NoC: A Priori-Knowledge-Based Optimization Framework for Heterogeneous PIM-Based NoCs

Shuai Yuan[1], Angxin Cai[1], Qiushi Lin[2], Guoxing Wang[1], Yu Wang[2], Zhenhua Zhu[2], and Yanan Sun[1]
[1]Department of Micro-Nano Electronics, Shanghai Jiao Tong University, Shanghai, China
[2]Department of Electronic Engineering, Tsinghua University, Beijing, China
Corresponding Authors: {sunyanan@sjtu.edu.cn, zhuzhenhua@mail.tsinghua.edu.cn}

Abstract*—Network-on-Chip (NoC) accelerators with heterogeneous Processing-in-Memory (PIM) cores achieve superior performance than homogeneous ones for neural networks. Dedicated simulators and architecture search frameworks are pivotal for obtaining performance, power, and area (PPA) metrics, as well as guiding the design process. However, existing simulators are primarily designed for homogeneous NoC and lack support for simulating heterogeneous PIM-based NoC architectures. Besides, current search frameworks for heterogeneous NoC architectures only focus on workload allocation and mapping strategies, failing to explore heterogeneous PIM configurations in a larger design space. In this work, we propose HPIM-NoC, a joint simulation and search framework for heterogeneous PIM-based NoC architectures. HPIM-NoC not only supports the simulation of heterogeneous PIM cores, but also provides more accurate latency results by introducing NoC transmission delays and pipelines in co-simulation. HPIM-NoC implements a three-stage heterogeneous search process based on priori knowledge and employs a specific simulated annealing algorithm tailored for heterogeneous architecture search. The search process is accelerated by precomputing core PPA metrics and reducing NoC simulation frequency. In addition, the framework integrates a customized layout algorithm to optimize the placement of heterogeneous NoC, minimizing communication latency and overall area. Experimental results on various neural networks demonstrate that HPIM-NoC can quickly find near-optimal configurations within a limited time. The proposed acceleration method reduces the search time of HPIM-NoC by 2.12×, 2.17×, and 2.96×, respectively. Compared to homogeneous architectures, the Fusions of Metrics (FoMs) of heterogeneous PIM-based NoC architectures found by HPIM-NoC are reduced by 1.18%, 16.94%, and 37.41% for ResNet-18 under three settings, respectively.

***Index Terms*—heterogeneous, PIM, NoC, simulation, architecture search**

I. Introduction

In recent years, Convolutional Neural Networks (CNNs) have shown exceptional capabilities in applications such as object detection and face recognition. However, as CNN models grow more complex, traditional von-Neumann architectures, like CPUs and GPUs, face significant challenges due to the massive data movements between memory and computing units. The data movements consume over 80% of system energy and worsen the "memory wall" problem [1]. To address these challenges, Processing-In-Memory (PIM) architectures [2], [3] have been introduced, performing Matrix-Vector Multiplication (MVM) directly within memory arrays to eliminate data movement overhead and improve energy efficiency.

As the number of neural network parameters increases, single-core PIM accelerators face limitations in computation capacity and still require weight refreshes with significant overhead [4]. Current approaches often design homogeneous or heterogeneous PIM-based NoC accelerators to store all weights on-chip. However, the performance of homogeneous PIM-based NoC architectures is tightly coupled to the PIM types and array sizes of individual cores, which restricts their scalability and flexibility [5]. SRAM-based digital PIM architectures with integrated logic computation [6], [7] exhibit low latency, while RRAM-based analog PIM architectures [8], [9] provide high computational density, despite the overhead of analog-to-digital conversion. Heterogeneous PIM-based NoC architectures, which combine different PIM types and core sizes, offer improved scalability and flexibility to meet stringent power, performance, and area (PPA) requirements. The main challenge in designing heterogeneous PIM-based NoC architectures lies in the vast design space, which makes manual design almost impossible and requires efficient design space exploration. The exploration time depends on the simulation per architecture and the search strategy. In terms of simulation, most existing multi-core simulators like DNN+NeuroSIM v2.0 [10] and MNSIM 2.0 [11] target homogeneous PIM-based architectures and do not support the simulation of heterogeneous architectures. Moreover, existing simulators [10], [11] primarily focus on simulating core-level metrics, lacking accurate models for simulating inter-core communication in NoC. Besides, general-purpose NoC simulators such as Booksim 2.0 [12] cannot model the inter-layer data dependencies of neural networks, resulting in inaccuracies in latency simulation outcomes.

In terms of architectural search, current architecture search frameworks face limitations due to the lack of simulators tailored for heterogeneous PIM-based NoC architectures, which hinders their ability to evaluate diverse hardware configurations efficiently. Frameworks such as PIM-HLS [13] and [14] typically focus on searching for workload allocation and mapping schemes rather than hardware configurations. Other frameworks, like Gibbon [15] and AIG-CIM [16], primarily search for homogeneous configurations or limited ranges of heterogeneous configurations, thus missing the opportunity to explore a wider design space involving multiple PIM types, core sizes, and array configurations. As a result, there is a critical need for dedicated search frameworks to find optimal configurations for heterogeneous PIM-based NoC architectures.

In this paper, we propose HPIM-NoC, a priori-knowledge-based optimization framework for heterogeneous PIM-based

NoC architectures. HPIM-NoC enables the simulation of heterogeneous PIM-based NoC architectures, integrating more comprehensive pipeline modeling. Additionally, HPIM-NoC features an architecture search framework targeting heterogeneous PIM core configurations, utilizing priori knowledge to further accelerate the search process. The contributions include:

- The simulator implemented in HPIM-NoC overcomes the constraints of homogeneous simulators that cannot model heterogeneous core configurations. A heterogeneous architectural prototype is defined and a workload allocation algorithm is introduced for heterogeneous cores. HPIM-NoC models the data transmission in neural networks and enables the simulation of pipelining between core computation and NoC transmission.
- A novel search framework is proposed by employing a customized simulated annealing (SA) [17] algorithm with specific operators and reward function targeting for heterogeneous architectures. A three-stage search process is designed based on priori knowledge, effectively addressing the challenge of the vast search space in heterogeneous architectures. Additionally, a fine-tuning method is introduced to ensure the validity of architectural configurations throughout the three-stage search process. The search process is accelerated by precomputing core PPA metrics and reducing NoC simulation frequency to minimize redundant computations. Furthermore, a customized placement algorithm is designed to optimize heterogeneous NoC layouts, reducing communication latency and overall area.
- Experimental results demonstrate that the Fusions of Metrics (FoMs) of heterogeneous PIM-based NoC architectures found by HPIM-NoC are reduced by 1.18%, 16.94%, and 37.41% for ResNet-18 than homogeneous architectures under the three settings, respectively. The layouts of heterogeneous PIM-based NoC are also obtained with the shortest total wirelength and optimized total area. The results demonstrate the effectiveness of HPIM-NoC in quickly exploring the near-optimal heterogeneous PIM-based NoC architecture within a limited time.

II. Related Works

Currently, PIM simulators play an important role in the design and optimization of neural network accelerators. DNN+NeuroSIM v2.0 [10] and MNSIM 2.0 [11] provide circuit-level and architecture-level modeling homogeneous PIM-based multi-array architectures with various PIM types. However, these works lack the simulation of data transmission on the NoC and pipeline between computation and transmission, leading to potential deviations in simulation results. SIAM [18] provides simulation for homogeneous PIM-based NoC architectures with different mapping schemes and parameters. All of the above simulators are designed for homogeneous PIM-based architectures and cannot support the simulation of heterogeneous PIM-based NoC architectures.

In the field of NoC simulation, Booksim 2.0 [12] is widely used to evaluate the performance of general-purpose NoC architectures. Booksim uses the injection rate to represent the number of packets transmitted per unit of time. During the simulation, random numbers are continuously generated. If the random number exceeds the injection rate, a packet is sent along the designated path, and the transmission latency from sending to receiving is recorded. However, the computational characteristics of neural networks introduce inter-layer data dependencies. The differences in dataflows between general-purpose NoCs and specific NoCs for neural networks lead to deviations in the simulation results of Booksim.

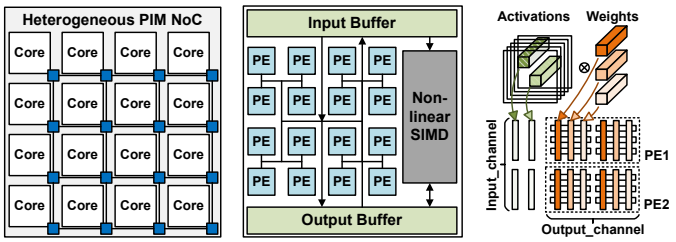


Fig. 1. The prototype of heterogeneous PIM-based NoC architecture for simulation. (a) The heterogeneous PIM-based NoC accelerator. (b) The core in heterogeneous PIM-based NoC. (c) The partitions of activations and weights.

An architecture search framework provides an efficient and systematic method for exploring the vast design space, which is crucial for guiding the design of neural network accelerators. PIM-HLS [13] proposes a dynamic weight scheduling algorithm for scheduling and mapping across heterogeneous PIM arrays. Additionally, [14] optimizes mapping and scheduling strategies by defining communication priorities and NoC communication behavior tables. However, these works only explore mapping and scheduling strategies, without searching for hardware configurations like the PIM types and array sizes. Gibbon [15] automatically searches for the optimal hardware parameters and network structures for homogeneous PIM-based designs. AIG-CIM [16] employs a design space exploration framework with only three different configurations of SRAM-based PIM arrays for heterogeneous PIM NoC accelerator. Therefore, a specific search framework is still lacking to guide the design of heterogeneous PIM-based NoC architectures.

III. The Design of HPIM-NoC

HPIM-NoC consists of a simulator (Section III-A) and a search framework (Section III-B) for heterogeneous PIM-based NoC architectures. Heterogeneous simulation is achieved by calculating the PPA of each core separately and comprehensive pipeline modeling. Based on the heterogeneous NoC simulator, the architecture search framework of HPIM-NoC adopts a priori-based three-stage search approach to reduce the complexity of the heterogeneous search process.

A. Heterogeneous PIM-based NoC Simulation

Existing simulators lack the integration of simulation of heterogeneous cores and NoC transmission, thus failing to provide accurate performance evaluation for heterogeneous PIM-based NoC architectures. A novel simulator is essential to address the challenges of workload allocation, inter-core data transmission, and precise simulation of neural network

TABLE I
CONFIGURABLE PARAMETERS OF THE PROTOTYPE

Configuration Level	Configurable Parameters	Options
Algorithm	Neural Network	Network structure, layer size, etc.
PIM Core	PIM_Type	SRAM, RRAM
	PE_Num	$1 \times 1, 2 \times 2$, etc.
	Array_Size	$32 \times 32, 64 \times 64$, etc.
NoC	Topology	Mesh, Cmesh, etc.
	Core_Num	$1 \times 1, 2 \times 2$, etc.
Mapping	Mapping order	Normal, snake, zigzag

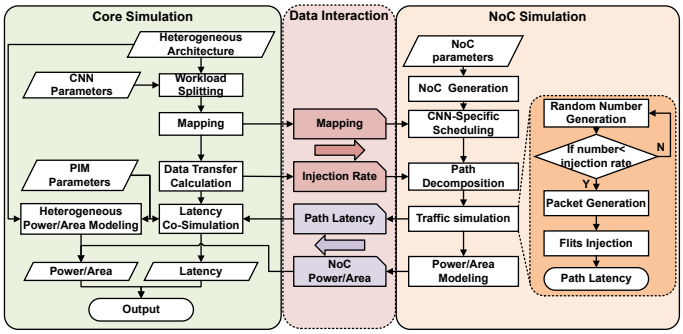


Fig. 2. The proposed simulation flow chart of HPIM-NoC.

data interactions. Before the implementation of the simulator, the prototype of heterogeneous PIM-based NoC architecture is essential to be defined first. As shown in Fig. 1, the prototype architecture can be divided into three levels, including the overall neural network accelerator (Fig. 1 (a)), the heterogeneous PIM core (Fig. 1 (b)), and the PE containing the PIM arrays and peripheral circuits. All configurable parameters of the architecture prototype are listed in Table I. As shown in Fig. 2, the proposed simulator of HPIM-NoC consists of three main components: intra-core simulation, NoC data transmission simulation, and data interaction. Based on the configurations of neural networks and heterogeneous architectures, the workload mapping and inter-core data transmission paths are generated. The PPA results of interconnection circuits are obtained by the NoC simulation, and then jointly simulated with the simulation results of each heterogeneous PIM core.

To simulate the heterogeneous PIM cores accurately, the core simulation must be flexible enough to account for various PIM types and configurations. The core simulation supporting heterogeneous PIM architectures is implemented based on the homogeneous PIM simulator MNSIM [11]. The partition of Weights and activations according to the cores and PEs is shown in Fig. 1 (c), then the data transfer volume between cores is obtained. Differences between the capacity of cores prevent the direct application of load-balancing methods from homogeneous systems to heterogeneous architectures. To address this, a backtracking algorithm (Algorithm 1) is designed to solve the workload allocation and mapping challenges for heterogeneous cores. Simulations are performed on each core and PPA results are obtained.

Existing NoC simulators, such as Booksim [12], do not support the simulation of CNN-specific inter-layer data transmission. An NoC simulator has been proposed based on Booksim to model the inter-layer data transmissions in CNN. The

Algorithm 1 Backtracking to Allocate Cores for Workload

Input: List of computing cores, workload size $W \times H$.
Output: Indices of selected cores.

```
Initialize a list of core usage, and an empty result list;
function Backtrack()
    Find the first empty spot in the workload grid;
    if no empty spot then
        return True        ▷ All workload is stored in cores
    end if
    for core c_i in unused cores do
        if core c_i can store workload at the empty spot then
            Place c_i in the grid and mark it as used;
            Append i to the result list;
            if Backtrack() then        ▷ Backtracking
                return True
            end if
            Remove c_i and unmark it as used;
        end if
    end for
    return False
end function
if Backtrack() then
    return result list
end if
```

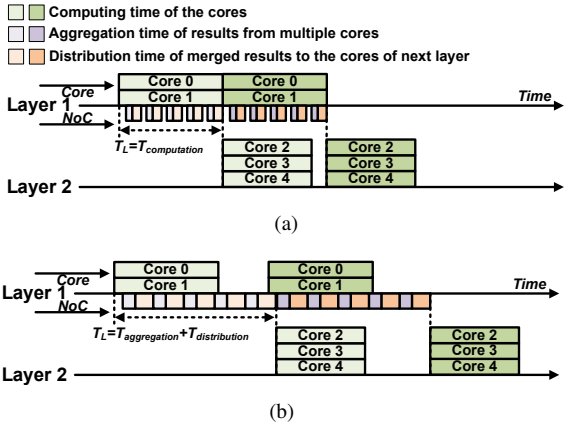


Fig. 3. The pipeline of computation in the cores and data transmission in the NoC. (a) The calculation time is larger than the transmission time. (b) The calculation time is smaller than the transmission time. Different shades of colors correspond to multiple sets of data.

injection rate for path i is calculated by the volume of data transmitted per second and the total bandwidth, as Equation 1:

$$\lambda_i = \frac{A_i \times N_{bits} \times FPS}{C_i \times W}, \tag{1}$$

where A_i represents the number of activations, N represents the data precision, FPS represents the number of frames per second, C_i represents the number of cores to receive the data, and W represents the bandwidth of one path.

The integration of core computation and NoC data transmission is proposed to enable a holistic simulation for both the processing and communication stages. The aggregation latency in each layer and the distribution latency between two adjacent layers in the neural network are both obtained by the maximum latency among multiple paths. The computation of data in cores

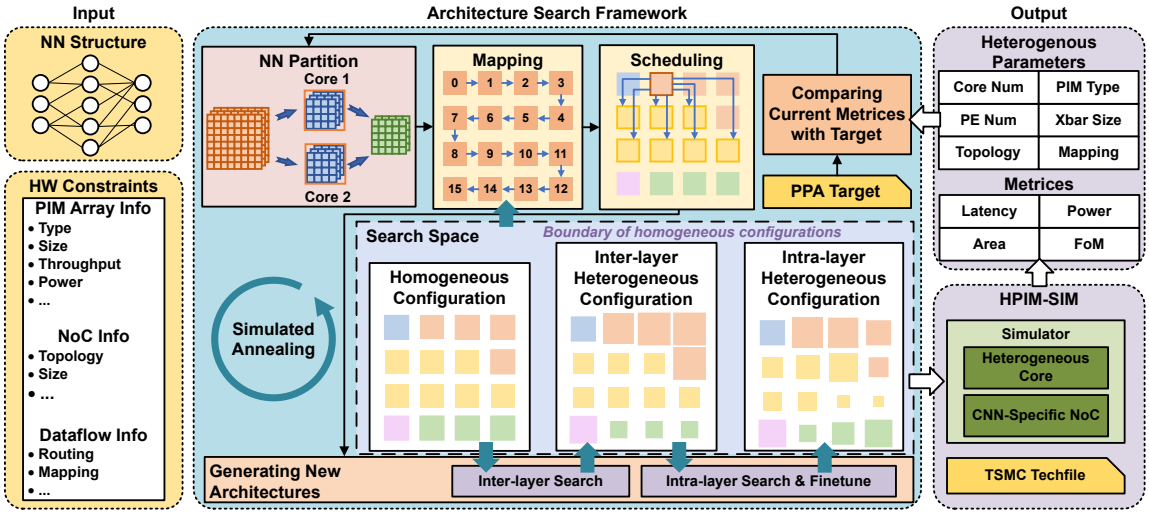


Fig. 4. The proposed search framework of HPIM-NoC.

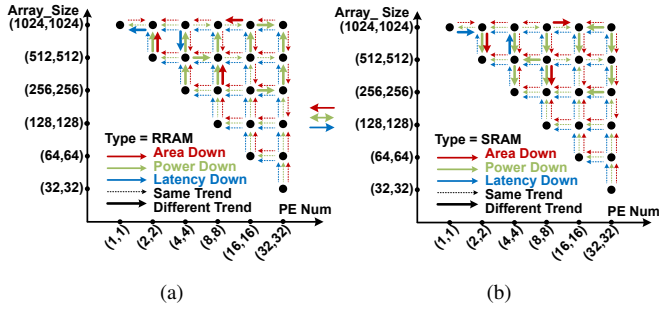


Fig. 5. The changing trend of the PPA results of each core with different configurations. (a) RRAM-based PIM core. (b) SRAM-based PIM core. Only a few RRAM-based and SRAM-based configurations have different trends.

and the transmission in NoC can be performed as the pipeline shown in Fig. 3, while previous simulators [10], [11] default to sequential computation and transmission. The total latency of each neural network layer is equal to the larger value of the calculation time for cores and the transmission time for NoC.

B. Heterogeneous PIM-based NoC Architecture Search

Despite the heterogeneous PIM-based NoC simulator's ability to quickly obtain PPA results for specific architectural configurations, the numerous configurable parameters make it difficult to manually design an NoC that meets the target PPA efficiently. Existing search frameworks designed for homogeneous NoC architectures are unable to effectively navigate the vast design space of heterogeneous architectures. A novel search framework for heterogeneous PIM-based NoC architectures has been proposed based on a customized simulated annealing (SA) algorithm, as shown in Fig. 4.

Due to the need for individual configuration of each computing core in a heterogeneous PIM-based NoC, the search space is significantly larger than that of homogeneous architectures. As the number of cores increases, the exponential growth in configuration possibilities makes it computationally expensive to explore all potential configurations. For example, the number of possible heterogeneous configurations with 16×16 cores can reach up to 10^{476} ($N_{Mapping} \times (N_{PIM_Type} \times N_{PE_Num} \times N_{Array_Size})^{16\times16}$). The enormous search space requires efficient methods to narrow down configurations to those that are most likely to meet target PPA requirements.

Given that heterogeneous architectures are composed of various cores from homogeneous architectures, the proposed HPIM-NoC search framework adopts a priori-knowledge-based approach, utilizing a three-stage search process to narrow the search space and quickly identify a viable architectural solution. The framework first identifies the homogeneous configuration closest to the PPA target as the starting point for further exploration. Since the PPA trends of cores are not strictly linear with respect to Array_Size and PE_Num, the PPA results and variation trends for cores are precomputed, as illustrated in Fig. 5. Subsequently, the search framework proceeds to explore inter-layer heterogeneous architectures by changing cores according to the trends in Fig. 5, applying the same configuration to each PIM core for a given neural network layer. For example, if the current homogeneous configuration (RRAM-based PIM, Array_Size=(256, 256)) gets higher latency than the target PPA, the cores for a specific layer can be adjusted to SRAM-based PIM or Array_Size can be increased to (512, 512) following the trend in Fig. 5. If the desired PPA target remains unmet, the search process continues to investigate intra-layer heterogeneous architectures, employing different configurations for the PIM cores within each layer. The three-step strategy drastically reduces the size of the search space at each stage, speeding up the process while ensuring the effectiveness of the search results. For example, the search space for heterogeneous configurations with 16×16 cores is reduced from 10^{476} to a range of 10^{21} to 10^{43} (largest single-stage search space among the three stages), significantly accelerating the search process.

Then, a specific SA algorithm is designed for heterogeneous PIM-based NoC architectures with a set of operators. Four operators are proposed to modify configurations within the same layer, including changes to the PIM type, PE number, and array size, as well as adjusting the core order using a new mapping scheme. Additionally, the intra-layer search introduces an extra operator to swap two cores within the

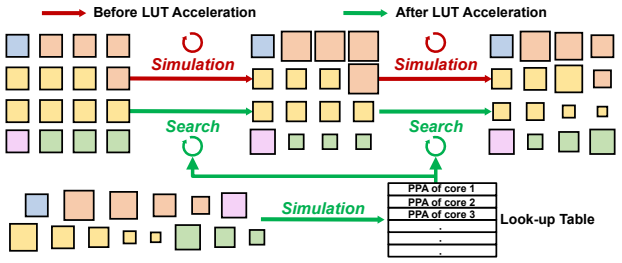


Fig. 6. The searching process before and after acceleration.

same layer. During the intra-layer search process, changing the size of the cores within a given layer may result in the number of weights exceeding the total available capacity. To address this, a method is proposed to finetune the heterogeneous PIM cores within a layer after each operation to ensure the feasibility of the configuration. The total capacity of the cores corresponding to the current neural network layer is calculated, and the number of weights is checked. If the weights exceed the available capacity, additional cores are allocated during inter-layer heterogeneity for that layer. If the weights fit within the capacity but the difference exceeds the smallest core's capacity, the smallest core is removed.

Since the PPA results include three metrics, a composite metric Fusion of Metrics (FoM) is proposed as Equation 2:

$$\text{FoM} = A^{\alpha} \times P^{\beta} \times L^{\gamma}, \tag{2}$$

where A, P, and L represent area, power, and latency respectively. α, β, and γ are user-defined parameters that control the importance assigned to different metrics during the search process. The simulated annealing algorithm employs FoM as the feedback indicator in the search process. The number of operations performed in the SA algorithm decreases with the number of steps. The search concludes when the target PPA is achieved or the search steps are completed.

Due to the simulation required by each step in the search process, the simulation time becomes unacceptable after multiple iterations. Each simulation will re-simulate all cores of the current heterogeneous architecture to obtain PPA metrics, significantly slowing down the overall search speed. To address this, an acceleration method based on the look-up table (LUT) for simulation is proposed. As shown in Fig. 6, the red arrows represent the original search and simulation process, while the green arrows indicate the accelerated process using the lookup table. All possible core configurations are simulated beforehand, and the PPA results are stored. During the search, the pre-computed results are reloaded and accumulated, eliminating redundant core simulations. After accelerating core simulations, NoC simulations become the most time-consuming part of the search process. To further optimize, the frequency of NoC simulations is reduced during the search process by leveraging the trend of data transmission patterns between cores gradually stabilizing during the search process. Finally, a heterogeneous PIM-based NoC placement algorithm is implemented using a force-directed approach. The algorithm ensures a compact arrangement of heterogeneous cores by leveraging the attractive forces along the topological interconnections.

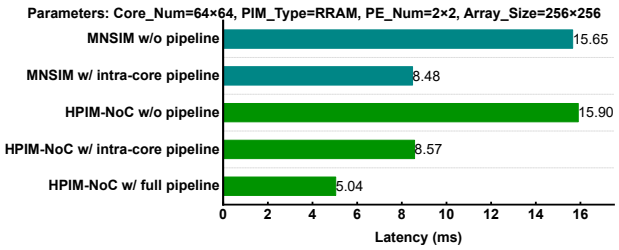


Fig. 7. The simulation results of MNSIM and the proposed HPIM-NoC with different pipeline modes for VGG-8.

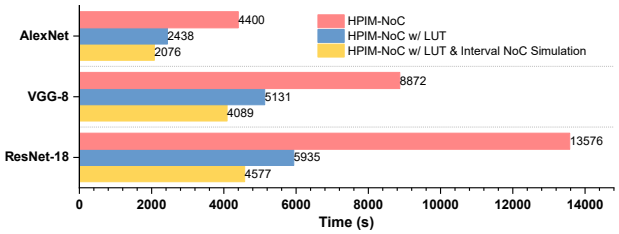


Fig. 8. The search times of the proposed HPIM-NoC with different acceleration methods for ResNet-18 under ($\alpha = 1, \beta = 1, \gamma = 1$). The search step is set to 100 and Core_Num is 24×24 for a fair comparison.

IV. Experimental Results

First, the correctness of the proposed simulator and co-simulation process is validated by comparing it with MNSIM through simulations of homogeneous architectures. MNSIM supports two modes: non-pipeline and intra-core pipeline. The proposed HPIM-NoC simulator supports three modes: non-pipeline, intra-core pipeline, and full pipeline between computation and transmission. The latency results under different pipelining conditions were obtained, as shown in Fig. 7. The results of the HPIM-NoC simulator show only slight differences from MNSIM in the first two modes, indicating the correctness of the HPIM-NoC simulation. The difference arises because MNSIM does not account for the NoC data transmission latency. In the full pipeline mode, the simulator HPIM-NoC gets lower latency than in the intra-core pipeline mode, demonstrating the effectiveness of the co-simulation.

In the search framework, the parameters for the simulated annealing algorithm are set as follows: initial temperature $T_0 = 100$, final temperature $T_f = 0$, and cooling rate $\alpha = 0.99$. The number of operations per step is set as $\text{random}(0, 1) \times \frac{T}{4} + 1$. We tested the search performance under three settings: ($\alpha = 0, \beta = 0, \gamma = 1$) for improved latency, ($\alpha = 0, \beta = 1, \gamma = 1$) for better energy-delay product, and ($\alpha = 1, \beta = 1, \gamma = 1$) for an optimized customized FoM. AlexNet, VGG-8, and ResNet-18 are selected as the CNN models to evaluate HPIM-NoC, while other models are also applicable. ImageNet is selected as the dataset. The acceleration methods tailored for the search process are first tested, with the time of searching architectures for three networks shown in Fig. 8. The proposed acceleration method reduces the search time of HPIM-NoC by 2.12×, 2.17×, and 2.96×, respectively.

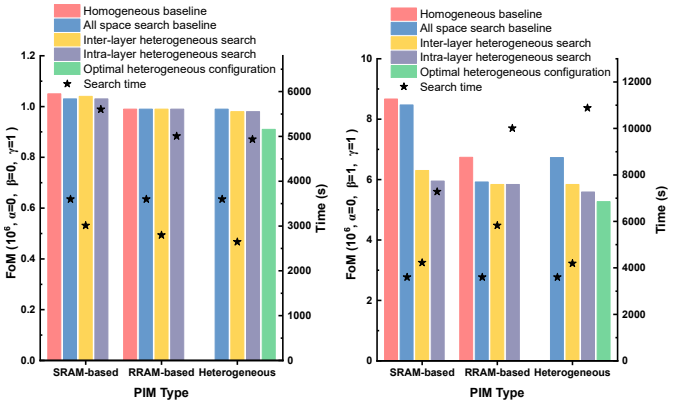
The search results for the three networks are shown in Table II. Due to the inherent randomness of the search process, the resulting topologies, core counts, and configurations of each core vary under different search parameters. HPIM-NoC effec-

TABLE II
THE SEARCH RESULTS OF THE PROPOSED HPIM-NOC FOR DIFFERENT NEURAL NETWORKS.

Neural Network	Search Method	Topology	Core_Num	Area (mm^2)	Power (W)	Latency (ms)	Time (s)
AlexNet	$(\alpha=0,\beta=0,\gamma=1)$	Mesh	13×13	98.95	21.45	0.24	2180
	$(\alpha=0,\beta=1,\gamma=1)$	Mesh	11×11	62.89	11.97	0.27	4322
	$(\alpha=1,\beta=1,\gamma=1)$	Mesh	10×10	73.49	12.99	0.28	3749
VGG-8	$(\alpha=0,\beta=0,\gamma=1)$	Cmesh	16×16	179.09	44.12	1.59	4365
	$(\alpha=0,\beta=1,\gamma=1)$	Mesh	16×16	112.08	21.05	1.51	6990
	$(\alpha=1,\beta=1,\gamma=1)$	Mesh	14×14	129.45	26.32	1.51	6559
ResNet-18	$(\alpha=0,\beta=0,\gamma=1)$	Cmesh	22×22	357.35	86.57	0.98	4936
	$(\alpha=0,\beta=1,\gamma=1)$	Mesh	26×26	321.23	57.72	0.97	10882
	$(\alpha=1,\beta=1,\gamma=1)$	Mesh	24×24	288.69	55.44	0.90	11110

TABLE III
THE SEARCH RESULTS OF THE PROPOSED HPIM-NOC FOR RESNET-18 WITH $(\alpha=1,\beta=1,\gamma=1)$

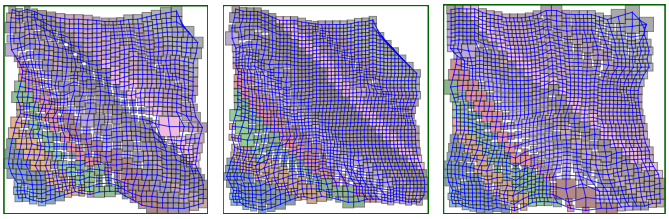
PIM_Type	Search Method	Area (mm^2)	Power (W)	Latency (ms)	FoM (10^{16})	Improvement	Time (s)
SRAM-based	Homogeneous config w/o search	383.01	82.38	1.05	3.32046	/	/
RRAM-based	Homogeneous config w/o search	340.32	63.43	1.06	2.29240	Baseline	/
Heterogeneous	Inter-layer heterogeneous	317.98	60.65	0.92	1.77191	-22.71%	6658
	Intra-layer heterogeneous	**288.69**	**55.44**	**0.90**	**1.43500**	**-37.41%**	**11110**
	Optimal heterogeneous config	**288.69**	**55.44**	**0.90**	**1.43500**	**-37.41%**	∞



(a) (b)

Fig. 9. The search results of the proposed HPIM-NoC for ResNet-18 with (a) $(\alpha=0,\beta=0,\gamma=1)$ and (b) $(\alpha=0,\beta=1,\gamma=1)$.

tively identifies heterogeneous NoC architectures optimized for the target PPA, showcasing distinct tendencies across various performance metrics. The detailed results for the first two settings for ResNet-18 are shown in Fig. 9 (a) and (b), while the detailed results for ResNet-18 with the third setting are presented in Table III. Homogeneous PIM-based NoC architectures and full-space search results were used as baselines for comparison. The results show that full-space search fails to identify solutions with effective improvement within the limited given time. Under the three settings for ResNet-18, the FoMs of heterogeneous PIM-based NoC architectures found by HPIM-NoC are reduced by 1.18%, 16.94%, and 37.41% compared to homogeneous architectures, respectively. The heterogeneous NoC layouts for ResNet-18 with three configurations are shown in Fig. 10, achieving the shortest total wirelength and optimized total area. By minimizing the distance between cores based on the topological interconnections during the placement process, the layouts achieve virtually no crossover points in subsequent routing. The results demonstrate the effectiveness of HPIM-NoC in exploring the near-optimal heterogeneous PIM-based NoC architectures, achieving significant performance improvements compared to homogeneous architectures.



(a) (b) (c)

Fig. 10. The layout examples of (a) $(\alpha=0,\beta=0,\gamma=1)$ with Core_Num = 22×22, (b) $(\alpha=0,\beta=1,\gamma=1)$ with Core_Num = 26×26, and (c) $(\alpha=1,\beta=1,\gamma=1)$ with Core_Num = 24×24 for ResNet-18. Different colors of cores represent mapping to different neural network layers. The lines represent topological interconnections between cores.

V. CONCLUSION

In this work, we propose HPIM-NoC, a comprehensive simulation and architecture search framework specifically designed for heterogeneous PIM-based NoC architectures. HPIM-NoC employs a modified simulated annealing algorithm with a three-stage search process, and a fine-tuning method is introduced to ensure the validity of the discovered architectures. Experimental results demonstrate that the accelerated HPIM-NoC achieves over a $10\times$ reduction in search time during the same exploration process. The FoMs of heterogeneous PIM-based NoCs found by HPIM-NoC are reduced by 1.18%, 16.94%, and 37.41% for ResNet-18 under three settings, respectively, compared to homogeneous architectures. HPIM-NoC is capable of quickly finding near-optimal configurations, providing a powerful tool for heterogeneous PIM-based NoC designs.

ACKNOWLEDGMENT

This work was supported by the National Key R&D Program of China (2023YFB4502200), the National Natural Science Foundation of China (#62174110 and #62325405), the Natural Science Foundation of Shanghai (#23ZR1433200), the Shanghai 2024 "Science and Technology Innovation Action Plan" Fundamental Research Program in Integrated Circuit (No. 24JD1400300), the Beijing National Research Center for Information Science and Technology (No. BNR2024TD03001), and the SJTU-SAINCHIPS Research Grant HC-CN-ZXIC20240321001.

REFERENCES

[1] M. M. S. Aly, T. F. Wu, A. Bartolo, Y. H. Malviya, W. Hwang, G. Hills, I. Markov, M. Wootters, M. M. Shulaker, H.-S. P. Wong *et al.*, "The n3xt approach to energy-efficient abundant-data computing," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 19–48, 2018.

[2] P. Chi, S. Li, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "Prime: A novel processing-in-memory architecture for neural network computation in reram-based main memory," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 27–39, 2016.

[3] J. Zhang, Z. Wang, and N. Verma, "In-memory computation of a machine-learning classifier in a standard 6t sram array," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, 2017.

[4] W. Li, X. Sun, H. Jiang, S. Huang, and S. Yu, "A 40nm rram compute-in-memory macro featuring on-chip write-verify and offset-cancelling adc references," in *ESSCIRC 2021-IEEE 47th European Solid State Circuits Conference (ESSCIRC)*. IEEE, 2021, pp. 79–82.

[5] M. Jagasivamani, C. Walden, D. Singh, L. Kang, S. Li, M. Asnaashari, S. Dubois, B. Jacob, and D. Yeung, "Memory-systems challenges in realizing monolithic computers," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 98–104.

[6] X. Si, J.-J. Chen, Y.-N. Tu, W.-H. Huang, J.-H. Wang, Y.-C. Chiu, W.-C. Wei, S.-Y. Wu, X. Sun, R. Liu *et al.*, "A twin-8t sram computation-in-memory unit-macro for multibit cnn-based ai edge processors," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, 2019.

[7] B. Yan, J.-L. Hsu, P.-C. Yu, C.-C. Lee, Y. Zhang, W. Yue, G. Mei, Y. Yang, Y. Yang, H. Li *et al.*, "A 1.041-mb/mm 2 27.38-tops/w signed-int8 dynamic-logic-based adc-less sram compute-in-memory macro in 28nm with reconfigurable bitwise operation for ai and embedded applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 188–190.

[8] C.-X. Xue, W.-H. Chen, J.-S. Liu, J.-F. Li, W.-Y. Lin, W.-E. Lin, J.-H. Wang, W.-C. Wei, T.-W. Chang, T.-C. Chang *et al.*, "24.1 a 1mb multibit reram computing-in-memory macro with 14.6 ns parallel mac computing time for cnn based ai edge processors," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*. IEEE, 2019, pp. 388–390.

[9] Q. Wang, Y. Yang, Z. Wang, S. Bao, J. Sun, L. Shan, L. Bao, Y. Gao, H. Zhang, Y. Ling *et al.*, "A logic-process compatible rram with 15.43 mb/mm 2 density and 10years@ 150° c retention using sti-less dynamic-gate and self-passivation sidewall," in *2023 International Electron Devices Meeting (IEDM)*. IEEE, 2023, pp. 1–4.

[10] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2306–2319, 2020.

[11] Z. Zhu, H. Sun, T. Xie, Y. Zhu, G. Dai, L. Xia, D. Niu, X. Chen, X. S. Hu, Y. Cao *et al.*, "Mnsim 2.0: A behavior-level modeling tool for processing-in-memory architectures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4112–4125, 2023.

[12] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2013, pp. 86–96.

[13] Y. Zhu, Z. Zhu, G. Dai, F. Tu, H. Sun, K.-T. Cheng, H. Yang, and Y. Wang, "Pim-hls: An automatic hardware generation tool for heterogeneous processing-in-memory-based neural network accelerators," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[14] H. Sun, T. Xie, Z. Zhu, G. Dai, H. Yang, and Y. Wang, "Minimizing communication conflicts in network-on-chip based processing-in-memory architecture," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[15] H. Sun, Z. Zhu, C. Wang, X. Ning, G. Dai, H. Yang, and Y. Wang, "Gibbon: An efficient co-exploration framework of nn model and processing-in-memory architecture," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4075–4089, 2023.

[16] Y. Jing, M. Wu, J. Zhou, Y. Sun, Y. Ma, R. Huang, L. Ye, and T. Jia, "Aigcim: A scalable chiplet module with tri-gear heterogeneous compute-in-memory for diffusion acceleration."

[17] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[18] G. Krishnan, S. K. Mandal, M. Pannala, C. Chakrabarti, J.-S. Seo, U. Y. Ogras, and Y. Cao, "Siam: Chiplet-based scalable in-memory acceleration with mesh for deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

MIA-aware FinFlex Cell Legalization with Power-Driven Cell Version Substitution

Da-Wei Huang and Shao-Yun Fang
Department of Electrical Engineering, National Taiwan University of Science and Technology, Taipei 106, Taiwan
{D11207008, syfang}@mail.ntust.edu.tw

***Abstract*—FinFlex standard cells have been proposed in sub-3 nm process nodes by TSMC to offer great design flexibility and a good trade-off between performance, power, and area efficiency. In the FinFlex technology, each cell type has a number of versions, varying in cell heights and threshold voltages, enabling flexible cell substitution for placement optimization. This paper presents the first work in the literature on standard cell legalization for the FinFlex technology, addressing cell version substitution, power efficiency, and minimum implant area (MIA) constraints. Our proposed algorithm comprises three stages: intra-row violation-aware pre-processing considering timing and power levels, directed acyclic graph-based initial legalization with cell version substitution, and dynamic programming-based inter-row violation removal. Experimental results show the superior performance of our approach over a baseline using state-of-the-art techniques.**

I. Introduction

As reported in [15], TSMC adopts the FinFlex standard cell innovation in 3 nm process node to offer great design flexibility and good trade-off between performance, power, and area efficiency. For example, in the FinFlex 3-2 fin configuration, the cells placed on the rows equipped with 3 fins provide higher driving strengths compared to those placed on 2-fin rows. Different from the traditional 2-2 fin configuration, a 3-fin row has a larger row height compared to that of a 2-fin row, which is considered as non-integer multiple-cell-height or hybrid-row-height (HRH) designs in existing studies [1], [2], [5], [9], [12]. It is expected that HRH designs can also adopt multiple-row-height cells as those in conventional mixed-cell-height designs. For example, [2] and [12] reveal that a double-row-height cell spanning two rows can provide even larger driving strength than a taller but single-row-height cell. They consider four possible version variants of each standard cell considered. The version denoted as *SMALL (S)* of a cell represents its low-driven version, and *WIDE (W)*, *TALL (T)*, and *COMBO (C)* are the high-driven versions. Therefore, a WIDE cell is expected to be wider than a SMALL cell of the same cell type, and a TALL and a COMBO cell are thinner than a WIDE cell. SMALL and WIDE cells are placed in short rows, TALL cells in tall rows, and COMBO cells across a short row and a tall row.

The most critical challenge emerging from this novel design style is that it greatly increases the complexities of cell placement. Dobre et al. propose the first design optimization flow for HRH designs [5]. After that, Chen et al. propose an analytical hybrid-row-height-aware global placement algorithm and a dynamic block region generate approach [1]. Lin and Chang propose a cell legalization approach for HRH designs uniform or non-uniform row structures [9]. The three studies assume that cells have different cell heights (8T or 12T), but they do not consider multiple-row-height cells. In addition, as reported in [9], the legalization results suffer from 16% HPWL overhead compared to the initial global placement results, showing a huge gap in terms of wirelength optimization and the great design difficulty of such novel technology. To mitigate the problem, Liang et al. propose adopting cell version substitution during legalization [12], which allows a cell of the SMALL version to be switched to any of its high-driven versions and a high-driven cell can be switched to one of the other two high-driven versions, to guarantee timing and release the placement resource of short rows. The results in [12] show that applying cell version substitution can contribute to less than 1% wirelength overhead after legalization compared to the given global placement. However, the power overhead caused by transforming low-driven cells to high-driven versions is not considered. This insufficiency motivated this work in the first place.

This work was partially supported by TSMC, Synopsys, MPI, and NSTC of Taiwan under Grant No's NSTC 113-2927-I-011-502, 113-2640-E-002-001, 113-2640-E-006-001, 113-2222-E-011-005-MY3, and 114-2927-I-011-501.

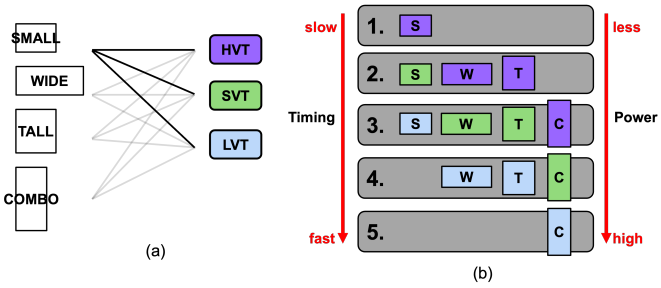


Fig. 1. The FinFlex technology offers multiple versions and threshold voltages (VTs) for each cell type. (a) Four cell sizes and three VTs. (b) Cell classification into five timing and power levels in this work.

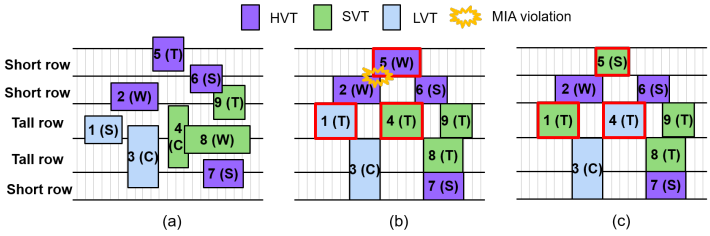


Fig. 2. Motivation of this work. (a) A legalization instance. (b) A legalization result with previous work. (c) An ideal result by considering VT swaps.

It is important to note that in [15], multiple threshold voltages (VTs) are also offered in the FinFlex standard cell innovation in the 3 nm process node, and thus various combinations of the cell height and VT provide a huge optimization range in terms of performance and leakage power to meet different design purposes. For a single cell with a fixed height, adopting a lower VT can have a smaller delay while suffering from larger leakage power. When considering cell version substitution, adopting multiple VTs makes the operation more flexible and contributes to leakage power minimization. As illustrated in Fig. 1(a), if there are four cell sizes and three VT types, high VT (HVT), standard VT (SVT), and low VT (LVT), then the total number of available versions is 12. Each version exhibits unique timing and power characteristics, resulting in a more complex optimization scenario. To simplify the problem, we classify the 12 versions into five groups in this paper, which correspond to the five timing and power levels as shown in Fig. 1(b). The cells in the same group are assumed to have the same timing and power performance. To guarantee the timing and power closure of a design, a cell can only be substituted with a version belonging to the same group. For example, in power level 2, a SMALL cell with SVT can only be substituted with its WIDE or TALL version with HVT, according to our cell classification. This work can also effectively handle practical situations where the timing and power performances of all versions differ.

However, the designs using multiple threshold voltages must comply with an additional set of design rules due to the minimum implant area (MIA) constraint, which will be introduced in Section II-A. There have been many existing studies focusing on MIA-aware legalization and detailed placement problems [3], [4], [7], [8], [11], [13], [14], [16],

but they are not developed for HRH designs. Although dealing with the HRH legalization problem or handling the MIA-related rules alone is already difficult, we excitedly observed that cell version substitution in HRH designs may facilitate MIA compliance, which serves as the second motivation of this work. Figure 2(a) shows a legalization instance with the row configuration of HRH designs suggested by [2]. The cell colors indicate three VTs and cell versions are denoted by S, W, T, and C. Figure 2(b) shows a result the state-of-the-art work [12] may generate, where Cells 1, 4, 5, 8 are performed version substation to minimize cell displacements, without considering VTs. Among them, Cell 5 experiences an MIA violation, Cell 1 causes power overhead, and Cell 4 suffers from timing degradation, according to our timing and power assumption in Fig. 1(b). As illustrated in Fig. 1(c), using different VTs for Cells 1 and 4 such that the new versions are in the same groups as the old ones can guarantee the timing and power performance. In addition, using a different version as well as a different VT for Cell 5 can even resolve the MIA violation.

This paper, therefore, proposes an algorithm flow for MIA-aware and power-driven HRH legalization with cell version substitution. The major contributions of this work are as follows:

- This is the first work in the literature to address MIA compliance in HRH designs through performance-driven cell version substitution.
- A timing and power level-aware cell clustering algorithm is proposed that leverages the flexibility of version substitution to eliminate potential intra-row MIA violations and contribute to better legalization results.
- A directed acyclic graph (DAG)-based legalization approach is developed to determine the best version of each cell contributing the minimized displacements.
- A dynamic programming-based method is adopted to systematically remove inter-row MIA violations.
- Experimental results demonstrate significant performance improvements in terms of cell displacement, MIA violations, and runtimes compared to the state-of-the-art approach [12].

In the following, Section II introduces some background knowledge of this work. Section III details the proposed algorithms. Finally, Sections IV and V respectively show the experimental results and briefly conclude this paper.

II. Preliminaries

This section first explains design rules induced by the MIA constraint and then briefly introduces an optimal mixed-cell-height detailed placement algorithm, which inspires us to develop our dynamic programming-based MIA violation removal method.

A. Minimum Implant Area (MIA) Constraint

The MIA constraint sets a minimum implant area requirement, meaning a narrower cell can violate this rule if its neighboring cells have different threshold voltages, which results in an *intra-row MIA violation*, as the one shown in Fig. 3(a). It is common to transform the MIA constraint into a minimum width rule, min_w. In addition, two vertically adjacent cells with an overlap smaller than min_w are also prohibited and defined as an *inter-row MIA violation*, as the two vertically adjacent cells in Fig. 3(b). There are two common techniques to fix intra-row MIA violations. The first one is to directly insert filler cells beside a narrow cell to increase the implant width, as shown in Fig. 3(c). The second method is to cluster neighboring narrow cells with the same VT. Similarly, filler cell insertion (as illustrated in Fig. 3(d)) and cell shifting are usually used to resolve inter-row MIA violations. Considering that existing methods are not developed for HRH designs and typically address only a limited subset of the aforementioned operations, there is a strong need for effective algorithms that incorporate cell clustering, cell shifting, filler cell insertion, and cell version substitution.

B. Optimal Detailed Placement for Mixed-Cell-Height Designs

Huang et al. [6] propose a dynamic programming algorithm that can optimally solve the detailed placement problem for mixed-cell-height designs based on a given set of placement operations such as cell flipping and shifting. The algorithm breaks the traditional left-to-right cost propagation procedure in all previous studies and regards a growing group of multi-row-height cells as a sub-problem solved in each iteration. As shown in Fig. 4(a), [6] constructs a graph for the multi-row-height

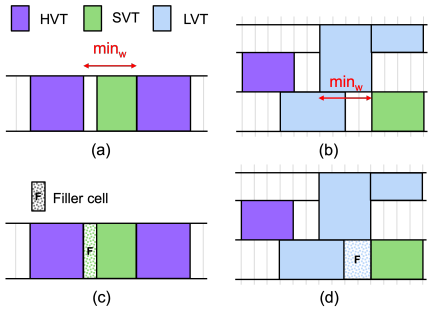


Fig. 3. MIA-induced design rule violations. (a) An intra-row violation. (b) An inter-row violation. (c)(d) Filler cell insertion for violation removal.

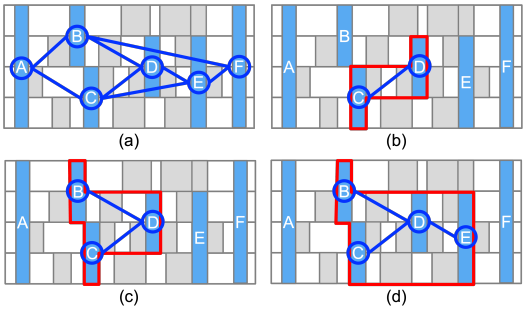


Fig. 4. The dynamic programming algorithm to derive optimal detailed placement for mixed-cell-height designs [6].

cells in a given placement, where each edge indicates that the two corresponding cells are adjacent by neglecting the single-row-height cells between them. In Fig. 4(b), the first subproblem may be composed of Cells C and D, deriving the optimal placement solution for each state combination of C and D. A state of a cell indicates a specific placement location and orientation for the cell. The second subproblem may be formed by clustering Cell B into the group, as shown in Fig. 4(c), enumerating the optimal solutions for each state combination of B, C, and D. Fig. 4(d) shows the third subproblem by incorporating E. Because after forming this subproblem, D is independent of the other multi-row-height cells outside the group, it can be omitted in the following process and its cost can be well considered in each state combination of B, C, and E. In this way, the optimal solution can be obtained when all multi-row-height cells are included in the group. Due to the optimality and the ability to consider the discrete states of each cell, we leverage this approach and make appropriate extensions in our work, which will be detailed below.

III. Proposed Algorithms

This section presents the overall algorithm flow, followed by a detailed explanation of each step.

A. Overall Framework

Our algorithm for hybrid-row-height placement, incorporating cell version substitution and complex MIA constraints, is summarized in Fig. 5. It consists of three major stages: (1) the pre-processing stage, which addresses the intra-row MIA violation issue and determines cell ordering, (2) the legalization stage, which primarily determines the suitable location for each cell, selects its version, and resolves cell overlaps, and (3) the post-processing stage, which minimizes remaining inter-row violations while ensuring that previously resolved violations are not re-generated. To handle the intra-row MIA violations, we propose a filler-aware clustering algorithm. This method determines whether to cluster cells or insert filler cells for narrower cells based on the density of their surrounding areas, allowing for more effective utilization of empty space. To legalize cells, we propose a DAG-based approach that determines the version and position of cells simultaneously. This method offers cells maximum flexibility in placement while minimizing

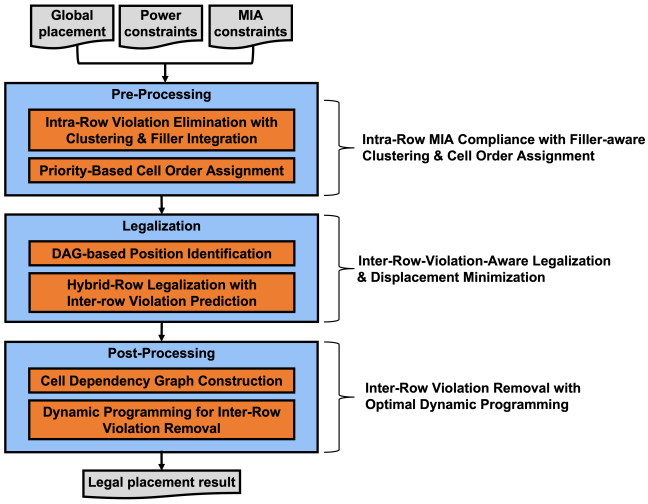


Fig. 5. The proposed algorithm framework.

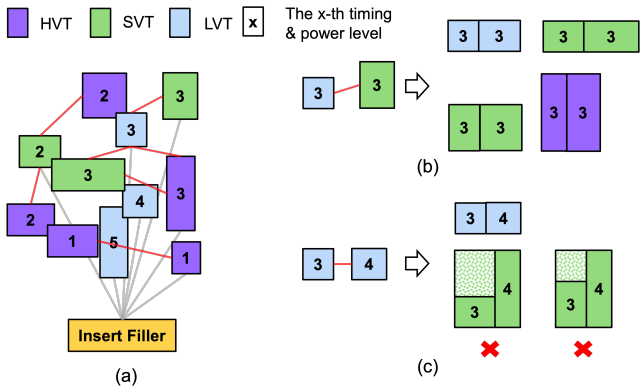


Fig. 6. Cell clustering to eliminate potential intra-row MIA violations. (a) A clustering graph. (b) Clustering two cells with the same timing and power level. (c) Clustering two cells with different levels.

displacement and avoiding inter-row MIA violations. Finally, we introduce new perspectives to mitigate inter-row violation. By simultaneously employing filler insertion and small amounts of cell shifting, great numbers of inter-row MIA violations can be removed. This method ensures that the subsequent detailed placement stage can address the remaining violations more easily and effectively.

B. Pre-Processing

In this section, we utilize cell cluster and filler cell insertion to resolve intra-row MIA violations and determine cell ordering for the following legalization step.

1) Intra-Row Violation Elimination with Clustering & Filler Integration: As introduced in Section II-A, an intra-row MIA violation is caused by a cell narrower than min_w and placed between two cells of different VTs. To resolve each potential intra-row violation, one technique involves adding a filler cell beside a narrow cell, and the other focuses on clustering cells with the same VT. The disadvantage of inserting filler cells lies in requiring additional space. On the other hand, clustering cells can lead to more cell displacements. Determining which cells should be assigned filler cells or clustered requires an effective strategy to minimize potential issues in the subsequent legalization stage.

To address this challenge, we propose a filler-aware clustering algorithm designed to effectively balance space utilization and displacement. For cells located in high-density regions, where available space is constrained, clustering can optimize space usage, freeing up more room for other cells and post-legalization optimization. Conversely, in lower-density regions, inserting filler cells can mitigate displacement while avoiding excessive resource consumption. To calculate the density of cells in a local region, we establish a window centered at each cell, extending outward by a distance equal to the combined height of a short row and a tall row. The total cell area within this window is then divided by the effective area of the window to obtain the density. To accelerate the density calculation, we utilize a QuadTree structure to record cell positions, achieving a time complexity of $\mathcal{O}(\log n)$.

A clustering graph $G(V, E, W)$ is constructed to facilitate the cell clustering task, as shown in Fig. 6(a). Each cell narrower than min_w is modeled as a vertex v_i, and it is connected to other vertices representing the other cells in the window centering at v_i with the same power and timing level. Each edge indicates that the two vertices form a clustering candidate. Note that although clustering two cells with the same VT is more intuitive to directly resolve a potential intra-row violation for a narrow cell (which is also a common way adopted in previous studies), we found that clustering two cells of the same power and timing level contributes to larger flexibility during the legalization stage. Fig. 6(b) demonstrates an example of clustering two cells belonging to the third power and timing level. With cell version substitution, four possibilities are present to cluster them, and the option most favorable for legalization can be selected. In contrast, Fig. 6(c) shows an example of clustering two cells with the same VT between in different timing and power levels.

Algorithm 1: Filler-aware clustering

Input: A global placement result
Output: Intra-row MIA free result

Construct the clustering graph $G(V, E, W)$;
repeat
  Find an edge $e_{uv} \in E$ with the minimum weight w_{uv};
  if e_{uv} *is connected to the filler vertex* **then**
    Set x as the non-filler vertex between u and v;
    $x.Is_Filler \leftarrow TRUE$ and $x.Is_Cluster \leftarrow FALSE$;
    Delete e_{uv};
  end
  else
    $u.Is_Filler \leftarrow FALSE$ and $u.Is_Cluster \leftarrow TRUE$;
    $v.Is_Filler \leftarrow FALSE$ and $v.Is_Cluster \leftarrow TRUE$;
    if *both* u *and* v *are cell vertices* **then**
      Group u and v as a cluster u_ℓ;
      Set the position of u_ℓ at the median point of u and v;
      Assign the density of u_ℓ as the maximum density of u and v;
      Assign the width of u_ℓ as the total width of u and v;
      Delete the edge e_{uv};
      Delete the edge $e_{ui} \in E$, if Cell i is wider than min_w;
      Delete the edge $e_{vj} \in E$, if Cell j is wider than min_w;
      Update the edge weights connecting to u_ℓ by Equation (2);
    end
    else
      Set x as the cell vertex and set u_ℓ as the cluster;
      Merge cell x into u_ℓ;
      Update the position, width, and density of u_ℓ;
      Delete the edge e_{xu_ℓ};
      Delete the edge $e_{xi} \in E$ if Cell i is wider than min_w;
      Update the edges connecting to u_ℓ by Equation (2);
    end
  end
until $E == \emptyset$;
return;

Since a cell can only be substituted with a version in the same level, only one feasible combination exists. Although we may also switch the VTs of the two cells such that two more combinations can be considered, filler cell insertion is then required, ultimately offering no advantage to the cell clustering operation.

In addition to cell vertices, a vertex representing filler cell insertion is also constructed in the clustering graph and connected to the vertices of narrow cells, indicating that each narrow cell can also be clustered with a filler cell. Each edge is associated with an edge weight $w_{uv} \in W$ calculated as follows:

$$w_{uv} = \theta_{uv} \times (\alpha_1 \times dist_{uv} + \alpha_2 \times width_{uv} - \alpha_3 \times density_{uv}), \quad (1)$$

where $dist_{uv}$ is the Manhattan distance between Cells u and v, $width_{uv}$ is the total cell width of u and v, and $density_{uv}$ is the larger local density of the two windows of the two cells. α_1, α_2, and α_3 are user-

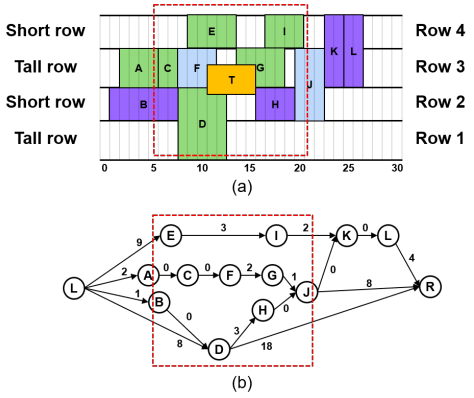


Fig. 7. DAG-based cell legalization. (a) A target cell T being legalized and other already legalized cells. (b) The DAG and the region to find candidate legalization locations for T.

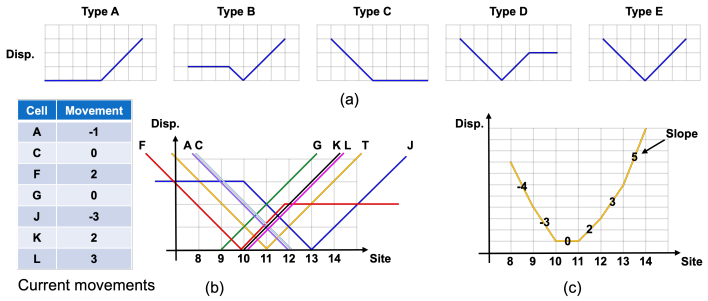


Fig. 8. Total displacement calculation for a candidate interval. (a) The five displacement curves [10]. (b) The displacement curves of the affected cells if T is inserted in the interval (FG). (c) The final displacement curve by merging those in (b).

defined parameters used for normalization such that the three terms fall in the range of $[0, 1]$. Note that if a vertex of the edge e_{uv} is the filler vertex, $dist_{uv}$ is set as one-quarter of the window width, discouraging cell clustering when the distance of the two cells exceeds this value. θ_{uv} is set as 0.5 if both the two cells are narrower than min_w; otherwise, θ is set as 1.

With the clustering graph G, the proposed filler-aware clustering method is outlined in Algorithm 1. The algorithm iteratively clusters two cells or a cell and a cluster with the minimum edge weight in each iteration. If one of the vertices of an edge is the filler vertex, the two flags Is_Filler and $Is_Cluster$ are marked accordingly (Lines 4–8). If both the two vertices represent cells, we merge the two cells as a cluster u_ℓ, compute the position, density, and width for u_ℓ, and delete edges that are not necessary for the following (Lines 12–21). A similar process is applied for the case when one of the two vertices is already a cluster (Lines 22–29).

2) Priority-Based Cell Order Assignment: An effective cell ordering strategy can lead to faster convergence during sequential legalization and help minimize displacement as well. In our approach, cells that are wider, located in dense areas, or have fewer available versions are given higher priorities. In contrast, cells that do not meet these criteria have more placement options, making them easier to legalize. This prioritization ensures that critical cells are addressed first, facilitating a smoother and more efficient legalization process. Therefore, the priority of each cell i is defined as follows:

$$P_i = \beta_1 \times width_i + \beta_2 \times density_i + \beta_3 \times \frac{1}{candidate_i}, \tag{2}$$

where $width_i$, $density_i$, and $candidate_i$ are the width, density, and the number of candidate versions with the same timing and power level of Cell i. A cell with a higher P_i has a priority to be legalized.

C. Legalization

After the pre-processing stage, all potential intra-row violations are eliminated. In this stage, the final version of each cell is determined and is placed on a legal row to minimize displacement and the possibility of incurring inter-row violations. This step is crucial for ensuring outstanding placement performance with minimal inter-row violations. To legalize cells efficiently, We adopt a directed acyclic graph (DAG) as a horizontal constraint graph G to record the current legalization result, as shown in Fig. 7. A directed edge is connected from cell u to v in G if v is immediately on the right to u in some rows. The left and right boundaries and macros are treated as fixed cells. Each edge weight indicates the distance between the two legalized cells.

1) DAG-based Position Identification: The legalization procedure is performed sequentially. For a target cell T being legalized, we use the window of T to identify its candidate intervals in the DAG. As illustrated in Fig. 7(b), all edges within the window are considered potential placement candidates for T. For a non-COMBO version, the process involves straightforwardly selecting the appropriate candidate row for each candidate edge. In contrast, for a COMBO version, the approach combines candidate intervals across adjacent rows, including both short and tall rows. To enhance the quality of the placement result, once the $cluster$ is placed, it is de-clustered back to its original configuration. This de-clustering allows for more space to be utilized when searching for candidate intervals in subsequent steps, improving overall placement flexibility and efficiency. To prevent intra-row violations from reoccurring, when selecting the intervals between de-clustered cells, it is essential to ensure that the VT of the target cell matches that of the de-clustered cells.

As illustrated in Fig. 7(b), for the target cell T of a non-COMBO version, the candidate intervals for Cell T are (LD, DR), (BD, DH, HJ), (AC, CF, FG, GJ), (LE, EI, IK), separately for the four rows from bottom to top. If T is a COMBO version, the candidate intervals between Rows 1 and 2 are the combinations of $\{(LD, DR), (BD, DH, HJ)\}$. Notice that when enumerating the combinations of two adjacent rows, we need to ensure their legality. For example, the candidate interval (LD, DH) and (LD, HJ) on Row 1 and 2 are illegal combinations, because T cannot simultaneously be placed on the right and left sides of Cell D. Moreover, since candidate (AC) is a $cluster$, the target cell must have the same VT when selecting this interval to prevent the intra-row violation on Cell C.

When evaluating the displacement of inserting T at each candidate interval, we need to consider the $movement$s of other legalized cells. The $movement$ of a cell represents the changes in the x-coordinate of a cell from its global position to its current position. When the $movement$ is positive/negative, it indicates that the cell is positioned on the right/left of its global position, respectively. To evaluate the change in overall displacement after placing the target cell in a candidate interval, we use the five displacement curves proposed in [10], as illustrated in Fig. 8. For the placed cells on the right side of the target cell, the resulting curve will be classified as either type A or B. On the other hand, if the placed cells are on the left side of the target cell, the resulting curve will be classified as either type C or D. Type E represents the displacement curve specifically for the target cell.

When the target cell T is placed into a candidate interval, it may cause overlap with neighboring cells on both sides, making it essential to assess the extent of the impact. First, we calculate the total overlap created by inserting T and identify the non-overlapping sites with the adjacent cells. As illustrated in Fig. 7(a), for the interval (FG), T needs to be placed at $site$ 12 to be non-overlapped with F, and it needs to be placed at $site$ 9 to be non-overlapped with G, with a total overlap of 3. Referring to Fig. 8(b), and based on the left table, the current $movement$ of Cell F is 2, and thus its displacement curve is classified as type D (shown in red). This indicates that when Cell T is placed right to $site$ 12, Cell F does not need any shifting, resulting in a horizontal curve segment. When Cell T is placed between $site$ 10 to 12, the displacement decreases, because Cell F moves toward its global position. When Cell T is placed left to $site$ 10, the displacement increases. On the other side, based on the left table, since the current $movement$ of Cell G is 0, its displacement curve is type A (shown in green). This means that when Cell T is placed right to $site$ 9, the displacement of Cell G increases linearly. The turning points in the displacement curve for Cell F are at sites 10 and 12, while the turning point for Cell G is at site 9.

Afterward, we need to address the overlapping issue by shifting the

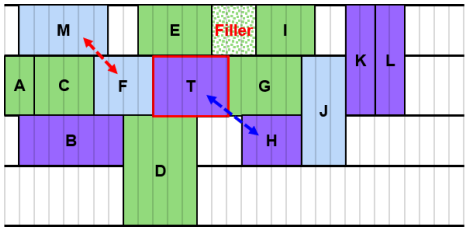


Fig. 9. Inter-row MIA violation prediction during legalization.

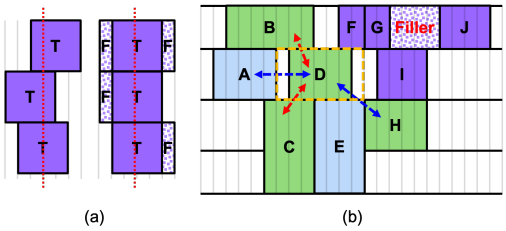


Fig. 10. Dynamic programming-based inter-row violation removal. (a) Six available states of each cell for a specific cell shifting and filler insertion scenario. (b) A placement instances with two inter-row violations and two potential violations.

cells outward. We use the DAG structure and Breadth-First Search (BFS) to find out whether sufficient space exists to eliminate the overlap. This approach can accurately calculate the total displacement of affected cells, which is the major difference between our approach and that in [10]. BFS traverses all affected cells on both sides. As it encounters available space, the overlap is progressively reduced until it becomes less than or equal to 0. Continuing the previous example in Fig. 7(a), since the overlap is 3, we first try to resolve the overlap by traversing to the right side. Upon reaching the next cell, J, since there is 1-site spacing between G and J, the overlap is reduced to 2. However, since overlap still exists, the traversal must proceed further to the right to fully resolve it. In the lower row of J, the overlap can be resolved due to the 8 available sites between J and the right boundary. In the upper row of J, the traversal encounters K and L sequentially. With no spacing between them, both K and L are affected by Cell T, and the traversal continues to the right until the overlap is completely eliminated. For each affected cell, a displacement curve is constructed based on its current $movement$ and using the similar procedure mentioned above, as shown in Fig. 8(b).

After traversing both sides, the displacement curves of all affected cells are merged into a single curve, as shown in Fig. 8(c). The resulting total displacement curve forms a convex function, indicating that shifting the target cell will either maintain or increase the overall displacement. Constructing the total displacement curve involves summing the slopes at each turning point from left to right. Following this, we identify the segments where the slope changed from negative to non-negative or from non-negative to positive, which represent the best intervals for placing the target cell. In the example shown in Fig. 8(c), the target cell can be placed at site 10 or 11, resulting in minimized displacement with this ordering.

Note that we assign a large cost if BFS traverses to a macro, preventing the macro from being moved. In addition, in some dense regions, placing the target cell in a candidate interval located in a dense region may move cells outward very far, which is time-consuming and undesirable. Thus, we set an upper bound for BFS traversal to avoid such non-ideal situations.

2) Hybrid-Row Legalization with Inter-Row Violation Prediction: In addition to using the total displacement of affected cells as the metric to evaluate the quality of placing the target cell in an interval, considering whether it has a high possibility of resulting in inter-row MIA violations is also important. In previous works, inter-row violations were typically checked for all cells in adjacent rows, which is time-consuming and can lead to pessimistic results. In fact, if multiple cells with the same VT are placed in the same row consecutively, the white spaces among them can be filled with filler cells of the same VT. Therefore, it is only necessary to check the leftmost and the rightmost cells for inter-row violations. As shown in Fig. 9, an inter-row violation can be asserted by only checking Cells G and I. However, since Cells E and I have the same VT, filler cells with the same VT can be inserted between them, thereby eliminating any inter-row violation between G and I. This ensures that the estimation will not result in errors when checking Cell G and I due to the presence of the filler cell.

When temporarily placing the target cell in a candidate interval and shifting all affected cells without overlaps, we will check whether the placement may result in a significant number of inter-row violations. Additionally, we also check potential violations within a user-defined distance to ensure comprehensive validation. If the placement result generates numerous violations and potential violations, a penalty is assigned to the candidate based on the severity of these violations. As shown in Fig. 9, when Cell T is temporarily placed in the interval (FG) and given a user-defined distance of two sites, the result shows one violation and one potential violation, respectively indicated with red and blue double arrows. Note that we only perform this violation prediction procedure if the target cell is located in a dense region, because inter-row MIA violations can be easily removed in sparse regions.

D. Post-Processing

When all cells are legalized, inter-row violations may still occur in the placement. To mitigate violations, we extend the Dynamic Programming (DP) algorithm introduced in Section II-B to systematically remove these violations. In addition to utilizing cell shifting as in all existing studies, we also consider filler cell insertion, which is an effective inter-row violation removal method without cell displacements for sparse regions.

1) Cell Dependency Graph Construction: The dynamic programming algorithm introduced in Section II-B can theoretically generate an optimal placement refinement result under a pre-defined set of available cell states. To resolve inter-row violations, we apply cell shifting with a pre-defined maximum cell shifting and a pre-defined set of filler cell insertion configurations. As illustrated in Fig. 10(a), if the maximum cell shifting is limited to one placement site and we only consider inserting a single unit filler cell to the left or right adjacent to a target cell, there are overall six states for each cell.

Similar to [6], we construct a graph for a set of vertices representing cells that are dependent on each other considering cell overlaps and inter-row violations. Different from that in [6], cell dependency is not restricted to horizontally adjacent cells; it will also occur between vertically adjacent cells. As shown in Fig. 10(b), Cell D is dependent on Cells B and C due to the current inter-row violations. In addition, according to the pre-defined maximum cell shifting of one placement site, the yellow box indicates the possible location where D is placed after shifting. Therefore, D is also dependent on Cells A and H, because D and A may overlap, and D and H may suffer from an inter-row violation, after shifting. As a result, in the graph constructed for the instance in Fig. 10(b), D connects to the other four cells highlighted by the double arrows. The vertex cost for each cell is determined by the cell displacement from its global position represented by the corresponding state. Each edge is associated with a cost table, recording the penalties due to cell overlap and inter-row violations caused by each state combination of the two connected cells. It is important to note that the costs induced by cell overlaps must be set higher than the cost of inter-row violations to prioritize preventing overlaps.

2) Dynamic Programming for Inter-Row Violation Removal: Constructing the dependency graph of a given placement, the dynamic programming-based algorithm introduced in Section II-B is applied to find the optimal refined placement without cell overlaps and with the minimized number of inter-row violations. Since we simultaneously consider cell shifting and filler cell insertion by enumerating cell states, the total cell displacement can also be optimized by inserting filler cells in sparse regions and automatically determining which side a filler cell will be inserted by a target cell.

Note that the cells in a dense region could be very compact, increasing the likelihood of overlaps and inter-row violations when cells are shifted. The dependency graph constructed for a dense region may be very complex with many edges, and solving the graph during the dynamic programming process can lead to a much longer runtime or even memory explosion. Therefore, we set an upper bound on the number of vertices considered at a time for dense regions. In this way, the placement

TABLE I
EXPERIMENTAL RESULTS ON LEGALIZATION PERFORMANCE.

Benchmark		Enhanced [12]			Ours		
Case	Init_Density (%)	Total Disp.	Density (%)	CPU (s)	Total Disp.	Density (%)	CPU (s)
des_perf_a_md1	59.27	248,819	59.22	64.536	102,829	56.15	46.723
des_perf_a_md2	58.75	185,649	58.18	58.014	95,200	54.65	34.122
des_perf_b_md1	70.16	128,399	67.69	27.708	74,003	61.85	2.391
des_perf_b_md2	72.19	128,985	69.89	39.166	73,159	64.20	3.260
edit_dist_1_md1	69.04	168,358	64.75	9.055	82,714	60.21	2.438
edit_dist_a_md2	66.90	170,187	63.54	32.946	95,166	59.26	3.341
edit_dist_a_md3	63.38	227,345	60.96	47.833	115,694	57.26	19.436
fft_2_md2	76.03	40,705	74.27	10.774	23,164	71.18	0.726
fft_a_md2	37.05	32,642	36.03	0.349	15,341	32.63	0.293
fft_a_md3	37.10	33,076	35.26	0.338	15,385	32.21	0.298
pci_bridge32_a_md1	54.40	35,279	54.31	5.859	25,806	48.80	0.627
pci_bridge32_a_md2	55.57	37,475	55.28	12.156	21,522	49.88	4.120
pci_bridge32_b_md1	26.01	35,967	25.62	1.282	16,908	22.86	0.328
pci_bridge32_b_md2	21.14	36,155	21.02	4.648	18,479	18.88	0.397
pci_bridge32_b_md3	22.99	37,243	22.37	4.613	19,031	19.99	0.434
Normalized	-	1.000	1.000	1.000	0.473	0.895	0.622

TABLE II
EXPERIMENTAL RESULTS ON INTER-ROW VIOLATIONS.

Benchmark	Enhanced [12]		Ours			
			w/o Prediction		w/ Prediction	
Case	I_IRV.	F_IRV.	I_IRV.	F_IRV.	I_IRV.	F_IRV.
des_perf_a_md1	7,540	4,467	9,310	3,849	8,311	2,859
des_perf_a_md2	7,364	4,033	9,276	2,604	8,294	1789
des_perf_b_md1	7,473	1,606	9,440	356	8,268	186
des_perf_b_md2	7,902	2,377	9,818	507	9,024	311
edit_dist_1_md1	5,778	497	7,888	131	7,454	83
edit_dist_a_md2	6,606	1,959	8,588	463	7,445	198
edit_dist_a_md3	6,843	2,796	8,780	1,801	7,478	1,137
fft_2_md2	2,114	577	2,373	104	2,128	58
fft_a_md2	1,023	16	1,348	9	1,287	3
fft_a_md3	1,037	9	1,470	9	1,419	2
pci_bridge32_a_md1	1,985	454	2,306	120	2,139	69
pci_bridge32_a_md2	2,324	872	2,464	372	2,278	223
pci_bridge32_b_md1	1,385	106	1,648	21	1,584	11
pci_bridge32_b_md2	1,515	247	1,889	60	1,826	37
pci_bridge32_b_md3	1,592	299	1,933	48	1,805	43
Normalized	1.000	0.325	-	-	-	-
	-	-	1.000	0.093	0.918	0.061

refinement procedure can be conducted in a much efficient way with such a divide-and-conquer strategy.

IV. EXPERIMENTAL RESULTS

We implemented our MIA-aware FinFlex Cell legalization with C++ programming language and conducted the experiments on a 3.9 GHz Linux machine with 62 GB of memory. We adopted and modified the benchmark circuits from the ICCAD-2017 CAD Contest in Multi-Deck Standard-Cell Legalization [17]. We ignored the fence region constraint and kept all macros in the same positions. To preserve the original placement area, we assume that the heights of a short row and a tall row are respectively 0.8 and 1.2 times the height of the original row. We adopt the following procedure to generate cells conforming to the cell versions considered in this work. For each cell in the given global placement result of a benchmark circuit, we first ignore its original cell height, and its original width w is regarded as the cell width of its SMALL version. Then the cell is randomly assigned to a timing and power level, along with a randomly selected version and threshold voltage within that level. If the cell is selected as a TALL/WIDE/COMBO version, its width is set as $1w/2w/0.7w$. The maximum cell shifting is set as one placement site and the three filler cell insertion configurations mentioned in Section III-D1 are adopted, and thus the six states in Fig. 10(a) are used in our implementation for dynamic programming-based inter-row violation removal.

To demonstrate the effectiveness of the proposed algorithm flow, we compare our results with [12], which is the state-of-the-art legalization work for HRH design. [12] separates cell version substitution and legalization into two distinct stages, and the version substitution stage only ensures that the densities of short and tall cells of each local region are below a maximum threshold. Consequently, during the legalization stage, cell versions cannot be further adjusted based on the current placement conditions. Since [12] only determines cell versions and legalizes cells without considering the MIA constraint, for a fair comparison, we integrate the cell clustering technique proposed in [3] and our dynamic programming-based post-processing approach to enhance [12] with MIA awareness.

Table I shows the legalization performances of [12] and our approach. The column "Init_Density" gives the initial density of each benchmark, and the columns "Total Disp.", "Density", and "CPU" separately represent the total cell displacement in terms of sites, the final placement density, and the runtime in seconds. The results show that our algorithm averagely achieves a 53% improvement in the total cell displacement, a 10% reduction in placement density, and a 34% reduction in runtime. The enhanced performance in placement is due to the simultaneous consideration of cell version substitution and filler cell insertion when performing cell clustering. In addition, the flexibility of our approach allowing cell version substation in the legalization stage further contributes to minimizing displacement.

Table II presents the results on inter-row violations at different stages, comparing the outcomes with and without using the proposed predicted model. The column "I_IRV" represents the number of inter-row violations after the legalization stage, and "F_IRV" indicates that after dynamic programming-based post-processing optimization. "w/ Prediction" and "w/o Prediction" respectively indicate the results with and without the use of the inter-row violation prediction approach during legalization. Several features can be found in the results. First, our results have larger numbers of initial inter-row MIA violations, which is because the proposed legalization approach generates more compact placements to minimize the total displacement. Second, the proposed dynamic programming-based inter-row violation removal approach is very effective and efficient. With slight shifts and three simple filler cell insertion configurations applied through the dynamic programming-based post-processing approach, the number of violations decreases significantly for every approach, resulting in at most 94% violation removal. Third, utilizing the predicted model not only reduces the inter-row violations after the legalization stage but also results in lower numbers of remaining violations. Since the approach of [12] can result in many high-density local regions and cells are placed without considering MIA violations, the violations in the dense regions are difficult to be removed in the post-processing stage with minor cell adjustments.

V. CONCLUSIONS

This paper introduces a novel approach for standard cell legalization in the TSMC's FinFlex technology for sub-3 nm nodes, focusing on cell version substitution, power and timing efficiency, and MIA-induced design rules. The proposed three-stage method composed of pre-processing, DAG-based legalization, and dynamic programming for inter-row violation removal optimizes cell placement and minimizes violations. Experimental results demonstrate that this approach outperforms existing state-of-the-art techniques, enhancing design flexibility, and offering an efficient solution for FinFlex-based IC design.

REFERENCES

[1] J. Chen, Z. Huang, Y. Huang, W. Zhu, J. Yu, and Y.-W. Chang, "An Efficient EPIST Algorithm for Global Placement with Non-Integer Multiple-Height Cells," in *proceedings of Design Automation Conference (DAC)*, 2020.

[2] K.-Y. Chen, H.-C. Hsu, W.-K. Mak, and T.-C. Wang, "HybridGP: Global Placement for Hybrid-Row-Height Designs," in *proceedings of Asia South Pacific Design Automation Conference (ASP-DAC)*, 2022.

[3] J. Chen, P. Yang, X. Li, W. Zhu, and Y.-W. Chang, "Mixed-Cell-Height Placement with Complex Minimum-Implant-Area Constraints," in *proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2018.

[4] J. Chen, Y.-W. Chang, and Y.-Y. Wu, "Mixed-Cell-Height Detailed Placement Considering Complex Minimum-Implant-Area Constraints," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 40, no. 10, pp. 2128–2141, 2021.

[5] S. A. Dobre, A. B. Kahng, and J. Li, "Design Implementation With Noninteger Multiple-Height Cells for Improved Design Quality in Advanced Nodes," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 4, 2018.

[6] D.-W. Huang, Y.-J. Jiang, and S.-Y. Fang, "Spacing Cost-aware Optimal and Efficient Mixed-Cell-Height Detailed Placement for DFM Considerations," *proceedings of International Conference on Computed-Aided Design (ICCAD)*, 2023.

[7] E. Jeong, H. Park, and T. Kim, "A Systematic Removal of Minimum Implant Area Violations under Timing Constraint," in *proceedings of Design, Automation & Test in Europe (DATE)*, 2022.

[8] A. B. Kahng and H. Lee, "Minimum Implant Area-Aware Gate Sizing and Placement," in *proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, 2014.

[9] Z.-Y. Lin and Y.-W. Chang, "A Row-Based Algorithm for Non-Integer Multiple-Cell-Height Placement," in *proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2021.

[10] H. Li, W.-K. Chow, G. Chen, B. Yu, E. F.Y. Young, "Pin-Accessible Legalization for Mixed-Cell-Height Circuits," *Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 41, no. 1, pp. 143–154, 2022.

[11] H.-C. Lin and S.-Y. Fang, "MIA-aware Detailed Placement and VT Reassignment for Leakage Power Optimization," in *proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023.

[12] S.-H. Liang, T.-L. Hsiung, W.-K. Mak, and T.-C. Wang, "Hybrid-Row-Height Design Placement Legalization Considering Cell Variants," in *proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2023.

[13] S.-I Lei, W.-K. Mak, and C. Chu, "Minimum Implant Area-Aware Placement and Threshold Voltage Refinement," in *proceedings of Asia South Pacific Design Automation Conference (ASP-DAC)*, 2016.

[14] K.-H. Tseng, Y.-W. Chang, and C. C.-C. Liu, "Minimum-Implant-Area-Aware Detailed Placement with Spacing Constraints," in *proceedings of ACM/IEEE Design Automation Conference (DAC)*, 2016.

[15] S.-Y. Wu, C.-H. Chang, M.-C. Chiang, C.-Y. Lin, J.-J. Liaw, J.-Y. Cheng, J.-Y. Yeh, H.-F. Chen, S.-Y. Chang, K.-T. Lai, M.-S. Liang, K.-H. Pan, J.-H. Chen, V.-S. Chang, T.-C. Luo, X. Wang, Y.-S. Mor, C.-I. Lin, S.-H. Wang, M.-Y. Hsieh, C.-Y. Chen, B.-F. Wu, C.-J. Lin, C.-S. Liang, C.-P. Tsao, C.-T. Li, C.-H. Chen, C.-H. Hsieh, H.-H. Liu, P.-N. Chen, C.-C. Chen, R. Chen, Y.-C. Yeo, C.-O. Chui, W. Chang, T.-L. Lee, K.-B. Huang, H.-J. Lin, K.-W. Chen, M.-H. Tsai, K.-S. Chen, X.-M. Chen, Y.-K. Cheng, C.-H. Wang, W. Shue, Y.-Ku, S.-M. Jang, M. Cao, L.-C. Lu, and T.-S. Chang, "A 3nm CMOS FinFlex^{TM} Platform Technology with Enhanced Power Efficiency and Performance for Mobile SoC and High Performance Computing Applications," in *proceedings of International Electron Devices Meeting (IEDM)*, 2022.

[16] Z. Zhu, Z. Huang, P. Yang, W. Zhu, J. Chen, H. Zhou, and S. Dong, "Mixed-Cell-Height Legalization Considering Complex Minimum Width Constraints and Half-Row Fragmentation Effect," *Integration*, vol. 71, no. C, pp. 1–10, 2020.

[17] ICCAD-2017 CAD Contest in Multi-Deck Standard-Cell Legalization, http://iccad-contest.org/2017/index.html

Tropical: Enhancing SLO Attainment in Disaggregated LLM Serving via SLO-Aware Multiplexing

Jinming Ma[1*], Jiefei Chen[12*], Xiuhong Li[3#], Jiangfei Duan[14], Haojie Duanmu[15]
Xingcheng Zhang[16], Chao Yang[3], Dahua Lin[14]
[1]Shanghai Artificial Intelligence Laboratory, [2]Fudan University, [3]Peking University
[4]The Chinese University of Hong Kong, [5]Shanghai Jiao Tong University, [6]Sensetime Research
Email: majinming@pjlab.org.cn, lixiuhong@pku.edu.cn

***Abstract*—To guarantee service quality in transformer based large language model (LLM) serving, it is essential to meet the latency constraints of both the *prefill* phase (measured by Time-to-First-Token, TTFT) and the *decode* phase (measured by Time-per-Output-Token, TPOT). *Non-disaggregated* serving places *prefill* and *decode* on the same worker, while *disaggregated* serving places the *prefill* and *decode* on isolated workers. However, no single architecture excels in both TTFT and TPOT metrics. After conducting a root cause analysis, we concluded that in *disaggregated* LLM serving, *prefill* execution has minimal interference with *decode* execution but result in high queuing times. In contrast, *non-disaggregated* LLM serving effectively reduces queuing times but introduces significant interference between prefills and decodes.**

In order to leverage the best aspects of both *non-disaggregated* and *disaggregated* LLM serving, we have designed and implemented *Tropical*. *Tropical* introduces an sevice-level objectives (SLO)-aware multiplexing strategy that balances the queuing time and the interference, enabling the LLM serving to achieve high TTFT and TPOT SLOs simultaneously. Our evaluation of real-world datasets reveals that *Tropical* outperforms both state-of-the-art *non-disaggregated* and *disaggregated* LLM serving systems, achieving up to $2.09\times$ more requests within a 90% SLO attainment. Specially, compared to the *disaggregated* LLM serving system, *Tropical* improves P90 TTFT performance by $9\times$ with only an 15% reduction in P90 TPOT. Against the *non-disaggregated* LLM serving systems, *Tropical* delivers a $2.8\times$ performance improvement in P90 TPOT while maintaining the same P90 TTFT.

***Index Terms*—LLM Serving, SLO-Aware Scheduling.**

I. Introduction

With the rapid development of the field of natural language processing, large language models (LLMs) [1]–[5] have become increasingly versatile. The transformer based LLM inference can be divided into two phases: the *prefill* and the *decode*. A request first complete *prefill* to generate the first token, and then process the *decode* repeatedly, generating a series of tokens in a streaming manner. In LLM serving, both phases have their own latency metrics that are highly relevant to user experience, which are time-to-first-token (TTFT) for *prefill* and time-per-output-token (TPOT) for *decode*. TTFT measures the time from when the user sends a request to when they first receive feedback. TPOT describes the average time for a single request to output tokens. As users typically expect to receive feedback quickly, optimizing TTFT is crucial. Besides, if TPOT is too long, users will experience delays while reading or processing the generations of LLM, which will affect the smoothness and continuity of the user experience. For a single request, both the TTFT and TPOT service-level objectives (SLOs) must be satisfied in LLM Serving.

LLM serving systems are categorized into *non-disaggregated* LLM serving systems and *disaggregated* LLM serving systems based on whether they separate two phases to isolated workers. *Disaggregated* LLM serving [6]–[9] use isolated workers to serve *prefill* and *decode* dividely. *Non-disaggregated* LLM serving systems [10]–[13] maximize GPU resource utilization by colocating *prefill* and *decode* process in the same worker. However, no single architecture excels in both TTFT and TPOT SLOs attainment. Because of high interference in *non-disaggregated* LLM serving and high queuing time in *disaggregated* LLM serving.

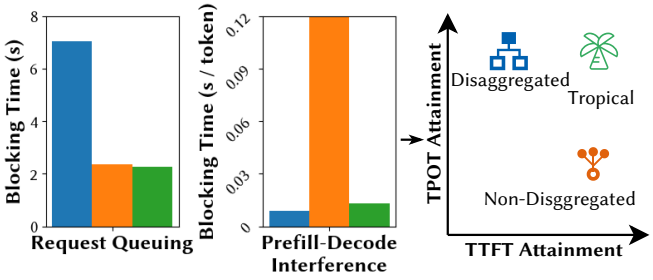


Fig. 1: Comparision among *Non-Disaggregated*, *Disaggregated* LLM Serving and *Tropical*.

Non-disaggregated LLM serving experiences significant interference from *prefill* on *decode*. We evaluate the interference in Fig 1 (b) on a real dataset [8]. We quantize the interference by blocking time, the time a request in *decode* is blocked by other *prefill* executions. Interference in *non-disaggregated* serving is significantly higher than in *disaggregated* serving, leading to higer TPOT SLO violations. Instead, *disaggregated* LLM serving faces the mismatch between the allocation of *prefill-decode* workers and workload requirements. Because *prefill* can only use the resources of the *prefill* workers and cannot utilize the idle resources of the *decode* workers, the *prefill* workers are frequently under high workload, leading to longer queuing times for *prefill*. As shown in Fig 1 (a), the P90 queuing time for *disaggregated* serving is significantly higher than that for *non-disaggregated* serving. In *disaggregated* LLM serving, dynamically switching the role of *prefill* and *decode* is one of the strategies to address the issue of excessive *prefill* queuing times. However, the overhead of switching from *decode* workers to *prefill* workers is not negligible. Besides, the short-term volatility and unpredictability of LLM serving workloads also make it difficult for *prefill-decode* worker role switching to keep up with workload changes.

Based on our observations, we have found that collocating *prefill* and *decode* on the same worker in an SLO-aware manner can also improve the system's SLO attainment without switching overhead, which is called SLO-aware multiplexing. Based on the insight, we have designed and implemented *Tropical*. *Tropical* is a SLO-aware scheduler that manages multiple workers. *Tropical* introduces an multiplexing strategy that effectively reduces queuing time in the scenarios with highly dynamic *prefill* workloads. *Tropical* keeps the interference within an acceptable range that satisfies both TTFT and

[*] Equal contribution. [#] Corresponding author.

TPOT SLOs. As shown in Fig 1, *Tropical* achieves a Pareto optimality for TTFT and TPOT SLOs compared with both kinds of LLM serving.

In summary, this paper makes the following contributions:

1) We dive into the bottlenecks of *disaggregated* and *non-disaggregated* LLM serving and reveals that the main reason for the extended TTFT in *disaggregated* LLM serving is the queueing time. Conversely, the primary cause of the prolonged TPOT in *non-disaggregated* LLM serving is the interference.
2) We employ SLO-aware multiplexing to balance queuing time and interference, enabling the LLM serving to have high TTFT and TPOT SLOs attainment simultaneously.
3) Conduct a comprehensive evaluation of *Tropical* against state-of-the-art *non-disaggregated* and *disaggregated* LLM serving.

Our evaluation reveals that *Tropical* outperforms both state-of-the-art *non-disaggregated* and *disaggregated* LLM serving systems, achieving up to 2.09× more requests within a 90% SLO attainment. Specially, compared to the *disaggregated* LLM serving system, *Tropical* improves P90 TTFT by 9× with only an 15% reduction in P90 TPOT. Against the *non-disaggregated* LLM serving systems, *Tropical* delivers a 2.8× improvement in P90 TPOT while maintaining the same P90 TTFT.

II. Background

A. Prefill-Decode Scheduling

Disaggregated LLM serving has longer queuing times, while non-disaggregated LLM serving has more significant interference. We have drawn a timeline example to illustrate the reasons. Fig 2 (a) represents an *non-disaggregated* serving system [12], where prefills are scheduled eagerly once there are prefills in queue, resulting in a shorter queuing times, which means a shorter TTFT. However, the execution time for prefill is generally longer than that for decode. The execution of prefill will block the execution of decode, leading to a worse TPOT. State-of-the-art *non-disaggregated* LLM serving decompose long prefill into chunks. However, the execution of decode is still slow down because of the interference. Especially, the execution time of the chunks will significantly rise in the long context scenarios. Fig 2 (b) represents a *disaggregated* serving system, where requests in the *prefill* phase and the *decode* phase are executed on isolated workers. Requests in the *decode* phase are not interfered with by the other requests in the *prefill* phase, thus achieving a better TPOT. However, *prefill* can only utilize the resources of one worker, increasing queuing times and resulting in a worse TTFT. In addition, as shown in Fig 2 (b), the slower processing speed of the prefill phase also leads to idle time in the decode phase. Besides, since requests in the *prefill* and *decode* phases cannot be executed in the same batch, the computational resources of the GPU are not fully utilized, leading to *model FLOPs Utilization (MFU) inefficiency*.

B. SLO in LLM Serving

In this section, we define the TTFT SLO, TPOT SLO, and SLO attainment in the LLM serving. In addition, we define *slack* to describe the difference between the actual execution time and the SLO.

Generally, we define TTFT as the time from when the user sends a request to when they first receive feedback. Specially, we define TPOT as the total time spent generating tokens divided by the number of tokens generated, which equates to the average delay between tokens. We posit that users will only perceive waiting when the average generation rate surpasses their reading speed. Consider the following example: a LLM serving system rapidly delivers 20 tokens to the user, then encounters a 1-second pause before resuming token delivery. Although the maximum token-to-token delay for this request is 1

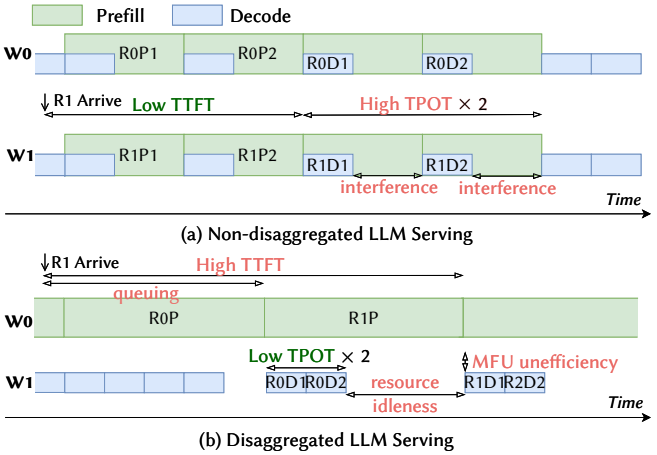


Fig. 2: *Prefill-Decode* scheduling comparison.

second, users still have a sufficient number of tokens to read during the pause, thus maintaining a positive user experience. Conversely, if users do not have enough information to process, they will perceive a lag in response. In *non-disaggregated* LLM serving, the TPOT can be break down to the execution time and the interfernce caused by prefill execution. In *disaggregated* LLM serving, as the *prefill* and *decode* are executed in different workers, the interference of *disaggregated* can be composed into the decode execution and the migration time of KVCache from other prefill workers.

We define *slack* as the gap between system performance and its SLO. When the system's performance metrics exceed the SLO, this additional performance can be considered as slack, which provides a certain buffer, allowing the system to withstand a certain degree of load changes or performance fluctuations without violating the SLO. We utilize the slack during the *decode* phase to handle tasks from *prefill* phase without affecting the user experience.

Let $\mathcal{R}$ be the set of total requests r, L_r be the total generation lengths of r. The request SLO Attainment A is given by the following formulas:

$$\mathcal{R}_{TTFT} = \left\{ T^r_{prefill} \leq TTFT_SLO \mid r \in \mathcal{R} \right\}, \tag{1}$$

$$\mathcal{R}_{TPOT} = \left\{ \frac{\sum T^r_{decode}}{L_r} \leq TPOT_SLO \mid r \in \mathcal{R} \right\}, \tag{2}$$

$$A = \frac{|\mathcal{R}_{TTFT} \cap \mathcal{R}_{TPOT}|}{\mathcal{R}}. \tag{3}$$

III. Characterization and Motivation

In this section, we investigate the characteristics of LLM serving to gain key insights and motivations that will guide our design and implementation of *Tropical*.

A. Workloads in LLM Serving

We observe the dataset [8] to analyze the workload of real-world LLM serving. Fig 3 (a) shows the total number of tokens arrived during an observation gap. It is evident that the arrival of tokens exhibits extremely high uncertainty in the short term. This uncertainty arises from two main factors. First, similar to other DNN serving [14], [15], the serving process itself is random, and user arrivals are unpredictable. Additionally, in the case of LLM serving, the number of input tokens for LLM serving also demonstrates significant dynamics. In Fig 3 (b), we present a scatter plot of a single request's (prefill, decode) sample, which illustrates the input uncertainty. In addition, it

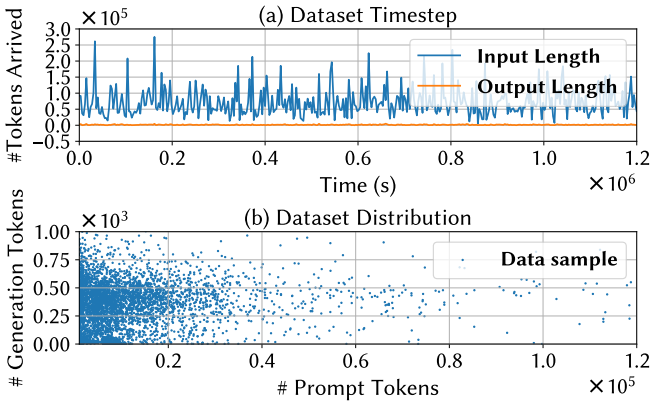


Fig. 3: Mooncake Dataset Token Lengths Over Time and Distribution.

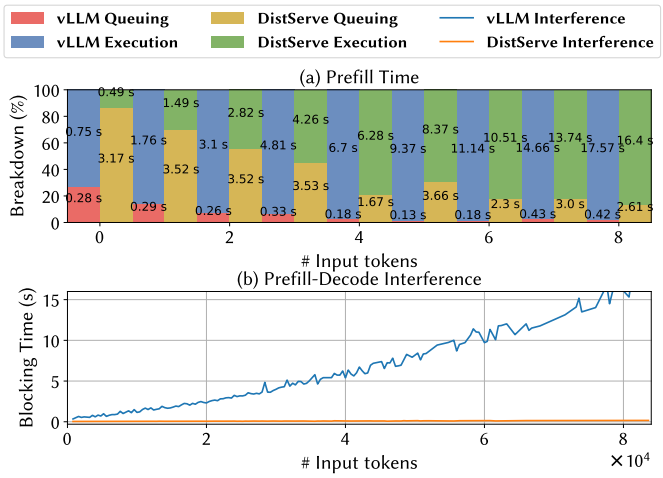


Fig. 4: Queuing and Interference Comparision of Non-Disaggregated Serving and Disaggregated Serving.

can be observed that the distribution of prefill text lengths follows a long-tail pattern. Both Fig 3 (a) and (b) indicate that, compared to the output, the input has a higher dynamic range during the LLM serving process. Similar conclusions can also be drawn from the dataset evaluated in other LLM serving-related works [7].

Characterization I: The lengths of prefill workloads in LLM serving exhibit high dynamism.

B. Efficiency of Non-Disaggregated and Disaggregated Serving

we conduct a quantitative analysis of both *non-disaggregated* and *disaggregated* scheduling strategies. In Fig 4 (a), we analyze the prefill time of the dataset [8] and provided a breakdown to observe the execution time, queuing time, and their proportions for different context lengths. We observed that when the text is short, the TTFT of the *non-disaggregated* LLM serving system is significantly better than that of the disaggregated LLM serving system. Although the decode phase in *non-disaggregated* LLM serving does interfere to the prefill, resulting in slightly higher execution times compared to *disaggregated* LLM serving, the queuing time in *non-disaggregated* LLM serving is much lower than that in *disaggregated* serving. This is particularly noticeable in scenarios with relatively shorter contexts, where the prefill time is dominated by queuing time, which is nearly $10\times$ longer than the execution time.

We observed that when the context is long, the TPOT of the non-disaggregated LLM serving system is significantly worse than

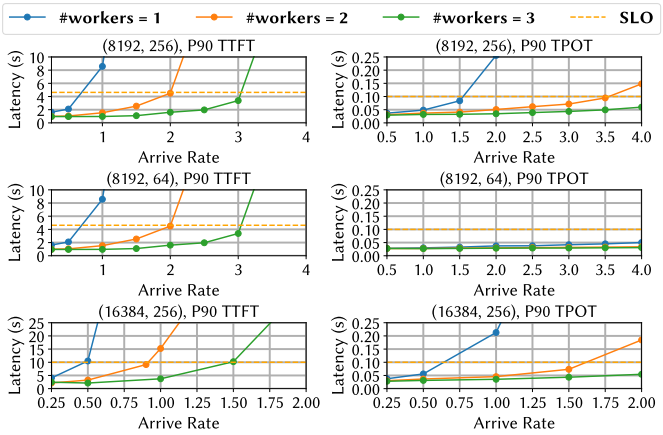


Fig. 5: P90 Latency in Different Worker Resource Allocations.

that of the *disaggregated* LLM serving system. Fig 4 (b) illustrates the interference caused by *prefill* execution on *decode* phase. As the interference primarily arises from the migration of KVCache in *disaggregated* LLM serving and from the prefill execution in *non-disaggregated* LLM serving. With the support of high-bandwidth KVCache transmission, the migration latency is significantly lower than that of prefill execution time. Therefore, the interference in *disaggregated* LLM serving is much less than that in *non-disaggregated* LLM serving. It is also evident that as the text length increases, the effectiveness of *disaggregated* LLM Serving in mitigating *prefill-decode* interference becomes more pronounced.

By combining the insights from Fig 4 (a) and (b), we observe that when the context length is relatively short, the queuing time in *prefill* phase significantly larger than the interference to *decode* phase. This means that, for short texts, the benefits of *disaggregated* LLM serving to mitigate interference are likely outweighed by queuing time.

Characterization II: The *disaggregated* architecture suffers from high queuing times but benefits from interference elimination. In contrast, the *non-disaggregated* architecture has lower queuing times but suffers from interference. Notably, as the length increases, the interference effects in the *non-disaggregated* architecture become more pronounced, while for shorter lengths, the proportion of queuing in the *disaggregated* architecture dominates.

C. Worker Resource Allocation in Disaggregated Serving.

With 4 worker resources, we examine different ratios of worker allocation under three (input, output) length configurations. Fig 5 shows the performance of P90 latency. We observed that it is difficult to match workloads and resources between *prefill* workers and *decode* workers simultaneously. With a configuration of (8192, 256), if we allocate 3 workers for *prefill* and 1 worker for *decode*, the LLM Serving can handle arrival rates of 3.0 for *prefill* and 1.6 for *decode*. In contrast, with 2 workers for *prefill* and 2 for *decode*, *prefill* can accommodate arrival rates of 2.0, while *decode* can handle up to 3.5. Fig 5 also shows the service capability of the LLM is constrained by both the *prefill* and *decode* capacities. According to the principle of the *leaky bucket effect*, static resource allocation for *prefill* and *decode* makes it difficult to achieve an optimal match between resources and workloads. When one phase is overloaded while the other remains idle, the system's performance can deteriorate significantly. In addition, due to the higher computational intensity, *prefill* requires more computing resources compared to *decode* as shown in Fig 5. Related works [7] shows the optimal ratio of *prefill-decode* can be up to 9:1. Besides, Fig 5 also reveals that different samples, such as (8192, 64) and

(16384, 256), have significantly different optimal ratios of prefill and decode workers.

Characterization III: Static ratios of *prefill* and *decode* worker allocation make it difficult to simultaneously match resources and workloads in both phases.

D. Challenge in LLM Serving

We summarize the challenges according to the observation.

Queuing and Interference balancing. From III-B, it can be observed that *non-disaggregated* LLM serving places *prefill* and *decode* on the same worker. However, the execution of *decode* is preempted by *prefill*, leading to Interference. Besides, if *prefill* and *decode* are separated and placed on different workers, *prefill* can only utilize a portion of the resources, causing *prefill* to potentially operate at a high service intensity or even overload, leading to a decline in performance by queuing. Existing LLM serving systems are unable to effectively balance queuing and interference times, making it difficult to simultaneously meet TTFT and TPOT SLOs.

Resource misallocation. According to III-C, even under fixed input-output length configurations, the ratio of prefill to decode can still lead to resource misallocation. From Section III-A, it can be concluded that the length of requests varies greatly. As a result, the *prefill-decode* ratio can also fluctuate significantly. To address the issue of mismatched resource allocation, an effective approach is to implement *prefill-decode* role switching in iteration level [16]. The overhead of role switching from *prefill* to *decode* is minimal, as it only requires requests that have completed the *prefill* phase to continue to the *decode* phase on the same worker, without the need for inter-worker transmission of KV cache. However, when the system performs the switching from *decode* to *prefill*, the system will find itself in a dilemma. Since the *decode* phase is stateful, immediately switching a *decode* worker to a *prefill* worker requires either migration or recomputation, both of which can lead TPOT SLO violations. If ignored, the *prefill* workers will experience overload, leading to the high queuing time. Consequently, achieving a rapid switching from *decode* workers to *prefill* workers is also quite challenging.

E. Optimization Chance

SLO-Aware Multiplexing. As described in II-A, We define *slack* as the gap between system performance and its SLO. When the system's performance metrics exceed the SLO, this additional performance can be considered as slack, which provides a certain buffer, allowing the system to withstand a certain degree of workload changes or performance fluctuations without violating the SLO. In LLM serving scenarios, we can utilize the slack during the *decode* phase to handle tasks from *prefill* phase without affecting the user experience. Whether performing *prefill-decode disaggregation* to eliminate interference or colocating *prefill-decode* into the same worker, the goal is to guarantee requests completion within the *SLO*. In a *non-disaggregated* LLM serving, *prefill* and *decode* sharing the same worker is akin to trading interference for reduced queuing time, while disaggregated LLM serving does the opposite. We term the practice of co-locating *prefill* and *decode* on the same worker while maintaining TTFT and TPOT SLOs as *SLO-aware multiplexing*. If *SLO-aware multiplexing* can be implemented in disaggregated LLM serving, it would be able to reach an equilibrium of queuing time and interference, which enhances the total SLO attainment.

IV. System Design

Based on the observations and the solution presented in section III, we design *Tropical*, an SLO-aware scheduler to balance the interference and queuing time.

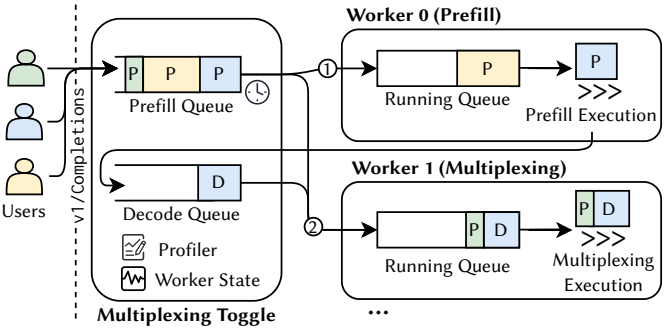


Fig. 6: Overview of *Tropical*.

A. Overview

The design of *Tropical* is illustrated in Fig 6. *Tropical* consists of two parts: the *multiplexing toggle* and a series of *workers*. In *Tropical*, workers can perform both *prefill* and *decode* execution. The phase in which they execute depends on the tasks sent to them by the multiplexing toggle. The multiplexing toggle primarily performs two functions: assignment and dispatching. For assignment, the multiplexing toggle assigns workers as prefill workers or multiplexing workers. For dispatching, It places incoming requests into the prefill queue initially and then dispatches them in an SLO-aware manner to competent workers. We will provide a detailed introduction to the design of the multiplexing toggle and its internal SLO-aware multiplexing mechanism in section IV-C and section IV-B.

B. SLO-Aware Multiplexing

As described in Figure 4, the *disaggregated* architecture suffers from high queuing times but benefits from interference elimination. In contrast, the *non-disaggregated* architecture has lower queuing times but suffers from interference. Additionally, as described in II-B and III-C, decode workers experience slack due to load fluctuations, and we aim to utilize the slack to serve prefills, thereby reducing the queuing time. In order to determine whether the interference from *prefill* execution will disrupt the requests that are currently in the *decode* phase, we will record the difference between the time taken for a single decode operation and the TPOT time, which is known as the TPOT slack in a single iteration.

If the predicted execution time is less than the slack time available in the *decode* worker, we treat the multiplexing worker as idle and schedule the prefill to the multiplexing worker to alleviate the pressure on the *prefill* workers. Although the insertion of *prefill* requests may lead to uneven output latency in decoding, recent advancements in LLM serving technologies [17], [18] aim to reduce fluctuations during the output process, ensuring a smoother flow. As described in Fig 7, the *decode* process to generate the third token of R_0 has accumulated enough slack to accommodate a prefill request. Therefore, *Tropical* routes a request from the prefill queue for execution. Although the prefill is inserted, R_0 does not violate the TPOT SLO.

Due to the significantly shorter execution time of short *prefill* than the long, it is more likely to route the short *prefill* request to multiplexing workers. As we concluded in 4, short *prefill* are queueing dominated, which gives *non-disaggregated* LLM serving an advantage. In contrast, long *prefill* cause greater interference with decoding, which makes *disaggregated* LLM serving outperform. SLO-aware multiplexing leverages the benefits of both serving strategy.

C. Multiplexing Toggle

As described in Fig 6, there are two paths for sending requests to workers. The multiplexing toggle provides a traffic control mechanism:

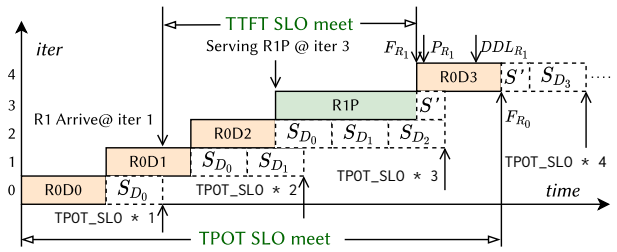


Fig. 7: P-D SLO-aware multiplexing by slack checking.

when the system is dominated by queuing, it follows Path ①, and when interference dominates, it follows Path ②. The multiplexing toggle can dispatch the request through Path ① to *prefill* workers for *prefill* execution, and then enter to the *decode* queue of the multiplexing toggle. The multiplexing toggle will dispatch the request again through Path ② to the multiplexing worker for *decode* execution. Alternatively, requests can bypass Path ① and directly enter the multiplexing workers. When the interference from *prefill* on *decode* remains within acceptable limits as described in IV-B, requests can be sent directly via Path ②. When the interference is not negligible, the *prefill* worker acts as an *interference inhibitor*, reducing the interference to an acceptable range. Since *prefill* can utilize the worker resources of both *prefill* and multiplexing workers, the system's queuing time will be significantly less than that of *disaggregated* serving.

The multiplexing toggle records the status of each worker, including monitoring the HBM watermark and the local queue of workers. Our design consolidates choice in Path ① and ②. Additionally, since the execution time of the LLM's *prefill* phase is highly predictable [19], we leverage offline profiling tools to estimate both the execution time of a *prefill* request and the queuing time when scheduling to the local worker. The multiplexing toggle conservatively sends requests to prefill workers only when the addition of the predicted execution time and the predicted queuing time in the workers, is less than the available slack to TTFT SLO. When a competent worker is found, an scheduling operator is created and requests are dequeued from *prefill* queue and enqueue to the running queue of the local worker.

Tropical also conservatively sends prefill requests to multiplexing workers. In addition to the TPOT slack budget outlined in IV-B, the multiplexing toggle records and predicts the execution time of the decoding batch for multiplexing workers. Given the high predictability of execution times, the multiplexing toggle will not route prefill requests to multiplexing workers when the execution time of the decoding batch is approaching the preset TPOT SLO. Additionally, since HBM usage is a critical factor affecting decoding execution, the multiplexing toggle will not select a multiplexing worker if the HBM usage for decoding exceeds the threshold. Furthermore, for multiplexing workers, we will also utilize chunked prefill to further smooth out the interference that prefill may cause to decoding.

When the watermark of all decode workers exceeds its threshold, indicating insufficient resources for *decode* workers, *prefill* workers can be directly assigned as multiplexing workers. If prefill frequently encounters delays (such as when *prefill* texts are too long), this will also trigger a transition from multiplexing workers back to prefill workers for a higher *prefill* throughput.

V. Evaluation

In this section, we evaluate the efficiency of *Tropical*.

A. Experimental Setup

Models. We choose InternLM-20B [20], an LLM that is popular and performs exceptionally well in long context scenarios, the max supported context window is 200K, which satisfies the requirement of the maximum number of tokens in the dataset.
Testbed. We evaluated *Tropical* on a server equipped with 8 NVIDIA A100 80GB GPUs, with 600 GB/s P2P bandwidth between GPUs.
workloads. We choose Mooncake [8]. It is obtained from the trace of real long context LLM serving.
Metrics. We use SLO attainment as the major evaluation metric. Same as previous relative LLM serving works [7], [21]. We set the TTFT SLO to $5\times$ the execution time of the latency in the corresponding phase under light worload. The other metrics we are concerned with are TTFT and TPOT average and P90 latency. To find the sources of benefit for TTFT, we analyze the queuing time. And we also show the cumulative distribution function (CDF) of the TTFT and TPOT.
Baselines. We compare *Tropical* with the following LLM serving:

- **vLLM** [11] vLLM is a representative *non-disaggregated* LLM serving system widely used in both academia and industry. It colocates the *prefill* and *decode* computation at the same worker and struggles to meet TPOT SLO.
- **vLLM with chunked prefill [12]**. Breaking down the prefill phase into multiple smaller chunks, and executing only one prefill chunk per iteration [12]. This approach can shorten the time prefills interfere decodes. However, this method still cannot effectively control the interference of prefills on decodes. We set the chunk size to 2048 based on the profiling of the workload.
- **DistServe** [6]. We implemented *disaggregated* serving based on vLLM [11], executing prefills and decodes in isolated workers, which is consistent with the implementation in DistServe [6].

Both the baselines and *Tropical* divided the 8 GPUs into 4 workers, configuring the degree of tensor parallelism of each worker to be 2. To evaluate the efficiency of SLO-aware multiplexing, *Tropical* and DistServe all use 2 wrokers for *prefill* and 2 workers for *decode*.

For both baselines and *Tropical*, we adapt the dispatching strategy of InFaas [22] with LLM serving, where the global dispatcher sends requests to the workers with the fewest unfinished token counts. In *Tropical*, the multiplexing toggle keeps track of the slack budget for multiplexing workers. Under the condition of satisfying the SLO, the multiplexing toggle will trigger multiplexing to piggyback *prefill* with the *decode* requests for execution.

B. SLO Attainment

In this section, we compare the end-to-end preformance of *Tropical* against the baselines on real world datasets. Fig 8 (a) illustrates that when we gradually increase the rate, more requests will violate the latency requirements and the SLO attainment decreases. Compared to baselines, *Tropical* has improved performance by serving 2.02x more users. From Fig 8, it can be observed that the performance of chunked-prefill is not optimal. This is due to the fact that as the chunk index increases, the time to compute a chunk also increases, which can probably lead to TPOT SLO violation. *non-disaggregated* LLM serving also has lower SLO attainment than *Tropical*. To evaluate that the SLO violation is caused by queuing delay or interference. We further break down the SLO violation into TTFT and TPOT violation, we show the Pareto frontier diagram in Fig 8 (b), analyzing the changes in TTFT SLO attainment and TPOT SLO attainment for different LLM serving systems with the arrival rate. We found that DistServe has worse TTFT SLO Attainment than others, indicating that the performance of *disaggregated* LLM serving is dominated by the queuing time in the prefill phase, while vLLM has higher TTFT SLO

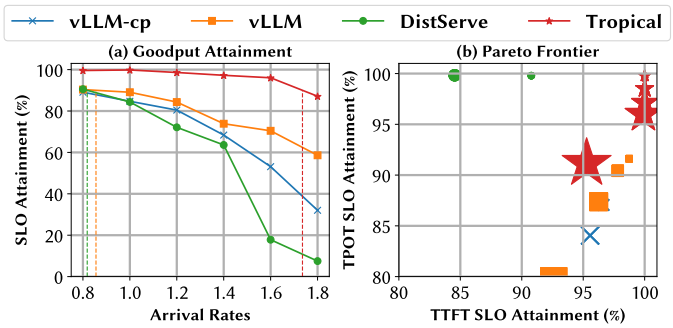


Fig. 8: SLO attainment.

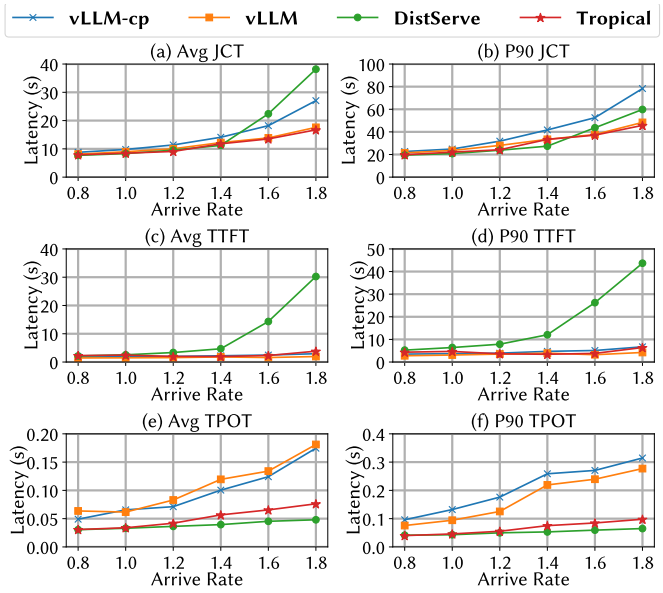


Fig. 9: Average and P90 Latency.

attainment, but compared to DistServe, its TPOT SLO is significantly reduced, suggesting that *non-disaggregated* LLM serving is dominated by *prefill-decode* interference. Evaluation show that the SLO-aware multiplexing can significantly reduce the TTFT queuing time and interference to ensure both TTFT and TPOT SLOs attainment.

C. Latency

As shown in Fig 9 (a) and (b), *Tropical* and vLLM outperform the other two serving systems for total latency. vLLM improves system worker resource utilization by co-location. *Tropical* achieves high worker resource utilization through SLO-aware multiplexing. *Tropical* improves TTFT metric by 9× compared to DistServe. In 9 (e) and (f), the TPOT performance of *Tropical* is up to 2.33× against vLLM and vLLM w/ chunked prefill. The TPOT performance of *Tropical* is slightly lower than DistServe. However, *Tropical* still meets the TPOT SLO for most requests as shown in Fig 8.

D. Queueing Time

To further analyze the sources of TTFT performance optimization against *disaggregated* LLM serving, we profile the queuing times of different LLM serving systems, as shown in Fig 10. Tropical improves P90 queueing time by 9× compared to DistServe. The average queuing time and P90 queuing time of *Tropical* are slight higher than vLLM in high arrival rates. However, as shown in Fig 4, We use prefill workers to reduce the *decode-prefill* interference, therefore, the TTFT for *Tropical* and vLLM are essentially the same.

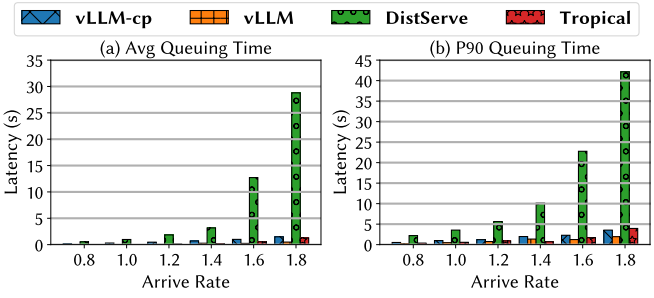


Fig. 10: Queuing Time.

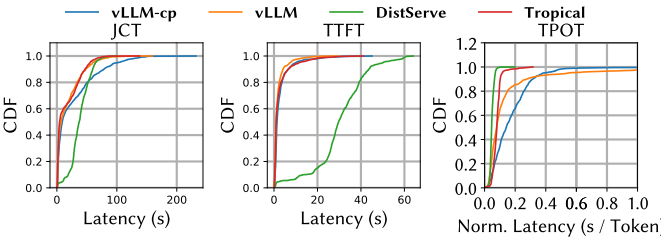


Fig. 11: CDF.

E. CDF

The CDF of serving is shown in Fig 11. It can be observed that, except for *Tropical*, all other serving systems exhibit significant tail latency. Instead, *Tropical* achieves a balance between TTFT and TPOT, thereby improving the overall service quality of the system.

VI. Related Work

LLM Serving. The *non-disaggregated* inference service systems [10], [11], [13], [23], [24], which process the prefill phase and decode phase within the same worker. The *disaggregated* serving system [6]–[8], [25]–[29] handle the prefill phase and decode phase using different workers, eliminating the interference between requests. In the field of LLM serving, kernel optimization is a critical area of research. These optimization techniques demonstrate the diversity and depth of kernel optimization in LLM serving [30], [31]. In the context of Deep Neural Networks (DNNs), there are numerous SLO-aware scheduling strategies [14], [19], [22], [32], [33]. Andes [17] defines a Quality of Experience (QoE) metric and enhanced the service quality for users in the LLM serving scenario by optimizing this metric. DistServe [6] introduced the concept of Goodput and defined the TTFT SLO and TPOT SLO for the LLM serving service scenario. However, it did not consider maintaining a high resource utilization while simultaneously meeting TTFT and TPOT SLOs by SLO-aware multiplexing.

VII. Conclusion

We conducted a detailed analysis of the drawback of existing *disaggregated* LLM serving in TTFT by queuing time, and the deficiencies of existing *non-disaggregated* LLM serving systems in TPOT by interference. We design *Tropical*, a multi-worker scheduler. *Tropical* balances queuing time and interference by SLO-aware multiplexing, enabling the LLM serving to have high TTFT and TPOT SLOs attainment simultaneously.

Acknowledgement

The authors sincerely thank anonymous DAC reviewers for their valuable comments on this paper. The research is supported by the Shanghai Municipal Science and Technology Major Project. Greatly thank Zihan Wang for his insightful feedback and constructive suggestions, which significantly improved the quality and conceptual framework of this work.

REFERENCES

[1] Brown *et al.*, "Language models are few-shot learners," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., 2020.

[2] A. Chowdhery *et al.*, "Palm: scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 1, Mar. 2024.

[3] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[4] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[5] Vaswani *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Red Hook, NY, USA: Curran Associates Inc., 2017.

[6] Y. Zhong *et al.*, "DistServe: Disaggregating prefill and decoding for goodput-optimized large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 193–210. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

[7] P. Patel *et al.*, "Splitwise: Efficient generative llm inference using phase splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 118–132.

[8] R. Qin *et al.*, "Mooncake: A kvcache-centric disaggregated architecture for llm serving," 2024. [Online]. Available: https://arxiv.org/abs/2407.00079

[9] C. Hu *et al.*, "Inference without interference: Disaggregate llm inference for mixed downstream workloads," 2024. [Online]. Available: https://arxiv.org/abs/2401.11181

[10] G.-I. Yu *et al.*, "Orca: A distributed serving system for Transformer-Based generative models," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 521–538. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/yu

[11] Kwon *et al.*, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 611–626. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[12] A. Agrawal *et al.*, "Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 117–134. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/agrawal

[13] L. Zheng *et al.*, "SGLang: Efficient execution of structured language model programs," in *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. [Online]. Available: https://openreview.net/forum?id=VqkAKQibpq

[14] H. Zhang *et al.*, "SHEPHERD: Serving DNNs in the wild," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. Boston, MA: USENIX Association, Apr. 2023, pp. 787–808. [Online]. Available: https://www.usenix.org/conference/nsdi23/presentation/zhang-hong

[15] M. Shahrad *et al.*, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 205–218. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/shahrad

[16] B. Wu *et al.*, "Loongserve: Efficiently serving long-context large language models with elastic sequence parallelism," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, ser. SOSP '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 640–654. [Online]. Available: https://doi.org/10.1145/3694715.3695948

[17] J. Liu *et al.*, "Andes: Defining and enhancing quality-of-experience in llm-based text streaming services," 2024. [Online]. Available: https://arxiv.org/abs/2404.16283

[18] B. Sun *et al.*, "Llumnix: Dynamic scheduling for large language model serving," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 173–191. [Online]. Available: https://www.usenix.org/conference/osdi24/presentation/sun-biao

[19] A. Gujarati *et al.*, "Serving DNNs like clockwork: Performance predictability from the bottom up," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 443–462. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/gujarati

[20] Z. Cai *et al.*, "Internlm2 technical report," 2024.

[21] B. Wu *et al.*, "Fast distributed inference serving for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2305.05920

[22] F. Romero *et al.*, "INFaaS: Automated model-less inference serving," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 397–411. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/romero

[23] J. Duan *et al.*, "Muxserve: Flexible spatial-temporal multiplexing for multiple LLM serving," in *Forty-first International Conference on Machine Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=R0SoZvqXyQ

[24] Z. Li *et al.*, "AlpaServe: Statistical multiplexing with model parallelism for deep learning serving," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 663–679. [Online]. Available: https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

[25] X. Miao *et al.*, "Spotserve: Serving generative large language models on preemptible instances," ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1112–1127. [Online]. Available: https://doi.org/10.1145/3620665.3640411

[26] H. Oh *et al.*, "Exegpt: Constraint-aware resource scheduling for llm inference," ser. ASPLOS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 369–384. [Online]. Available: https://doi.org/10.1145/3620665.3640383

[27] Y. Jin *et al.*, "P/d-serve: Serving disaggregated large language model at scale," 2024. [Online]. Available: https://arxiv.org/abs/2408.08147

[28] F. Strati *et al.*, "Déjàvu: Kv-cache streaming for fast, fault-tolerant generative llm serving," 2024.

[29] C. Hu *et al.*, "Memserve: Context caching for disaggregated llm serving with elastic memory pool," 2024. [Online]. Available: https://arxiv.org/abs/2406.17565

[30] T. Dao *et al.*, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Advances in Neural Information Processing Systems*, vol. 35, pp. 16 344–16 359, 2022.

[31] K. Hong *et al.*, "Flashdecoding++: Faster large language model inference on gpus," *arXiv preprint arXiv:2311.01282*, 2023.

[32] M. Han *et al.*, "Microsecond-scale preemption for concurrent GPU-accelerated DNN inferences," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 539–558. [Online]. Available: https://www.usenix.org/conference/osdi22/presentation/han

[33] R. S. Kannan *et al.*, "Grandslam: Guaranteeing slas for jobs in microservices execution frameworks," ser. EuroSys '19. New York, NY, USA: ACM, 2019. [Online]. Available: https://doi.org/10.1145/3302424.3303958

Gem5-AcceSys: Enabling System-Level Exploration of Standard Interconnects for Novel Accelerators

Qunyou Liu*, Marina Zapater†, David Atienza*,
*Embedded Systems Laboratory (ESL), EPFL, Switzerland, qunyou.liu@epfl.ch, david.atienza@epfl.ch
†University of Applied Sciences and Arts Western Switzerland (HES-SO), Switzerland, marina.zapater@heig-vd.ch

***Abstract*—The growing demand for efficient, high-performance processing in machine learning (ML) and image processing has made hardware accelerators, such as GPUs and Data Streaming Accelerators (DSAs), increasingly essential. These accelerators enhance ML and image processing tasks by offloading computation from the CPU to dedicated hardware. These accelerators rely on interconnects for efficient data transfer, making interconnect design crucial for system-level performance. This paper introduces Gem5-AcceSys, an innovative framework for system-level exploration of standard interconnects and configurable memory hierarchies. Using a matrix multiplication accelerator tailored for transformer workloads as a case study, we evaluate PCIe performance across diverse memory types (DDR4, DDR5, GDDR6, HBM2) and configurations, including host-side and device-side memory. Our findings demonstrate that optimized interconnects can achieve up to 80% of device-side memory performance and, in some scenarios, even surpass it. These results offer actionable insights for system architects, enabling a balanced approach to performance and cost in next-generation accelerator design.**

***Index Terms*—Memory Hierarchy, PCIe, Interconnects, Hardware Accelerators, System-Level Simulation**

I. Introduction

Transformer models, such as Vision Transformer (ViT) [1], have revolutionized machine learning (ML) and natural language processing (NLP), setting new benchmarks in various tasks [2]. Their growing adoption in real-time applications demands efficient and scalable hardware architectures to meet rising computational needs. Systolic arrays have shown promise in accelerating these tasks, offering parallel data flow and high computational throughput ideal for matrix multiplication, central to both traditional ML and transformer-based models, such as Google's Tensor Processing Unit [3] and TiC-SAT [4]. However, most studies on systolic-array-based architectures primarily focus on hardware-level evaluations, often neglecting the broader system-level context. This leaves a gap in understanding how the interconnects and memory hierarchy, which are vital components in moving data between processors and memory, impact the overall system performance. While some system-level evaluations exist, they often lack the use of standard interconnects such as PCIe, and heterogeneous memory hierarchy such as non-uniform memory access (NUMA), reducing their practicality for real-world applications.

To bridge this gap, we introduce Gem5-AcceSys, a novel framework enabling system-level exploration of interconnects and memory hierarchies in accelerators. Using a matrix multiplication accelerator tailored for transformer workloads as a case study, our design framework incorporates essential components such as PCIe [5], [6], NUMA architectures, and configurable memory hierarchies. This integration provides a practical platform for evaluating real-world performance under various configurations. Specifically, we assess the impact of standard interconnects (PCIe) and diverse memory types, including DDR3/4/5, GDDR6, and HBM2, on system efficiency. By emphasizing system-level architecture, our work provides a comprehensive understanding of how interconnects and memory hierarchies influence transformer's performance and how general matrix multiplication (GEMM) and Non-GEMM impact the system performance with the overhead introduced by standard interconnects. This research offers actionable insights for designing scalable and cost-effective accelerator systems tailored to modern ML workloads. More specifically, our contributions are as follows:

1) **Framework for Interconnect Exploration**: We present the framework, Gem5-AcceSys, to support PCIe interconnects, NUMA architecture, and processing near memory based on the Gem5 simulator, enabling system-level co-design with features like device-side memory, local buffer, SMMU, DMA, and RTL-based accelerators.
2) **PCIe Bandwidth impact on Performance Study**: We investigate the impact of varying PCIe bandwidth by adjusting the number of lanes and lane speeds for the target GEMM workload. Additionally, we vary the packet request size from the perspective of the accelerator to identify the optimal packet size for the design.
3) **Memory Access and Address Translation Study**: We analyze the impact of memory type and location on system performance from a system-level perspective. Additionally, we investigate address translation overhead for Transformer workloads, demonstrating the effects of virtual address spaces on accelerator efficiency.
4) **Impact of Non-GEMM and GEMM Workload Analysis**: We conduct a detailed runtime analysis of Transformer workloads, dividing them into two components: GEMM and Non-GEMM. To optimize performance, we profiled these workloads separately and identified thresholds to determine when device-side memory should be utilized.

II. State-of-the-art

Several open-source simulators have been developed to aid in the design and integration of specialized hardware accelerators, notably Gem5-Aladdin [7], Gem5-Salam [8], Gem5-RTL [9], and Gem5-X [10]. These frameworks provide valuable tools for early-stage design exploration and detailed evaluations of accelerators within system simulations. Gem5-Aladdin integrates accelerator modeling with system simulation, enabling pre-RTL design analysis. Gem5-Salam employs LLVM-based

methodologies to model and estimate the performance and area of custom accelerators. Gem5-RTL incorporates RTL designs for detailed performance assessments, while Gem5-X supports simulations of many-core heterogeneous systems with advanced memory technologies. However, these simulators have limitations that hinder their applicability for comprehensive system-level co-design. They rely on simplistic interconnects (e.g., basic buses, shared memory), lacking support for standard interfaces like PCIe. Memory hierarchy support is limited, with features like Non-Uniform Memory Access (NUMA) and processing near memory (PNM) absent [11]. High simulation overhead due to RTL integration, as seen in frameworks like Gem5-RTL, impedes rapid prototyping. Scalability issues arise when accurately simulating diverse accelerators within large-scale systems. Additionally, inadequate system integration is evident from the lack of features like Direct Memory Access (DMA) and System Memory Management Unit (SMMU) for efficient data handling [12], [13]. These limitations highlight the need for a comprehensive framework that supports standard interconnects, complex memory hierarchies, and realistic full-system interactions. To address this gap, we propose **Gem5-AcceSys** to enable detailed system-level co-design of accelerators.

III. Design Framework

A. Overall Architecture and Interface Integration

Fig. 1 illustrates our enhanced design framework, with newly added components highlighted in light blue, significantly expanding the Gem5 simulator's capabilities for contemporary system design. This framework integrates essential functionalities, including PCIe interconnects, DMA, SMMU, configurable sub-memory architecture, and RTL-based accelerator support. The system is organized into two primary sections: the CPU cluster with its cache and the accelerator system interfaced with the memory bus (MemBus) through PCIe components.

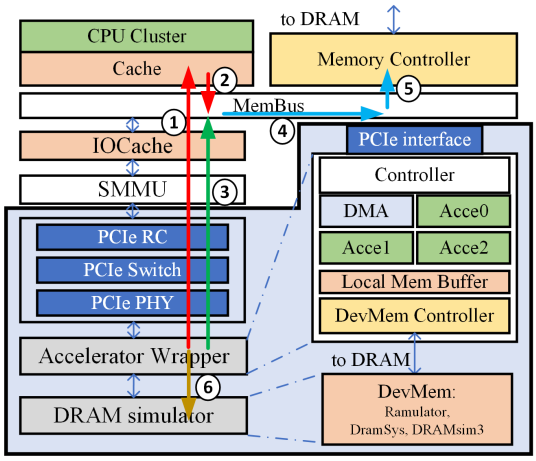


Fig. 1. Design Framework Architecture

The CPU cluster, comprising one or more CPUs and associated caches, connects to the main memory (DRAM) via a memory controller. Data transfers between the CPU and memory are facilitated by a MemBus. PCIe integration provides a realistic interface for peripheral device simulations, surpassing the limitations of conventional memory bus latency models. The DMA feature further enhances system efficiency by enabling direct memory transfers, reducing the data movement burden on the accelerator and simplifying its design. The SMMU enables virtual-to-physical address translation, streamlining driver development and improving memory mapping efficiency for accelerators [12], [13]. A configurable sub-memory system supports the optimization of memory configurations crucial for co-designing accelerator systems. This includes device-side memory with local buffers and host-side memory configurations to evaluate different system trade-offs. Key PCIe components include:

- **PCIe RC (Root Complex)**: The CPU endpoint that manages data and commands on the PCIe bus.
- **PCIe PHY (Physical Layer)**: Connects the RC to devices, facilitating communication via the PCIe Switch and Link.
- **PCIe Switch**: Routes traffic among PCIe devices, supporting multiple connections and enhancing scalability.

By positioning the SMMU between the MemBus and PCIe components, the framework enhances memory protection and address translation capabilities, enabling efficient use of virtual address spaces.

B. Accelerator Wrapper

The Accelerator Wrapper contains logic and interfaces for the hardware accelerator, including a PCIe interface that connects to the accelerator's controller. This controller manages data flows between the accelerator and system. Adjacent to this, the single accelerator or accelerator cluster, either RTL-based or C++-based, integrates with the system using Verilator [14] to convert RTL code to C++, compiling it into an executable that runs as a child process with shared memory system calls. A DMA block within the controller enables direct memory access, bypassing the CPU for higher performance. A Local Mem Buffer provides quick, temporary computation storage, and a Device Memory (DevMem) Controller oversees data transfers between the accelerator and device memory, boosting efficiency. The accelerator also interfaces with a configurable memory hierarchy.

C. Memory Hierarchy and Subsystem Performance Analysis

Our design framework incorporates a configurable memory hierarchy, including host-side and device-side memories, along with a last-level cache and a device-side cache, supporting comprehensive design space exploration. It interfaces with DRAM models like DRAMsim3 [15], Ramulator2 [16], and Dramsys5 [17] for accurate DRAM timing and power statistics. The framework offers three memory access methods: direct cache (DC) access, direct memory (DM) access and DevMem access. Fig. 1 illustrates DC and DM modes. In DC mode, data requests from the accelerator are directed to the cache hierarchy. Cache hits retrieve data immediately, reducing latency, while cache misses access host-side memory via the MemBus (arrow 2) and memory controller (arrows 4 and 5), adding latency. This mode enables detailed analysis of cache effects on performance and allows adjustments to cache size and latency. To support this, we modified the simulator to implement a cache coherency model between the accelerator's cache and the CPU cache. In DM mode, requests bypass the cache and go straight to main memory via arrows 3 and 5, minimizing latency by eliminating

cache checks. This method requires software management of data coherency since it bypasses the cache system. Arrows 6 bypasses the whole PCIe system to access device-side memory avoid the data movement overhead introduce by PCIe system. By leveraging configurable memory hierarchies and flexible access methods, our design framework enables the selection of optimal memory configurations to enhance accelerator performance. Additional insights into the impact of memory hierarchy on performance are discussed in Section V.

D. Comparison with Existing Simulators

To highlight the unique features of Gem5-AcceSys and how it extends beyond existing Gem5-based simulators, we provide a concise comparison in Table I. This table summarizes key features such as accelerator (Acce) design level support, interconnect capabilities, address translation, memory simulation, kernel driver support, DMA capabilities, device-side memory support, simulation scope, and accelerator process models.

As shown in Table I, Gem5-AcceSys provides comprehensive support for features essential in modern accelerator design, addressing limitations present in existing simulators. This includes support for standard interconnects like PCIe, accelerator address translation via SMMU, integration with external memory simulators, kernel driver support, multi-channel DMA, and device-side memory.

IV. Experimental Setup

To demonstrate the capabilities of our design framework and analyze the impact of standard interconnects and configurable memory architectures on transformer workloads, we conduct experiments using the DC access method, with configurations and parameters detailed in Table II.

A. Hardware Accelerator Design

We use a Systolic Array (SA) accelerator named MatrixFlow [18]. MatrixFlow is a loosely-coupled SA specifically optimized for transformer models. It uses a new matrix multiplication technique alongside an optimized data structure, which significantly enhances data streaming efficiency and reduces memory overhead. MatrixFlow contains 16x16 multiply–accumulate units, and uses data in integer format as input and output. Through a combination of hardware-software co-design, including the integration of PCIe and DMA for efficient data handling, MatrixFlow achieves remarkable speed-ups of up to 400x compared to a single-threaded CPU system, and demonstrates substantial performance improvements over both loosely- and tightly-coupled accelerators in the state-of-the-art.

B. Workload

First, we select GEMM as a workload because it represents a general and widely used computational pattern, showcasing the versatility of our design framework in evaluating performance and analyzing bottlenecks. Next, to further illustrate our framework's capabilities in co-optimizing complex systems, we use the Vision Transformer (ViT) [1] as a case study. We evaluate workloads from the ViT_base, ViT_large, and ViT_huge models, which have hidden dimensions of 768, 1024, and 1280, respectively, and utilize 12 or 16 attention heads.

V. Performance Evaluation

To evaluate the performance of our design framework, we focus first on GEMMs and analyze its performance under various configurations. Based on the insights gained from this analysis, we narrow-down the set of interesting configurations and focus on the entire transformer.

A. Performance Bounds and Roofline Model

We begin by analyzing the roofline model of our accelerator system, focusing on GEMMs with dimensions 1024. To isolate the effects of computation on performance, we fix the PCIe bandwidth at 8 GB/s and vary the computation time of the systolic array within the Gem5 framework.

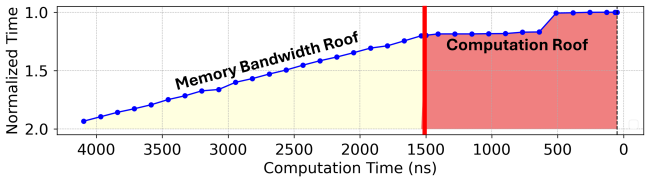


Fig. 2. Roofline Model of the Accelerator System

Fig. 2 illustrates the roofline model of the proposed accelerator. The x-axis represents computation time (ns), while the y-axis shows the normalized execution time. For computation times exceeding 1500 ns, the system operates in the **memory-bound** region, where performance improves linearly as computation time decreases. This indicates that performance is limited by memory bandwidth or PCIe data transfer speed. As computation time drops below 1500 ns, the system transitions to the **computation-bound** region, as shown by the plateau in performance. Here, further improvements in memory bandwidth have no significant effect because the computation speed of the systolic array becomes the bottleneck. This transition, marked by the red line, highlights the shift in limiting factors. To better understand these performance bounds, we proceed with a detailed analysis of GEMM workloads.

B. GEMMs

We start by analyzing the impact of the following factors:

- **PCIe Link:** Analyzing how PCIe bandwidth, latency , and packet sizes affect the data transfer efficiency and overall computation speed.
- **Memory Type, location, latency and bandwidth:** Assessing performance variations between device-side and host-side memory, under different memory technologies including DDR4, LPDDR, GDDR, and HBM.
- **Address Translation:** Analysing the overhead of Address Translation wrt matrix size and their effects on computational delays and resource utilization.

1) PCIe Link Performance Analysis:

a) Bandwidth Sweeping: To evaluate the impact of PCIe bandwidth on GEMM performance, we vary the number of lanes (2, 4, 8, 16) and their speeds (2 Gbps to 64 Gbps), using 2048x2048 matrices on the systolic array accelerator. As depicted in Fig. 3, increasing bandwidth consistently reduces execution time, with performance scaling until the system transitions from memory-bound to compute-bound at 16 lanes. Notably, bandwidth impacts performance significantly, with the highest bandwidth outperforming the lowest by up to

TABLE I
COMPARISON OF GEM5-BASED FRAMEWORKS FOR HARDWARE ACCELERATOR SIMULATION

Feature	Gem5-Aladdin	Gem5-SALAM	Gem5-RTL	Gem5-X	Gem5-AcceSys
Acce Design Level	C++	LLVM IR	RTL	C++	C++, RTL
Interconnect	Basic buses	Basic buses	Basic buses	Basic buses	Basic buses, PCIe
Acce Address Translation	Yes	No	No	No	Yes (SMMU)
External Memory simulator	No	No	No	No	Ramulator/DRAMsys
Kernel Driver Support	No	No	No	Limited	Yes
Multi-Channel DMA	Yes	No	No	No	Yes
Device-Side Memory	No	No	No	Yes	Yes
Full-System Simulation	Yes	Bare-metal	Yes	Yes	Yes
Acce Process Model	Integrated	Integrated	Integrated	Integrated	Child process (Multi-threaded)

TABLE II
SYSTEM CONFIGURATION

Component	Specification
CPU	ARM, 1 GHz
Data Cache	64 kB
Instruction Cache	32 kB
Last Level Cache	2 MB
IOCache	32 kB
Memory	DDR3_1600_8x8, 4 GB
PCIe Link	Version 2.0, 4 Gb/s, 4 Lanes
PCIe RootComplex	150ns Latency
PCIe Switch	50ns Latency

1109.9%. This highlights the effectiveness of our framework in identifying optimal bandwidth configurations for balanced performance and resource efficiency.

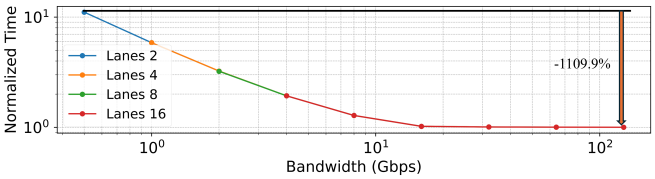


Fig. 3. Performance (Execution time) for Matrix Size 2048 under varying per-lane bandwidth and number of lanes

Key Takeaway #1: **PCIe bandwidth significantly impacts accelerator performance, particularly in memory-bound regions. However, as systems become compute-bound, the benefits of additional bandwidth diminish.**

b) Packet Size Sweeping: We configure the PCIe Link at 4GB/s, 8GB/s, 16GB/s, 32GB/s, and 64GB/s, analyzing the impact of request packet sizes from accelerator ranging from 64 bytes to 4096 bytes. As shown in Fig. 4, varying packet sizes result in differing processing times, even reach 36%. Initially, execution time decreases as packet size increases from 64 bytes, reaching a minimum around 256 bytes, reflecting efficiency gains at moderate sizes. Beyond this point, execution time increases with larger packet sizes up to 4096 bytes, forming a convex curve that highlights a non-linear relationship: both very small and very large packet sizes are less efficient. This behavior stems from the PCIe hierarchy, where packets must pass through the RootComplex and Switch. Larger packets disrupt the pipeline, causing stalls at each component before data reaches the accelerator.

Key Takeaway #2: **Packet size significantly affects execution time with a convex trend: 64-byte packets incur 12% overhead, and 4096-byte packets 36%, relative to the optimal 256-byte size.**

2) Memory performance analysis: To examine the performance impact of different memory types and locations, we

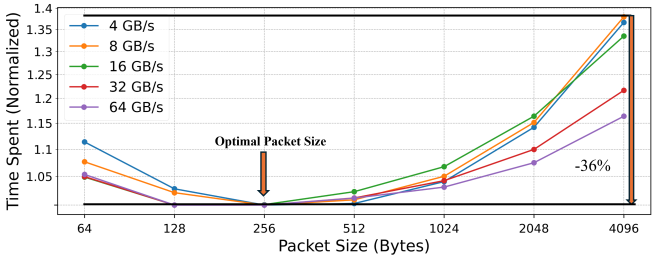


Fig. 4. Execution Time under different packet sizes for different PCIe bandwidth.

use ramulator2 as our backend DRAM model, and test DDR3, DDR4, HBM, and GDDR5 with the bandwidth and data rate configurations detailed in Table III.

TABLE III
MEMORY CONFIGURATION

Component	Channel	Data width	Bandwidth	Data Rate
DDR3	1	64	12.8 GB/s	1600 MT/s
DDR4	1	64	19.2 GB/s	2400 MT/s
DDR5	2	32	25.6 GB/s	3200 MT/s
HBM2	2	128	64 GB/s	2000 MT/s
GDDR6	2	64	32 GB/s	2000 MT/s

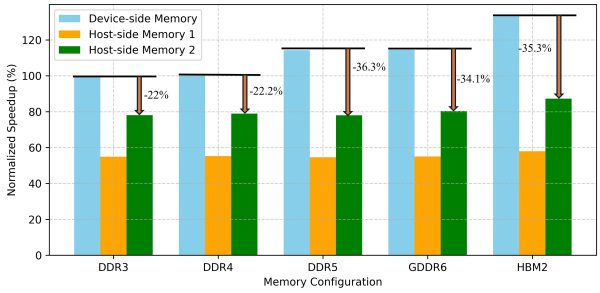


Fig. 5. Impact of DRAM type and location

a) Device-side vs Host-side Memory and Memory Type: Fig. 5 presents the normalized speedup (wrt DDR4 device-side data) comparison between device-side memory and two host-side memory configurations (one with a 2GB/s and one with a 64GB/s PCIe link) for DDR4, HBM, GDDR5, and LPDDR5.

The results demonstrate that device-side memory consistently outperforms host-side memory across all tested memory types, regardless of the speed of the PCIe bus. Host-side memory shows lower speedups, with a clear dependence on PCIe speed. When using the 64GB/s PCIe configuration, host-side memory can achieve around 78% of the performance relative to device-side memory, which is a substantial improvement over the 2 GB/s PCIe. Performance degradation depends on the memory

technology, and is aggravated for GDDR5 and HBM, as the gap between host-side and device-side memory grows larger.

Key Takeaway #3: **Implementing device-side memory significantly boosts performance, improving accelerator efficiency by up to 2 times compared to all other configurations.**

b) Memory bandwidth and latency sweeping: We investigate the impact of memory bandwidth and latency by varying one while keeping the other one constant, using HBM as a case study with gem5's default DRAM model.

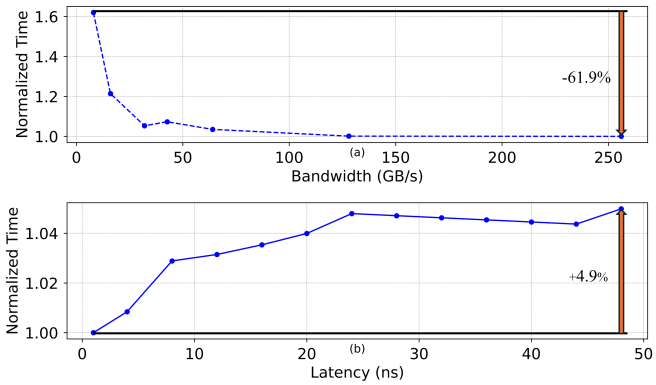


Fig. 6. Impact of Memory Bandwidth(a) and Memory Latency(b)

Fig. 6(a) shows that normalized execution time decreases significantly as bandwidth increases up to approximately 50GB/s, yielding up to a 60% performance improvement. This underscores the critical role of bandwidth in alleviating performance bottlenecks at lower levels. Beyond 100GB/s, the curve plateaus, with only a 1.7% improvement from 50GB/s to 256GB/s, thus other system components become the bottleneck.

Fig. 6(b) illustrates that execution time increases with latency, especially sharply from 1ns to 12ns, then plateaus between 12ns and 36ns. However, the overall time overhead due to increased latency is only about 4.9%. This analysis indicates that while both bandwidth and latency affect performance, the system is significantly more sensitive to changes in bandwidth. Thus, bandwidth is the more crucial factor in GEMM-dominated workloads.

Key Takeaway #4: **Bandwidth has a greater impact than latency in GEMM performance: bandwidth improves performance by 60% and latency only adds 5% overhead.**

3) Address Translation performance analysis: In this section, we analyze the impact of virtual address translation on system performance, focusing on larger matrix sizes. Table IV summarizes key metrics such as memory footprint, translation times, and page table walk (PTW) times.

As matrix size increases, translation overhead initially decreases, reaching 1.00% at 1024, but rises to 6.49% at 2048. This reflects how larger matrices reduce per-translation costs through amortization but face increasing complexity with larger address spaces. Similarly, the mean translation time drops to its lowest value at 1024 but spikes to 54.38 cycles at 2048, indicating inefficiencies. PTW times also increase with matrix size, reaching 368.14 cycles at 2048, highlighting the growing complexity of address translation for very large datasets. This highlights the critical need for careful optimization of address translation mechanisms to maintain system performance, particularly for large-scale workloads.

Key Takeaway #5: **Address translation overhead significantly impacts performance for large memory footprints. Larger matrices increase translation complexity and page table walk times, requiring optimization to avoid bottlenecks.**

C. Transformer Performance Evaluation

We apply the GEMM analysis insights to Transformer inference using four system configurations:

1) A system using host memory and small PCIe bandwidth of 2 GB/s with 4 4Gbps lanes (PCIe-2GB).
2) A system using host memory and moderate PCIe bandwidth of 8 GB/s with 8 8Gbps lanes (PCIe-8GB).
3) A system using host memory and PCIe bandwidth of 64 GB/s with 16 64Gbps lanes (PCIe-64GB).
4) A system without host memory, utilizing device-side memory instead (DevMem).

We select ViT as the target workload and configure each system based on the GEMM analysis:

- For PCIe-2GB and PCIe-8GB: packet size of 256 and DDR4 memory.
- For PCIe-64GB: packet size of 256 and HBM2 memory.
- For DevMem: packet size of 64 and HBM2 memory.

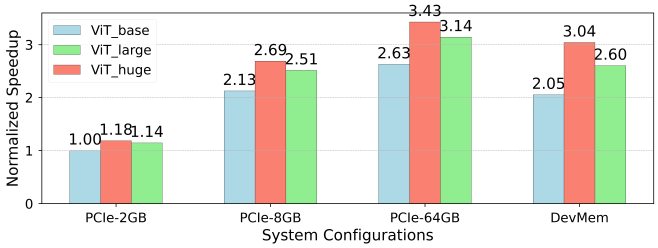


Fig. 7. Performance comparison of memory locations and interconnects

Fig. 7 shows that PCIe-64GB achieves significant performance improvements over the baseline PCIe-2GB, with speedups ranging from approximately 2.5x to 3.4x, highlighting the critical impact of PCIe bandwidth when using host memory. In contrast, the DevMem configuration, despite leveraging device-side memory (HBM2 with burst size of 64) and reduced data transfer times due to proximity to computational units, performs slightly worse than PCIe-64GB.

D. Transformer Workload Analysis

1) GEMM and Non-GEMM Performance Evaluation: To understand the performance discrepancies observed, we conduct a detailed analysis of the Transformer workload, profiling both GEMM and Non-GEMM operations across different ViT models and system configurations, as highlighted in prior research [19], [20]. Fig. 8 shows that while DevMem offers the best performance for GEMM workloads due to its high data bandwidth, it exhibits the poorest performance in Non-GEMM operations, incurring up to a 500% overhead compared to systems using the PCIe interface. This degradation stems from the high latency introduced by the NUMA architecture and longer access times to device-side memory. Consequently,

TABLE IV
RESULTS WITH LARGER MATRIX SIZES

Metric	64	128	256	512	1024	2048
Memory Footprint (Pages)	12.0	48.0	192.0	768.0	3072.0	12288.0
Translation Times	3130	18470	142738	1082780	8593259	68430699
Trans Mean Time	23.42683	20.37948	13.87159	9.91735	10.478634	54.38005
PTW Times	15	54	227	1034	7675	479244
PTW Mean Time	176.6666	281.90740	265.255507	252.465184	294.609381	368.141137
uTLB Lookup times	2350	17690	137290	1081610	8586250	68423690
uTLB Misses times	195	862	8644	65808	731513	10416279
Trans Overhead	6.02%	1.87%	1.59%	1.30%	1.00%	6.49%

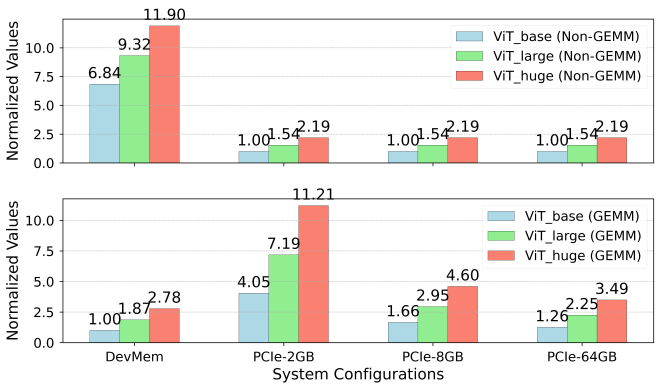


Fig. 8. Performance Comparison of GEMM and Non-GEMM workload

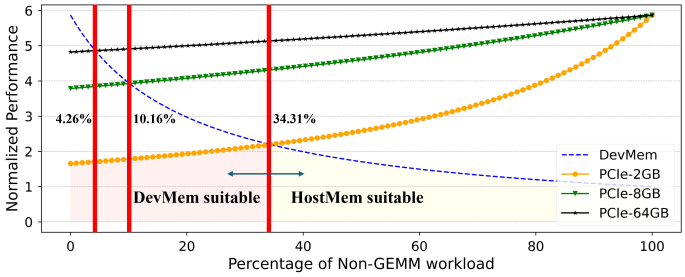


Fig. 9. Overall Transformer Performance as a Function of Non-GEMM Workload Fraction for various PCIe Bandwidths vs DevMem

despite DevMem's advantages in GEMM tasks, these overheads hinder system performance in Non-GEMM computations.

Key Takeaway #6: **Device-side memory overhead from NUMA architecture can degrade performance, accounting for 40% in Transformer experiments like ViT_large.**

2) Performance Analysis of GEMM and Non-GEMM Workloads: To quantify the performance trade-offs between GEMM and Non-GEMM workloads, we propose a model that expresses the total execution time of Transformers as a combination of these two components. By understanding their respective contributions and computational requirements, we can determine the optimal balance between device-side and host-side memory utilization. The total execution time $\text{Time}_{\text{overall}}$ of a Transformer workload can be expressed as:

$$\text{Time}_{\text{overall}} = T_{\text{other}} + \frac{W_{\text{GEMM}}}{P_{\text{GEMM}}} + \frac{W_{\text{NonGEMM}}}{P_{\text{NonGEMM}}}$$

Where T_{other} is the fixed time for other operations. W_{GEMM} is the fraction of GEMM workload ($0 \leq W_{\text{GEMM}} \leq 1$). P_{GEMM} and P_{NonGEMM} are the performance metrics for GEMM and Non-GEMM workloads, respectively. In our case, the Non-GEMM percentage represents the proportion of overall time spent on Non-GEMM workloads when executed on a PCIe system configuration.

We vary the W_{NonGEMM} from 0% to 100% to analyze the performance variance. Fig. 9 shows the performance variation with increasing Non-GEMM percentage. Our analysis reveals that **DevMem** outperforms the PCIe system when the GEMM workload fraction (W_{GEMM}) exceeds certain thresholds, which decrease as PCIe bandwidth increases. Specifically, DevMem is preferable when W_{GEMM} exceeds 34.31% for a PCIe bandwidth of 2 GB/s, 10.16% for 8 GB/s, and 4.27% for 64 GB/s. This indicates that as PCIe bandwidth increases, the advantage of using DevMem diminishes unless the workload is overwhelmingly dominated by GEMM operations.

Key Takeaway #7: **The choice between PCIe and DevMem depends on workload composition (GEMM vs. Non-GEMM) and PCIe bandwidth; DevMem is preferred when Non-GEMM fractions are below a threshold.**

VI. CONCLUSION

In this paper, we have introduced Gem5-AcceSys, a comprehensive framework that extends the Gem5 simulator to support standard interconnects like PCIe, NUMA architectures, and configurable memory hierarchies. This advancement addresses critical limitations of existing simulators, enabling detailed system-level co-design and realistic performance evaluation of hardware accelerators. Using Gem5-AcceSys, we conduct an in-depth analysis of a matrix multiplication accelerator tailored for transformer workloads. Our experiments demonstrate that optimized PCIe interconnects can achieve up to 80% of the performance of systems using device-side memory, and in some cases, even surpass them. We also uncover trade-offs between GEMM and Non-GEMM workloads, highlighting how their balance influences the optimal choice of memory configurations and interconnect strategies. By facilitating the exploration of standard interconnects and memory hierarchies, Gem5-AcceSys provides actionable insights for system architects, empowering them to balance performance and cost in designing efficient and scalable next-generation accelerator systems.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback, Pengbo Yu for hardware and manuscript guidance, and Gabriel Catel Torres and Clément Dieperink for software support.

This work was supported in part by the Swiss State Secretariat for Education, Research, and Innovation (SERI) through the SwissChips research project, and also by Intel as part of the Intel Center for Heterogeneous Integrated Platforms (HIP).

REFERENCES

[1] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proc. ICLR*, 2021.

[2] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proc. NAACL-HLT*, 2019, pp. 4171–4186.

[3] N. P. Jouppi *et al.*, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proc. 44th Annu. Int. Symp. on Computer Architecture (ISCA)*, Toronto, ON, Canada, Jun. 2017, pp. 1–12.

[4] AA. Amirshahi, J. A. H. Klein, G. Ansaloni, and D. Atienza, "TiC-SAT: Tightly-Coupled Systolic Accelerator for Transformers," in *Proc. 28th Asia and South Pacific Design Automation Conf. (ASP-DAC)*, Tokyo, Japan, Jan. 2023, pp. 657–663.

[5] J. Ajanovic, "PCI Express 3.0 Overview," in *Proc. 2009 IEEE Hot Chips 21 Symposium*, 2009, pp. 1–61.

[6] M. Vasa, C.-L. Liao, S. Kumar, C.-H. Chen, and B. Mutnury, "PCIe Gen-5 Design Challenges of High-Speed Servers," in *Proc. IEEE Conf. on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, 2020, pp. 1–3. DOI: 10.1109/EPEPS48591.2020.9231458.

[7] Y. S. Shao, S. L. Xi, V. Srinivasan, G.-Y. Wei, and D. Brooks, "Co-designing accelerators and SoC interfaces using gem5-Aladdin," in *Proc. 49th Annu. IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, Taipei, Taiwan, 2016, pp. 1–12. DOI: 10.1109/MICRO.2016.7783751.

[8] S. Rogers, J. Slycord, M. Baharani, and H. Tabkhi, "gem5-SALAM: A System Architecture for LLVM-based Accelerator Modeling," in *Proc. 53rd Annu. IEEE/ACM Int. Symp. on Microarchitecture (MICRO)*, 2020, pp. 471–482. DOI: 10.1109/MICRO50266.2020.00047.

[9] A. Armejach, G. López-Paradís, and M. Moretó, "gem5 + RTL: A Framework to Enable RTL Models Inside a Full-System Simulator," in *Proc. 50th Int. Conf. on Parallel Processing (ICPP)*, 2021, pp. 1–11. DOI: 10.1145/3472456.3472461.

[10] Y. M. Qureshi, W. A. Simon, M. Zapater, K. Olcoz, and D. Atienza, "Gem5-X: A Many-Core Heterogeneous Simulation Platform for Architectural Exploration and Optimization," *ACM Trans. Archit. Code Optim.*, vol. 18, no. 4, Art. 44, Dec. 2021, 27 pages. DOI: 10.1145/3461662.

[11] A. A. Khan, J. P. C. De Lima, H. Farzaneh, and J. Castrillon, "The Landscape of Compute-near-memory and Compute-in-memory: A Research and Commercial Overview," *arXiv:2401.14428*, 2024.

[12] J. Whitham and N. Audsley, "Studying the Applicability of the Scratchpad Memory Management Unit," in *Proc. IEEE Real-Time and Embedded Technology and Applications Symp. (RTAS)*, 2010, pp. 205–214. DOI: 10.1109/RTAS.2010.21.

[13] K. Paraskevas, K. Iordanou, M. Luján, and J. Goodacre, "Analysis of the Usage Models of System Memory Management Unit in Accelerator-Attached Translation Units," in *Proc. Int. Symp. on Memory Systems (MEMSYS)*, 2020, pp. 86–96. DOI: 10.1145/3422575.3422781.

[14] Y. Chi, X. Lin, and X. Zheng, "Design of a High-Performance SoC Simulation Model Based on Verilator," in *Proc. 5th Int. Conf. on Algorithms, Computing and Artificial Intelligence (ACAI)*, 2022, pp. 1–6. DOI: 10.1145/3579654.3579751.

[15] S. Li, Z. Yang, D. Reddy, A. Srivastava, and B. Jacob, "DRAMsim3: A Cycle-Accurate, Thermal-Capable DRAM Simulator," *IEEE Comput. Archit. Lett.*, vol. 19, no. 2, pp. 106–109, Jul.–Dec. 2020. DOI: 10.1109/LCA.2020.2973991.

[16] H. Luo, Y. C. Tuğrul, F. N. Bostancı, A. Olgun, A. G. Yağlıkçı, and O. Mutlu, "Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator," *arXiv:2308.11030v2*, 2023.

[17] L. Steiner, M. Jung, F. S. Prado, K. Bykov, and N. Wehn, "DRAMSys4.0: An Open-Source Simulation Framework for In-Depth DRAM Analyses," *Int. J. Parallel Program.*, vol. 50, no. 2, pp. 217–242, Apr. 2022. DOI: 10.1007/s10766-022-00727-4.

[18] Q. Liu, M. Zapater, and D. Atienza, "MatrixFlow: System-Accelerator Co-Design for High-Performance Transformer Applications," *arXiv preprint arXiv:2503.05290*, 2025.

[19] A. Ivanov, N. Dryden, T. Ben-Nun, S. Li, and T. Hoefler, "Data Movement Is All You Need: A Case Study on Optimizing Transformers," *arXiv:2007.00072*, 2020.

[20] R. Karami, S.-C. Kao, and H. Kwon, "NonGEMM Bench: Understanding the Performance Horizon of the Latest ML Workloads with NonGEMM Operations," *arXiv:2404.11788v3*, 2024.

NoiseZO: RRAM Noise-Driven Zeroth-Order Optimization for Efficient Forward-Only Training

Shuqi Wang*, Zhengwu Liu*†, Chenchen Ding, Chen Zhang, Taiqiang Wu, Jiajun Zhou and Ngai Wong†
Department of Electrical and Electronic Engineering, The University of Hong Kong.
{*sqwang26, dingcc, czhang6, takiwu*}*@connect.hku.hk*, {*zwliu, jjzhou, nwong*}*@eee.hku.hk*

***Abstract*—Compute-in-memory using emerging resistive random-access memory (RRAM) demonstrates significant potential for building energy-efficient deep neural networks. However, RRAM-based network training faces challenges from computational noise and gradient calculation overhead. In this study, we introduce NoiseZO, a forward-only training framework that leverages intrinsic RRAM noise to estimate gradients via zeroth-order (ZO) optimization. The framework maps neural networks onto dual RRAM arrays, utilizing their inherent write noise as ZO perturbations for training. This enables network updates through only two forward computations. A fine-grained perturbation control strategy is further developed to enhance training accuracy. Extensive experiments on vowel and image datasets, implemented with typical networks, showcase the effectiveness of our framework. Compared to conventional complementary metal-oxide-semiconductor (CMOS) implementations, our approach achieves a 21-fold reduction in energy consumption.**

I. Introduction

Artificial intelligence advancements are driving rapid increases in neural network size and complexity, challenging traditional complementary metal-oxide-semiconductor (CMOS)-based hardware using von Neumann architecture [1]. Moreover, Moore's Law is approaching physical limits, impeding further CMOS transistor miniaturization [2]. Compute-in-memory (CIM) architecture, particularly using resistive random-access memory (RRAM), emerges as a promising solution [3]. RRAM-based CIM effectively addresses the von Neumann bottleneck by mapping neural network weights onto RRAM conductances and performing matrix-vector multiplications through physical laws, demonstrating significant improvements across various network architectures [4]–[8].

Despite the demonstrated success of RRAM-based network inference, training of RRAM-based neural networks remains a significant challenge. Traditional back-propagation (BP) faces two critical limitations in RRAM hardware: first, the inherent device noise in RRAM degrades precision during both forward and backward stages [9], leading to less accurate weight updates and unstable training convergence; second, BP's gradient computation introduces significant hardware overhead [10], as it requires layer-by-layer error propagation and extensive differential calculations, consuming substantial computational resources.

Recent studies have explored solutions to these challenges. One approach is in-situ training with one-step RRAM programming method, which mitigates the effects of RRAM noise on accuracy [6]. To achieve more efficient gradient computations, threshold or sign-based BP was implemented in hardware [11], [12]. Additionally, an activity difference-based method was proposed to train RRAM-based recurrent neural networks [13]. However, these approaches still face certain limitations: they require extensive backward computations and treat RRAM noise as an adverse effect to be mitigated, often resulting in extra time and energy overheads.

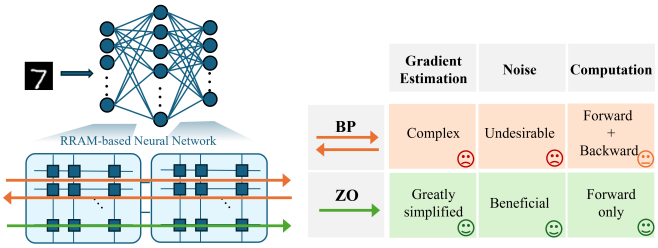


Fig. 1. Comparison of BP and ZO approach in terms of gradient estimation, noise and computation for RRAM-based neural network training.

To address these limitations, zeroth-order (ZO) optimization [14] offers a promising alternative for RRAM-based training. The derivative-free and hardware-friendly nature of ZO optimization enables gradient estimation through only forward passes, making it particularly suitable for efficiently training hardware neural networks [15]. This approach has already shown success in training optical neural networks (ONNs) [16], [17], achieving efficient gradient estimation through parallel mini-batch processing.

Building on these insights, we propose a novel forward-only RRAM training framework that leverages the intrinsic device characteristics in ZO optimization to estimate neural network gradients. Our main contributions are:

- We propose an RRAM-based intrinsic noise-driven ZO optimization (NoiseZO) framework, which transforms intrinsic RRAM noise from an undesirable limitation into a beneficial perturbation for ZO optimization, achieving a noise-robust training approach.
- We enhance NoiseZO's accuracy and convergence speed through the introduction of a fine-grained approach, enabling more precise control of perturbation.
- We validate our method's effectiveness through experiments on vowel and image datasets across typical networks.
- Our RRAM-based method demonstrates superior effi-

* Equal contributions, † Corresponding authors.

ciency, achieving a 21-fold reduction in energy consumption compared to conventional CMOS-based network training.

II. Preliminary

A. RRAM-based Networks: Principles and Noise

RRAM is a type of non-volatile memory that supports CIM architecture, where processing and memory units are co-located. RRAM-based CIM hardware offers efficient matrix-vector multiplication capabilities, which are core computations in various neural networks. Its successful implementations span a diverse range of networks: from single-layer networks for word classification [4] and grey-scale face recognition [5], to multi-layer architectures for the Modified National Institute of Standards and Technology (MNIST) handwritten digit recognition [6], and more complex structures including long short-term memory [8] and convolutional networks [7].

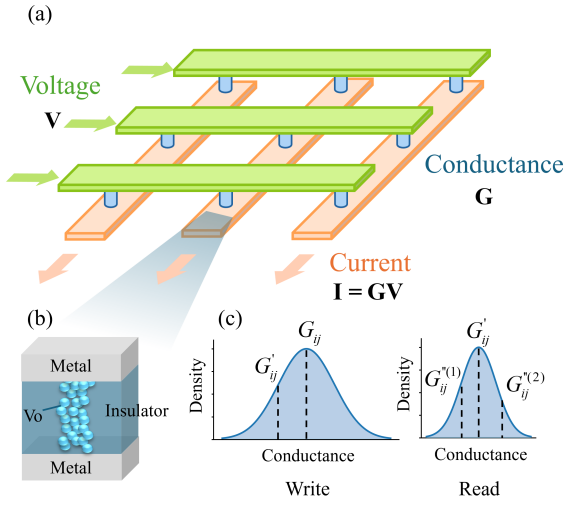


Fig. 2. (a) Matrix-vector multiplication implementation on RRAM array. (b) MIM structure of a typical RRAM device. (c) Write and read noise distributions in RRAM devices.

The principle of RRAM-based matrix-vector multiplication is illustrated in Fig. 2(a). In an RRAM array, each device's conductance G forms a matrix $\mathbf{G}$, with values either quantized multi-bit or fully analog. Applying a voltage vector $\mathbf{V}$ in parallel generates an output current vector $\mathbf{I}$ via Ohm's and Kirchhoff's laws: $\mathbf{I} = \mathbf{GV}$, enabling efficient single-step matrix-vector multiplication. Fig. 2(b) shows one type of RRAM with a metal-insulator-metal (MIM) structure, where conduction relies on conductive filaments (CFs) in the insulator. The formation of these CFs involves the generation and migration of oxygen ions (O) and vacancies (Vo), processes inherently associated with randomness. Therefore, noise is intrinsic during both RRAM write (map) and read operations. Lab measurements show that both write and read noises follow Gaussian distribution [18]. The actual written conductance $G_{ij}^{'}$ relative to target conductance G_{ij} follows a distribution with write deviation σ_w: $G_{ij}^{'} \sim \mathcal{N}(G_{ij}, \sigma_w^2)$, and the subsequent conductance read follows a distribution centered at $G_{ij}^{'}$ with read deviation σ_r: $G_{ij}^{''} \sim \mathcal{N}(G_{ij}^{'}, \sigma_r^2)$.

B. ZO Optimization

ZO optimization, unlike first-order (FO) methods like BP, estimates gradients through function evaluations, requiring only two forward passes [19]. This enables ZO to tackle problems where gradients are unavailable or costly, such as black-box model defense [20], on-chip optical network training [16], [17], and language model fine-tuning [15]. ZO methods can also be integrated with optimizers like SGD [21]–[23] and Adam [24], [25].

The core of ZO optimization is to estimate gradients by difference methods, primarily forward and central difference [19]. For example, using central difference with SGD optimizer, the gradient for a neural network with parameters $\boldsymbol{W} \in \mathbb{R}^d$, mini-batch input $\mathcal{B}$ and loss function $\mathcal{L}(\cdot)$ can be estimated as:

$$\widehat{\nabla}\mathcal{L}(\boldsymbol{W}; \boldsymbol{u}, \mathcal{B}) = \frac{\mathcal{L}(\boldsymbol{W} + \epsilon\boldsymbol{u}; \mathcal{B}) - \mathcal{L}(\boldsymbol{W} - \epsilon\boldsymbol{u}; \mathcal{B})}{2\epsilon}\boldsymbol{u}, \quad (1)$$

where $\boldsymbol{u} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$ and $\epsilon > 0$ is the perturbation radius. This formulation decomposes gradient into two components: the first fraction term computes the gradient magnitude, while the second term $\boldsymbol{u}$ determines the direction of gradient descent. This estimation is actually the directional derivative along $\boldsymbol{u}$. The unbiasedness of this gradient estimation has been theoretically proven [26], [27]. While smaller ϵ theoretically reduces estimation error, practical implementation requires balancing convergence rate, accuracy, and local optima escaping capability. The estimation variance, which scales with parameter dimension d, can be reduced through multiple-sample averaging.

III. Proposed Training Framework

A. Dual RRAM Array Framework for Forward-Only Training

We propose a training framework that leverages RRAM inherent noise as the stochastic perturbation required in ZO optimization, effectively turning a potential drawback into a computational advantage.

To leverage RRAM's inherent write noise for ZO optimization, we propose a dual-array framework. As illustrated in Fig. 3, when mapping identical target weight to two RRAM arrays with the same scale and connectivity, the inherent write noise naturally creates slight different conductance values in each array. This intrinsic variation provides the stochastic perturbation required by ZO optimization, eliminating the need for explicit gradient computation. Through conductance read operations, we obtain the discrepancy between the actual written weights in both arrays. Besides, feeding identical inputs to both arrays produces different outputs and corresponding loss values. These conductance matrix differences, together with loss differentials, serve as the key components for gradient estimation in ZO optimization.

B. Noise-Driven ZO Optimization (NoiseZO)

Consider a target weight $\mathbf{W} \in \mathbb{R}^d$. When the perturbation comes from the intrinsic write noise (standard deviation σ_w)

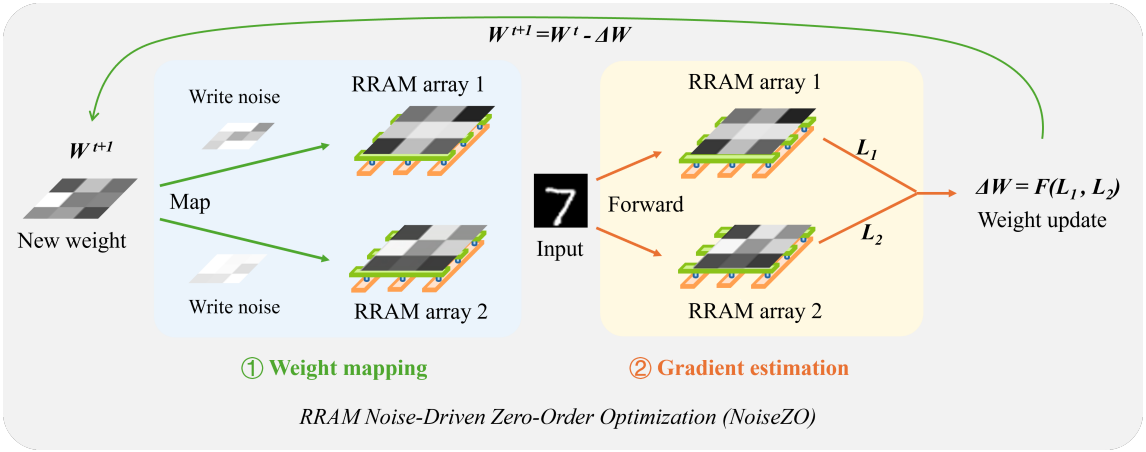


Fig. 3. Overview of the proposed noise-driven ZO optimization framework: iterative weight mapping and gradient estimation cycles enabled by intrinsic RRAM noise for forward-only RRAM-based neural network training.

of two RRAM arrays, the gradient estimation in noise-driven ZO optimization can be theoretically derived as follows:

$$\widehat{\nabla}\mathcal{L}(\boldsymbol{W};\mathcal{B}) = \frac{\mathcal{L}(\boldsymbol{W}+\boldsymbol{\xi}_1;\mathcal{B}) - \mathcal{L}(\boldsymbol{W}+\boldsymbol{\xi}_2;\mathcal{B})}{\sqrt{2}\sigma_w} \cdot \frac{\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2}{\sqrt{2}\sigma_w}, \tag{2}$$

where $\boldsymbol{\xi}_1$ and $\boldsymbol{\xi}_2 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_w^2)$ denote the write noise of two arrays. The actual written weight matrices in arrays are $\boldsymbol{W}_1^{'} = \boldsymbol{W}+\boldsymbol{\xi}_1$ and $\boldsymbol{W}_2^{'} = \boldsymbol{W}+\boldsymbol{\xi}_2$. For the gradient direction term (i.e. second fraction term), we construct the perturbation in ZO optimization as $(\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2)/\sqrt{2}\sigma_w$. Since the weights difference between arrays follows $\boldsymbol{W}_1^{'} - \boldsymbol{W}_2^{'} = \boldsymbol{\xi}_1 - \boldsymbol{\xi}_2 \sim N(\mathbf{0}, 2\boldsymbol{\sigma}_w^2)$, normalizing by $\sqrt{2}\sigma_w$ yields a perturbation that follows the same standard normal distribution as $\boldsymbol{u}$ in Eq. (1). For the gradient magnitude term (i.e. first fraction term), ZO in Eq. (1) deliberately adds opposite perturbations $\pm\epsilon\boldsymbol{u}$ to the weights, creating a parameter difference of $2\epsilon\boldsymbol{u}$ that follows $\mathcal{N}(\mathbf{0}, 4\epsilon^2)$. In contrast, our noise-driven ZO optimization utilizes inherent write noise from RRAM arrays, where two independent samples generating a natural parameter difference $\boldsymbol{\xi}_1 - \boldsymbol{\xi}_2 \sim \mathcal{N}(\mathbf{0}, 2\sigma_w^2)$. Accordingly, we estimate the gradient magnitude by normalizing the loss difference with $\sqrt{2}\sigma_w$. In essence, this gradient estimation employs a random direction as the update direction, with its magnitude determined by the projection of the true gradient onto this direction.

The above only considers the write noise during weight mapping. In practice, two additional sources of read noise affect the system: one occurs when applying input voltages to RRAM arrays during forward computation, and another appears when reading the conductance after the forward pass. After incorporating these read noise components, the gradient estimation becomes:

$$\widehat{\nabla}\mathcal{L}(\boldsymbol{W};\mathcal{B}) = \frac{\mathcal{L}(\boldsymbol{W}+\boldsymbol{\xi}_1+\boldsymbol{\zeta}_1^1;\mathcal{B}) - \mathcal{L}(\boldsymbol{W}+\boldsymbol{\xi}_2+\boldsymbol{\zeta}_2^1;\mathcal{B})}{\sqrt{2\sigma_w^2 + 2\sigma_r^2}} \cdot \frac{(\boldsymbol{\xi}_1+\boldsymbol{\zeta}_1^1+\boldsymbol{\zeta}_1^2) - (\boldsymbol{\xi}_2+\boldsymbol{\zeta}_2^1+\boldsymbol{\zeta}_2^2)}{\sqrt{2\sigma_w^2 + 4\sigma_r^2}}, \tag{3}$$

where $\boldsymbol{\zeta}_1^1, \boldsymbol{\zeta}_2^1 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_r^2)$ represent the read noise generated in the forward passes, and $\boldsymbol{\zeta}_1^2$ and $\boldsymbol{\zeta}_2^2 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_r^2)$ denote the read noise during conductance reads. For notation simplicity, we define $\boldsymbol{W}_1^{''} = \boldsymbol{W}+\boldsymbol{\xi}_1+\boldsymbol{\zeta}_1^1, \boldsymbol{W}_2^{''} = \boldsymbol{W}+\boldsymbol{\xi}_2+\boldsymbol{\zeta}_2^1$ as the effective weights during forward passes, $\boldsymbol{W}_1^{'''} = \boldsymbol{W}+\boldsymbol{\xi}_1+\boldsymbol{\zeta}_1^1+\boldsymbol{\zeta}_1^2$, $\boldsymbol{W}_2^{'''} = \boldsymbol{W}+\boldsymbol{\xi}_2+\boldsymbol{\zeta}_2^1+\boldsymbol{\zeta}_2^2$ as the read-out weights during conductance reads. For the magnitude term in Eq. (3), since the difference between two perturbed weights follows $\boldsymbol{W}_1^{''} - \boldsymbol{W}_2^{''} = (\boldsymbol{\xi}_1+\boldsymbol{\zeta}_1^1) - (\boldsymbol{\xi}_2+\boldsymbol{\zeta}_2^1) \sim \mathcal{N}(\mathbf{0}, 2\boldsymbol{\sigma}_w^2 + 2\boldsymbol{\sigma}_r^2)$, we set the denominator as $\sqrt{2\sigma_w^2 + 2\sigma_r^2}$. For the direction term, since $(\boldsymbol{\xi}_1+\boldsymbol{\zeta}_1^1+\boldsymbol{\zeta}_1^2) - (\boldsymbol{\xi}_2+\boldsymbol{\zeta}_2^1+\boldsymbol{\zeta}_2^2) \sim \mathcal{N}(\mathbf{0}, 2\boldsymbol{\sigma}_w^2 + 4\boldsymbol{\sigma}_r^2)$, normalizing by $\sqrt{2\sigma_w^2 + 4\sigma_r^2}$ yields standard normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$.

The proposed training algorithm is detailed in Algorithm 1. Unlike conventional training approaches that update weights directly on the hardware at the conclusion of each iteration, our method initiates each iteration with a write operation to two distinct RRAM arrays. This step is crucial as the resulting noise difference between arrays serves as the driving source for ZO optimization. Subsequently, we perform forward passes on both RRAM arrays and obtain their actual weights through conductance reads. By leveraging the properties of the RRAM arrays' write and read noise, we can seamlessly integrate the principles of ZO optimization into our training framework, providing a practical solution for gradient estimation in hardware-constrained environments. This gradient estimation accounts for the complete hardware implementation flow, making it feasible for practical implementation.

C. Fine-grained ZO Optimization

Since the complexity of ZO optimization grows with parameter dimensionality, we propose fine-grained ZO optimization that controls optimization granularity to enhance training efficiency and effectiveness. ZO optimization perturbs all the parameters simultaneously in a single query, treating the parameter space as a whole. This unified treatment yields only one gradient estimate, causing degraded convergence and accuracy as dimensionality increases. To address this limita-

Algorithm 1 NoiseZO

Require: parameters $\boldsymbol{W} \in \mathbb{R}^d$, loss function $\mathcal{L}(\cdot)$, write noise scale σ_w, read noise scale σ_r, total iterations T, training dataset $\mathcal{D}$, batch size B, learning rate schedule $\{\eta_t\}$

Initialize $\boldsymbol{W}^0$ randomly
for $t = 0, ..., T-1$ **do**
  Sample batch $\mathcal{B} \subset \mathcal{D}$
  $\boldsymbol{W}_1^{t'}, \boldsymbol{\xi}_1 \leftarrow AddNoise\ (\boldsymbol{W}^t, \sigma_w)$ ▷ Array$_1$: write
  $\boldsymbol{W}_1^{t''}, \boldsymbol{\zeta}_1^1 \leftarrow AddNoise\ (\boldsymbol{W}_1^{t'}, \sigma_r)$ ▷ forward
  $\mathcal{L}_1 \leftarrow \mathcal{L}(\boldsymbol{W}_1^{t''})$
  $\boldsymbol{W}_1^{t'''}, \boldsymbol{\zeta}_1^2 \leftarrow AddNoise(\boldsymbol{W}_1^{t''}, \sigma_r)$ ▷ read
  $\boldsymbol{W}_2^{t'}, \boldsymbol{\xi}_2 \leftarrow AddNoise\ (\boldsymbol{W}^t, \sigma_w)$ ▷ Array$_2$: write
  $\boldsymbol{W}_2^{t''}, \boldsymbol{\zeta}_2^1 \leftarrow AddNoise\ (\boldsymbol{W}_2^{t'}, \sigma_r)$ ▷ forward
  $\mathcal{L}_2 \leftarrow \mathcal{L}(\boldsymbol{W}_2^{t''})$
  $\boldsymbol{W}_2^{t'''}, \boldsymbol{\zeta}_2^2 \leftarrow AddNoise\ (\boldsymbol{W}_2^{t''}, \sigma_r)$ ▷ read
  $\|\hat{\nabla}\mathcal{L}_{\text{zo}}\| \leftarrow \frac{\mathcal{L}_1 - \mathcal{L}_2}{\sqrt{2\sigma_w^2 + 2\sigma_r^2}}$
  $\boldsymbol{W}^{t+1} \leftarrow \boldsymbol{W}^t - \eta_t \times \|\hat{\nabla}\mathcal{L}_{\text{zo}}\| \times \frac{(\boldsymbol{\xi}_1 + \boldsymbol{\zeta}_1^1 + \boldsymbol{\zeta}_1^2) - (\boldsymbol{\xi}_2 + \boldsymbol{\zeta}_2^1 + \boldsymbol{\zeta}_2^2)}{\sqrt{2\sigma_w^2 + 4\sigma_r^2}}$
end for

Subroutine: $AddNoise\ (\boldsymbol{W},\ \sigma)$
for $W_i \in \boldsymbol{W}$ **do**
  $noise_i \sim \mathcal{N}(0, \sigma^2)$
  $W_i^* \leftarrow W_i + noise_i$
end for
$\boldsymbol{noise} \leftarrow [noise_1, noise_2, \ldots, noise_d]^T$
return $\boldsymbol{W^*}, \boldsymbol{noise}$

tion, we propose fine-grained ZO optimization that divides parameters into subgroups and estimates gradients separately for each group. This decomposition allows independent estimation $\hat{\nabla}\mathcal{L}_{zo}$ for different parameter subgroups, leading to more precise gradient approximations. The individual estimates are then aggregated to form the complete gradient. By focusing on parameter subgroups rather than the entire space, our approach achieves more accurate gradient estimation while maintaining controllable query complexity.

- **Layer-wise ZO – for general architectures**: A structure-agnostic approach that optimizes the network layer by layer. If the network has n layers, each batch requires n queries to complete the parameter updates.
- **Component-wise ZO – for specific layers**: A specialized approach that operates on finer granularity within each layer, with strategies tailored to different layer types.
  – Neuron-wise ZO (for fully-connected layers): For a $M \times N$ weight matrix, we treat the $M \times 1$ weight vector connecting to each output neuron as a component, leading to N queries per layer.
  – Channel-wise ZO (for convolutional layers): For a $C \times K \times K$ kernel, each channel-wise kernel ($1 \times K \times K$) serves as a component, requiring C queries per kernel to preserve feature detector integrity.

D. Quantization and Sparsity

Weight quantization is essential prior to RRAM mapping, despite RRAM's inherently analog conductance. This process reduces write overhead and enhances network robustness to noise, optimizing the efficiency and reliability of RRAM-based neural network implementations. To ensure compatibility with practical RRAM hardware, we incorporate weight quantization into the training process. While this increases training complexity, it enables direct deployment on RRAM arrays with hardware constraints.

Besides, we also propose an NoiseZO variant that implements sparsified updates by modifying only a certain percentage of parameters. In a random direction within a multidimensional space, components with larger absolute values contribute more significantly to the update. For instance, with sparsity $s = 0.9$, updating only the top 10% of components by magnitude achieves the most crucial updates while reducing write cost to one-tenth of the original.

IV. Experimental Results

We conduct experiments on MNIST dataset [28] and a vowel dataset [29]. Specifically, a two-layer multi-layer perceptron (MLP) with (784-50-10) is trained for MNIST, and a three-layer MLP with (11-16-16-4) is trained for a subset of vowel recognition dataset. All experiments are conducted using Pytorch on a server with equipped with two NVIDIA GeForce RTX 3090 GPUs and AMD Ryzen Threadripper PRO 3975WX CPUs. We adopted an exponential learning rate decay strategy with a decay factor of 0.97 applied every 10 epochs, and a weight decay of 1×10^{-4}. This paper focuses on the more challenging task of training from scratch. Unlike fine-tuning, where pre-trained models provide a good "initialization" [30], training from scratch is more difficult. ZO optimization starting with random initialization often leads to convergence difficulties [31].

A. Training RRAM-based Network with NoiseZO

Fig. 4 demonstrates the training process and performance of our proposed method on MNIST using MLP (784-50-10). Specifically, Fig. 4(a) shows the evolution of weight distributions and heatmaps in the first fully-connected layer, where the range of conductance-based weights distribution gradually expands during training. As shown in Fig. 4(b), while our basic NoiseZO achieves an accuracy of 88.89%, approaching the baseline ZO (90.42%) [14], [15], our improved fine-grained NoiseZO surpasses it with 95.58% accuracy. Fig. 4(d) presents the confusion matrix of fine-grained NoiseZO on the test dataset, where digit '1' achieves the highest classification accuracy at 99%.

Our NoiseZO incorporates both write and read noises during forward computation for loss calculation, and constructs perturbations using noise differences introduced by an additional read process. This approach introduces an unavoidable estimation error due to the inconsistency between forward pass and read operations, as they are separate processes with different noise compositions. Consequently, NoiseZO inherently suffers

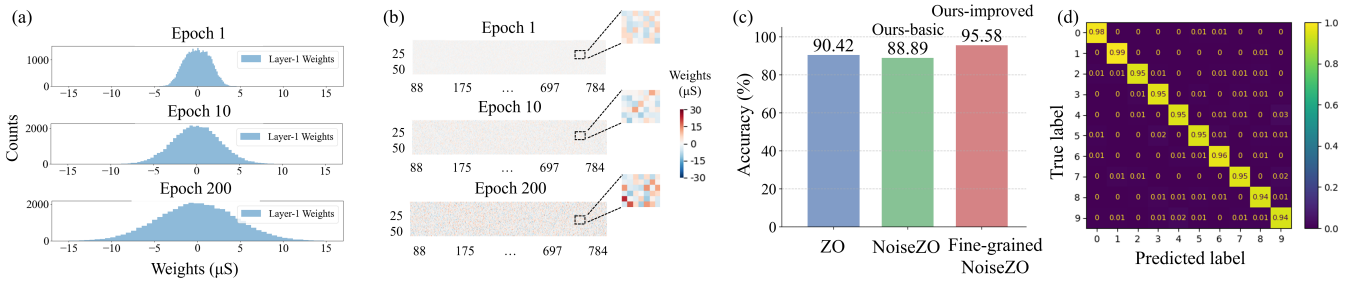


Fig. 4. Training of RRAM-based network using NoiseZO on MNIST. (a) Weight distribution evolution (NoiseZO). (b) Weight heatmap evolution (NoiseZO). (c) Comparison of test accuracy with ZO. (d) Final confusion matrix (Fine-grained NoiseZO).

from an unavoidable additional read noise interference, leading to slightly degraded performance compared to standard ZO. Fortunately, the read noise scale is typically smaller than write noise, making this method robust to this unideal interference.

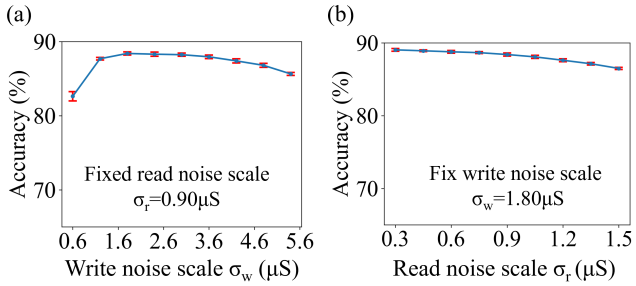


Fig. 5. Impact of varying noise scale on final test accuracy after training for MNIST classification. (a) Fixed read noise scale σ_r at 0.9 μS. (b) Fixed write noise scale σ_w at 1.8 μS.

We further explore the impact of both read and write noise scales on training performance. As shown in Fig. 5, the training effectiveness varies with different write and noise noise scales σ_w, σ_r. Each σ_w and σ_r combination was trained with five random experiments to eliminate the influence of randomness. When fixing read noise scale at 0.9 μS, the training effectiveness maintains stability across a certain range of write noise scale, with optimal results achievable within this range. In contrast, with write noise scale fixed at 1.8 μS, performance shows a slight but consistent degradation as read noise scale increases, aligning with our previous analysis of read noise interference. Through the above analysis, both σ_w and σ_r influence the training process in NoiseZO. Fortunately, in actual RRAM devices, read noise typically exhibits smaller scale than write noise, and noise scales can be moderately controlled through adjustments. Therefore, despite read noise interference, RRAM-based NoiseZO can achieve comparable accuracy to standard ZO.

B. Comparison of Different Methods

As an initial exploration of ZO optimization in RRAM, we validate our method on small-scale datasets. Given ZO's proven capability in large language models (LLMs) [15], the proposed approach, being the first ZO method specifically designed for RRAM hardware, shows promise for more complex tasks and larger networks. We focus on training from scratch in this paper, which presents greater convergence challenges than fine-tuning pre-trained models but better validates our method's training capability. As shown in Tab. I, fine-grained NoiseZO ultimately achieves 95.58% and 98.61% accuracy on MNIST and vowel tasks respectively, surpassing ZO and approaching standard BP performance while outperforming existing BP-based methods [6], [32] for RRAM-based neural networks. These advantages in both performance and hardware efficiency make our approach attractive for practical RRAM implementations.

TABLE I
PERFORMANCE OF TRAINING FROM SCRATCH: BP, ZO, NOISEZO, AND FINE-GRAINED NOISEZO ON MNIST AND VOWEL.

Dataset	Model	BP	ZO Methods		
			ZO	NoiseZO	Fine-grained NoiseZO
MNIST	MLP	97.57%	90.42%	88.89%	**95.58%**
Vowel	MLP	98.61%	94.44%	91.67%	**98.61%**

C. Evaluation on Fine-grained ZO

To evaluate the effectiveness of fine-grained NoiseZO, we trained under different noise scales, as shown in Fig. 6. For NoiseZO in Fig. 6(a), increasing both read and write noise scales leads to slower convergence and lower accuracy, consistent with our previous analysis of read noise interference. In contrast, fine-grained NoiseZO in Fig. 6(b) demonstrates faster convergence and higher accuracy across different noise scales, while exhibiting enhanced robustness to noise scale. These results confirm the effectiveness of dimension splitting in improving performance under various noise conditions.

D. Evaluation on Quantization and Sparsity

Furthermore, we evaluate the training performance with quantization, as shown in Tab. II. Note that 5-bit weight precision corresponds to 4-bit RRAM devices, as RRAM weights are derived from the difference of two conductance values. With 4-bit RRAM, our method achieves 88.78% and 87.5% accuracy on MNIST and vowel, demonstrating fine-grained NoiseZO's effectiveness in quantized training. Achieving this without pre-training, despite quantization errors and random initialization, also indicates our method's broad potential.

TABLE II
QUANTIZED TRAINING ACCURACY RESULTS FOR FINE-GRAINED NOISEZO METHOD

Dataset	Model	Full Precision	Quantization Bits				
			5-bit	6-bit	7-bit	8-bit	9-bit
MNIST	MLP (784-50-10)	95.58%	88.78%	91.67%	93.79%	94.61%	94.72%
Vowel	MLP (10-16-16-4)	98.61%	87.5%	86.11%	93.05%	93.05%	94.44%

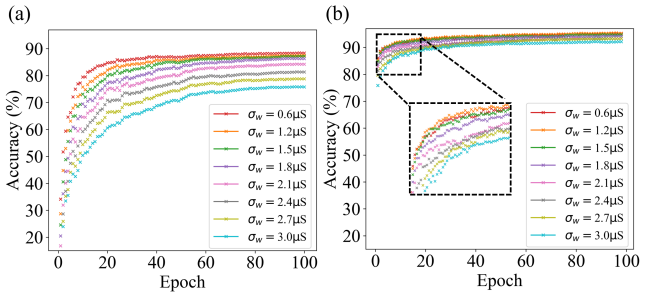


Fig. 6. Training curves of (a) NoiseZO and (b) fine-grained NoiseZO on MNIST classification, with write-read noise ratio $\sigma_w : \sigma_r = 2 : 1$.

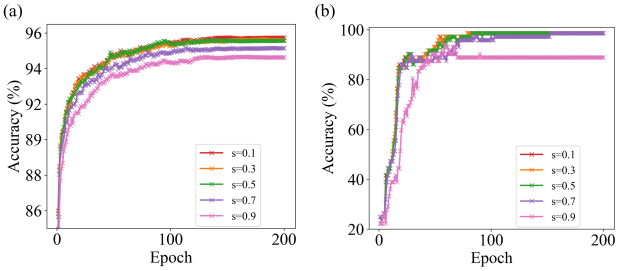


Fig. 7. Training curves under different sparsity levels s using fine-grained NoiseZO. (a) MLP (784-50-10) on MNIST. (b) MLP (10-16-16-4) on vowel recognition.

As shown in Fig. 7, test accuracy gradually decreases with increasing sparsity. Test accuracy remains comparable to full parameter updates when sparsity level is below 0.5, suggesting potential 50% reduction in mapping costs with negligible accuracy loss.

E. Benchmark of Energy Consumption

In this section, we analyze and compare the energy consumption of NoiseZO with that of BP on current state-of-the-art (SOTA) CMOS-based application-specific integrated circuits (ASICs) [33]–[36]. We take the training of a two-layer MLP (784-50-10) on MNIST as an example.

NoiseZO's energy consumption comprises two components: forward computations and network updating. We construct a simulation system for energy estimation, following the general practices in literature [37]–[39]. The system for implementing NoiseZO contains RRAM array and peripheral circuits which include analog-to-digital converters (ADC). Fig. 8(a) illustrates the energy breakdown of the RRAM-based NoiseZO training method, encompassing both the forward and update processes. And thanks to the efficient network updating method in NoiseZO, the energy consumption caused by updating is relatively smaller than forward. To calculate energy consumption of BP, we use the reported energy efficiency with related operation numbers to estimate the energy consumption of CMOS ASICs.

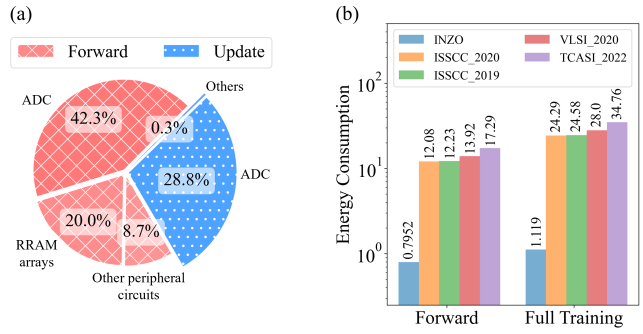


Fig. 8. Energy Consumption. (a) Energy breakdown of RRAM-based NoiseZO. (b) Comparison between RRAM-based NoiseZO training and ASIC-based BP training.

Fig. 8(b) illustrates the energy consumption comparison between the proposed RRAM-based NoiseZO and CMOS ASIC-based BP for this task. For the forward part, NoiseZO achieves about 15-fold advantages compared to SOTA ASICs. For the entire training, NoiseZO achieves over 21-fold advantages compared with the ASICs.

V. CONCLUSION

We propose NoiseZO, a novel forward-only training method for RRAM-based networks. We transform RRAM's inherent write noise from a limitation into an asset, leveraging it during forward computations to drive ZO optimization. Our approach also avoids the significant hardware overhead associated with backward gradient computation in BP methods. Furthermore, we introduce fine-grained NoiseZO as an enhancement to NoiseZO, which effectively accelerates convergence and improves accuracy. Remarkably, compared to fully CMOS-based training, our framework shows an impressive 21 times lower energy consumption. While currently validated on small-scale tasks, success of ZO methods in LLMs indicates our broader potential and motivates exploration with more complex tasks.

VI. ACKNOWLEDGMENT

This work was supported in part by the Theme-based Research Scheme (TRS) project T45-701/22-R, National Natural Science Foundation of China (62404187) and the General Research Fund (GRF) Project 17203224, of the Research Grants Council (RGC), Hong Kong SAR.

REFERENCES

[1] H. Mark, "1.1 computing's energy problem (and what we can do about it)," in *ISSCC*, pp. 9–13, 2014.

[2] C. E. Leiserson, N. C. Thompson, J. S. Emer, B. C. Kuszmaul, B. W. Lampson, D. Sanchez, and T. B. Schardl, "There's plenty of room at the top: What will drive computer performance after moore's law?," *Science*, vol. 368, no. 6495, p. eaam9744, 2020.

[3] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature electronics*, vol. 1, no. 6, pp. 333–343, 2018.

[4] M. Prezioso, F. Merrikh-Bayat, B. D. Hoskins, G. C. Adam, K. K. Likharev, and D. B. Strukov, "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.

[5] P. Yao, H. Wu, B. Gao, S. B. Eryilmaz, X. Huang, W. Zhang, Q. Zhang, N. Deng, L. Shi, H.-S. P. Wong, *et al.*, "Face classification using electronic synapses," *Nature communications*, vol. 8, no. 1, p. 15199, 2017.

[6] C. Li, D. Belkin, Y. Li, P. Yan, M. Hu, N. Ge, H. Jiang, E. Montgomery, P. Lin, Z. Wang, *et al.*, "Efficient and self-adaptive in-situ learning in multilayer memristor neural networks," *Nature communications*, vol. 9, no. 1, p. 2385, 2018.

[7] P. Yao, H. Wu, B. Gao, J. Tang, Q. Zhang, W. Zhang, J. J. Yang, and H. Qian, "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, 2020.

[8] C. Li, Z. Wang, M. Rao, D. Belkin, W. Song, H. Jiang, P. Yan, Y. Li, P. Lin, M. Hu, *et al.*, "Long short-term memory networks in memristor crossbar arrays," *Nature Machine Intelligence*, vol. 1, no. 1, pp. 49–57, 2019.

[9] S. Yu, W. Shim, X. Peng, and Y. Luo, "Rram for compute-in-memory: From inference to training," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, no. 7, pp. 2753–2765, 2021.

[10] Q. Zhang, H. Wu, P. Yao, W. Zhang, B. Gao, N. Deng, and H. Qian, "Sign backpropagation: An on-chip learning algorithm for analog rram neuromorphic computing systems," *Neural Networks*, vol. 108, pp. 217–223, 2018.

[11] B. Gao, Y. Zhou, Q. Zhang, S. Zhang, P. Yao, Y. Xi, Q. Liu, M. Zhao, W. Zhang, Z. Liu, *et al.*, "Memristor-based analogue computing for brain-inspired sound localization with in situ training," *Nature communications*, vol. 13, no. 1, p. 2026, 2022.

[12] W. Zhang, P. Yao, B. Gao, Q. Liu, D. Wu, Q. Zhang, Y. Li, Q. Qin, J. Li, Z. Zhu, *et al.*, "Edge learning using a fully integrated neuro-inspired memristor chip," *Science*, vol. 381, no. 6663, pp. 1205–1211, 2023.

[13] S.-i. Yi, J. D. Kendall, R. S. Williams, and S. Kumar, "Activity-difference training of deep neural networks using memristor crossbars," *Nature Electronics*, vol. 6, no. 1, pp. 45–51, 2023.

[14] J. C. Spall, "Multivariate stochastic approximation using a simultaneous perturbation gradient approximation," *IEEE transactions on automatic control*, vol. 37, no. 3, pp. 332–341, 1992.

[15] S. Malladi, T. Gao, E. Nichani, A. Damian, J. D. Lee, D. Chen, and S. Arora, "Fine-tuning language models with just forward passes," *Advances in Neural Information Processing Systems*, vol. 36, pp. 53038–53075, 2023.

[16] J. Gu, C. Feng, Z. Zhao, Z. Ying, R. T. Chen, and D. Z. Pan, "Efficient on-chip learning for optical neural networks through power-aware sparse zeroth-order optimization," in *AAAI*, no. 9, pp. 7583–7591, 2021.

[17] J. Gu, Z. Zhao, C. Feng, W. Li, R. T. Chen, and D. Z. Pan, "Flops: Efficient on-chip learning for optical neural networks through stochastic zeroth-order optimization," in *DAC*, pp. 1–6, 2020.

[18] H. Zhao, Z. Liu, J. Tang, B. Gao, Y. Zhou, P. Yao, Y. Xi, H. Qian, and H. Wu, "Implementation of discrete fourier transform using rram arrays with quasi-analog mapping for high-fidelity medical image reconstruction," in *IEDM*, pp. 12–4, 2021.

[19] S. Liu, P.-Y. Chen, B. Kailkhura, G. Zhang, A. O. Hero III, and P. K. Varshney, "A primer on zeroth-order optimization in signal processing and machine learning: Principals, recent advances, and applications," *IEEE Signal Processing Magazine*, vol. 37, no. 5, pp. 43–54, 2020.

[20] Y. Zhang, Y. Yao, J. Jia, J. Yi, M. Hong, S. Chang, and S. Liu, "How to robustify black-box ml models? a zeroth-order optimization perspective," *arXiv preprint arXiv:2203.14195*, 2022.

[21] S. Ghadimi and G. Lan, "Stochastic first-and zeroth-order methods for nonconvex stochastic programming," *SIAM journal on optimization*, vol. 23, no. 4, pp. 2341–2368, 2013.

[22] S. Liu, P.-Y. Chen, X. Chen, and M. Hong, "signsgd via zeroth-order oracle," in *ICLR*, 2019.

[23] S. Ghadimi, G. Lan, and H. Zhang, "Mini-batch stochastic approximation methods for nonconvex stochastic composite optimization," *Mathematical Programming*, vol. 155, no. 1, pp. 267–305, 2016.

[24] X. Chen, S. Liu, K. Xu, X. Li, X. Lin, M. Hong, and D. Cox, "Zo-adamm: Zeroth-order adaptive momentum method for black-box optimization," in *NeurIPS*, p. 7204–7215, 2019.

[25] S. Liu, J. Chen, P.-Y. Chen, and A. Hero, "Zeroth-order online alternating direction method of multipliers: Convergence analysis and applications," in *AISTATS*, pp. 288–297, 2018.

[26] Y. Nesterov and V. Spokoiny, "Random gradient-free minimization of convex functions," *Foundations of Computational Mathematics*, vol. 17, no. 2, pp. 527–566, 2017.

[27] A. S. Berahas, L. Cao, K. Choromanski, and K. Scheinberg, "A theoretical and empirical comparison of gradient approximations in derivative-free optimization," *Foundations of Computational Mathematics*, vol. 22, no. 2, pp. 507–560, 2022.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[29] D. Deterding, *Speaker Normalisation for Automatic Speech Recognition*. PhD thesis, University of Cambridge, 1989.

[30] Y. Zhang, P. Li, J. Hong, J. Li, Y. Zhang, W. Zheng, P.-Y. Chen, J. D. Lee, W. Yin, M. Hong, *et al.*, "Revisiting zeroth-order optimization for memory-efficient llm fine-tuning: A benchmark," *arXiv preprint arXiv:2402.11592*, 2024.

[31] A. Chen, Y. Zhang, J. Jia, J. Diffenderfer, J. Liu, K. Parasyris, Y. Zhang, Z. Zhang, B. Kailkhura, and S. Liu, "Deepzero: Scaling up zeroth-order optimization for deep model training," *arXiv preprint arXiv:2310.02025*, 2023.

[32] P. Zhou, D.-U. Choi, W. D. Lu, S.-M. Kang, and J. K. Eshraghian, "Gradient-based neuromorphic learning on dynamical rram arrays," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 12, no. 4, pp. 888–897, 2022.

[33] Y. Wang, Y. Qin, L. Liu, S. Wei, and S. Yin, "Swpu: A 126.04 tflops/w edge-device sparse dnn training processor with dynamic sub-structured weight pruning," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 10, pp. 4014–4027, 2022.

[34] S. Kang, D. Han, J. Lee, D. Im, S. Kim, S. Kim, and H.-J. Yoo, "7.4 ganpu: A 135tflops/w multi-dnn training processor for gans with speculative dual-sparsity exploitation," in *ISSCC*, pp. 140–142, 2020.

[35] S. Kim, J. Lee, S. Kang, J. Lee, and H.-J. Yoo, "A 146.52 tops/w deep-neural-network learning processor with stochastic coarse-fine pruning and adaptive input/output/weight skipping," in *VLSI*, pp. 1–2, 2020.

[36] J. Lee, J. Lee, D. Han, J. Lee, G. Park, and H.-J. Yoo, "7.7 lnpu: A 25.3 tflops/w sparse deep-neural-network learning processor with fine-grained mixed precision of fp8-fp16," in *ISSCC*, pp. 142–144, 2019.

[37] W. Zhang, X. Peng, H. Wu, B. Gao, H. He, Y. Zhang, S. Yu, and H. Qian, "Design guidelines of rram based neural-processing-unit: A joint device-circuit-algorithm analysis," in *DAC*, pp. 1–6, 2019.

[38] T. Andrulis, J. S. Emer, and V. Sze, "Raella: Reforming the arithmetic for efficient, low-resolution, and low-loss analog pim: No retraining required!," in *ISCA*, pp. 1–16, 2023.

[39] X. Peng, S. Huang, H. Jiang, A. Lu, and S. Yu, "Dnn+ neurosim v2. 0: An end-to-end benchmarking framework for compute-in-memory accelerators for on-chip training," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 11, pp. 2306–2319, 2020.

Live Region Mutation Testing for Commercial Cyber-Physical System Development Tool Chain

Lehuan Zhang[a], Shikai Guo[b,*], Zixuan Wang[b], Xiaoyu Wang[b], Xiaochen Li[a], He Jiang[a]
[a]School of Software, Dalian University of Technology, Dalian, China
[b]School of Information Science and Technology, Dalian Maritime University, Dalian, China
{lehuanzhang}@mail.dlut.edu.cn,{shikai.guo, 1120211459}@mail.dlut.edu.cn, {xiaochen.li,jianghe}@mail.dlut.edu.cn

***Abstract*—MathWorks Simulink, a commercial CPS development tool chain, is widely used as an industry standard for designing and analyzing system behavior and generating embedded code for deployment. However, bugs in Simulink can cause unexpected behaviors during model compilation, making their elimination critical. Existing methods face two key challenges: generating equivalent models with varied data flows (data flow equivalence) and creating diverse block types to comprehensively test the compiler (mutation diversity). To address these, we propose LION, a differential testing approach. LION ensures data flow equivalence by inserting "store-revert" block pairs between existing blocks and tackles mutation diversity by employing Markov Chain Monte Carlo (MCMC) sampling to generate diverse new blocks. Differential testing is then used to identify bugs. Experiments show LION outperforms state-of-the-art approaches like SLforge, SLEMI, and COMBAT, detecting 6-10 additional compiler bugs in two weeks. Over two months, LION uncovered and reported 16 valid bugs in the widely used stable version of Simulink.**

***Index Terms*—Cyber-Physical System, Simulink, Equivalence Modulo Input, Differential Testing, Compiler Testing.**

I. Introduction

As a de facto industrial standard [1], the commercial CPS development tool chain MathWorks Simulink has been widely used to design and analyze Simulink models (i.e., *block-diagrams* with connections) for simulating the behavior of the Simulink, in many safety-critical applications such as automotive [2] and aerospace [3]. Developers employ Simulink models to model the behavior of the target system. Each block represents a predefined function to manipulate the data passed through the Simulink model. When the simulation fits the expectation of developers, Simulink automatically generates embedded code that gets deployed in target applications. However, Simulink is a complex software system that may contain bugs [4, 5, 6]. These bugs cause compilation errors of Simulink models or inject subtle unexpected behaviors into the applications that use the embedded code [7].

The Simulink compiler is the core component enabling developers to share simulations as standalone executables. To identify flaws and improve stability, research primarily uses generation-based and mutation-based methods for testing. Generation-based approaches, like SLforge [5], randomly generate valid Simulink models, expecting identical results across different Simulink settings. If results vary, a bug may be present. Mutation-based methods, such as SLEMI [1] and COMBAT [8], systematically modify seed models while maintaining semantics, using Equivalent Model Input (EMI) testing. If outputs differ from the seed model, a compilation error is likely. EMI, a form of differential testing, has been shown to identify semantic and logic bugs that generation-based approaches may miss [1].

However, existing EMI approaches for Simulink compiler testing face two main challenges: *the data flow equivalence challenge* and *the mutation diversity challenge*. The first challenge arises because these methods often mutate blocks in "zombie regions"—areas of a Simulink model that are not executed or processed under a given input. Mutations in zombie regions may be ignored by the Simulink compiler, as they do not influence data flow. To identify more bugs, it is critical to mutate blocks in "live regions"—areas that can affect the data flow and be processed by the compiler, thus increasing the likelihood of triggering bugs. The second challenge is that most mutation strategies focus on modifying or deleting existing blocks, rarely introducing new block types. Since the Simulink compiler uses different simulation modes for optimizing various block types, generating equivalent model variants with new blocks is a challenge.

While live region mutation has been studied for testing traditional programming language compilers (e.g., LLVM [9], JVM [10]), these methods cannot be directly applied to Simulink compiler testing. Triggering more Simulink compiler bugs by altering active regions remains challenging due to several unique features of the Simulink language. Unlike traditional languages with formal specifications (e.g., Pascal, C [5]), Simulink lacks comprehensive language specifications. Additionally, Simulink's handling of "executed" (zombie regions) and "non-executed" (active regions) code differs from traditional languages; for instance, an *If* block outputs a default value even for unexecuted branches. Simulink also introduces complexities like explicit sampling times, where outputs vary at different intervals, and automatic block data type inference, where minor model changes can alter data types [1]. Existing Simulink compiler testing methods [1, 8] largely focus on zombie region mutations, with limited evidence on the effectiveness of live region mutation.

To test Simulink compiler effectively, we propose LION, a **LI**ve region bl**O**ck mutatio**N** approach, which considers the features of Simulink languages and addresses the aforementioned two challenges for Simulink compiler testing. LION consists of three components: equivalent data flow generation, model mutation, and differential testing. Given a seed Simulink model, the equivalent data flow generation component inserts a type of specific block pairs named "*store-revert*" at arbitrary points between two existing blocks. Each *store* block records the data passed through it. These data are reverted at the exit of the *revert* block to maintain the value equivalence. Since such modifications could change the automatically inferred block datatypes and break the equivalence modulo input property, we use *Datatype Conversion* blocks to maintain the datatype equivalence. By inserting *store-revert* block pairs, LION generates equivalent data flows with the seed Simulink model thus addressing the data flow equivalence challenge. The model mutation component learns the relationship of blocks from a repository of valid Simulink models. It uses MCMC sampling strategy to generate a sequence of new blocks between each *store-revert* block pair based on the learned relationship. Comparing to modifying/deleting existing blocks or reusing

* Corresponding author (Shikai Guo)

existing block sequences, LION generates new block sequences by sampling from a large number of valid and diverse Simulink models without knowing the language specifications, thus addressing the mutation diversity challenge. Finally, LION runs the seed Simulink model and its variants with the same Simulink settings to find bugs caused by output inconsistency on all sample time points.

We evaluate LION on multiple Simulink versions, including an old version Simulink R2018a and the stable widely-used version Simulink R2021b. LION finds 11 bugs on the two Simulink versions in 200 CPU hours, which significantly outperforms state-of-the-art generation-based and mutation-based approaches in Simulink compiler testing. LION can find 6-10 more bugs than the baselines on two versions of Simulink in two weeks. LION can continuously find new bugs in the Simulink, that has been fully tested before release. To run LION for two months, we have reported 16 confirmed bugs for a stable widely-used version Simulink R2021b, of which 12 bugs are new bugs. A further analysis shows that both equivalent data flow generation component and model mutation component contribute to the effectiveness of LION, which help LION efficiently generate a large number of Simulink model variants with a mutation success rate around 0.45. These results indicate LION is effective to improve the reliability of systems, that use Simulink to design embedded systems.

In summary, the main contributions of this work are:

- We propose LION, which uses equivalent data flow generation and model mutation to tackle the challenges of Simulink compiler testing.
- We conduct extensive experiments, which shows that the bug-finding capability of LION significantly outperforms state-of-the-art approaches.
- We release LION as an open-source tool for Simulink compiler testing [11].

II. Related Work

Recent research on Simulink has mainly focused on simulation acceleration [12], test coverage enhancement [13], and code generation optimization [14], while studies on CPS toolchain and Simulink compiler testing remain limited. Bugs in the Simulink compiler are often misidentified as model issues, leading to incorrect diagnoses and disrupting the development process. Therefore, improving test coverage and accuracy in Simulink compiler testing is essential for ensuring model correctness and boosting development efficiency. For CPS tool chain testing, prior works have primarily focused on using Simulink models for testing compilers via differential and EMI techniques. Tools such as CyFuzz [4] and SLforge [5] generate Simulink models to find faults, with SLforge being state-of-the-art in bug-finding capability. Mutation-based approaches like SLEMI and COMBAT [8] modify existing models to test compilers, though their bug-finding ability is constrained by their focus on specific regions. In contrast, our proposed approach, LION, employs MCMC sampling to generate diverse and complex model variants, targeting both live and inactive regions. This enables deeper testing of Simulink compilers and addresses limitations of prior methods such as COMBAT, which focuses solely on zombie regions. Experimental results demonstrate the superior performance of LION in detecting compiler errors.

In compiler testing for traditional programming languages (e.g., C, C++, Java), differential testing remains a cornerstone. Techniques like Randomized Differential Testing (RDT), Different Optimization Levels (DOL), and EMI approaches have been extensively used. Tools like Csmith [15] generates random programs to uncover compiler bugs, while other approaches mutate programs in dead and live regions to maintain equivalence modulo inputs. Additionally, recent works explore optimization sequence bugs and compiler-warning bugs. Unlike these efforts, which focus on traditional languages, our study emphasizes Simulink compiler testing and introduces LION as a novel live-region mutation strategy, outperforming existing Simulink-specific approaches in bug detection.

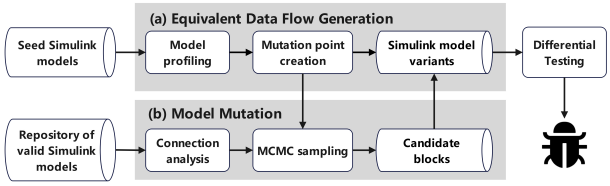


Fig. 1: The framework of LION.

III. Framework Of LION

In this section, we propose LION, a LIve region blOck mutatioN approach, to address the two challenges for Simulink compiler testing. The framework of LION is illustrated in Fig. 1.

LION consists of three components: equivalent data flow generation, model mutation, and differential testing. The equivalent data flow generation component profiles the seed Simulink model and inserts *store-revert* block pairs at arbitrary points between two blocks. The *store* block records the data passed through it, while the *revert* block reverts the recorded data at its output ports to address the data flow equivalence challenge. Between each block pair, the model mutation component uses the MCMC sampling strategy to generate a sequence of new blocks by sampling existing blocks and connections from a repository of valid and diverse Simulink models, which addresses the mutation diversity challenge. Finally, differential testing is used to find Simulink compiler bugs.

A. *Equivalent Data Flow Generation*

Within this component, LION creates mutation points in a seed Simulink model. LION aims to alter the data flow of the seed Simulink model at each mutation point, while keeping the data flows equivalent. The equivalent data flow generation component has two steps, including model profiling and mutation point creation. We explain this phase with the Simulink model example in Fig. 2. Blocks in white are the original blocks in the seed Simulink model.

***Model Profiling**:* This step executes the seed Simulink model and profiles the model by analyzing variables, their values, and their datatypes passed through input ports and output ports of each block. During the execution of a Simulink model, the Simulink compiler can identify both zombie regions and active regions. LION uses the Signal Range Coverage Tool [16] in Simulink for model profiling. For example, in Fig. 2, LION can capture the output of the *Abs* block (which is fed into the *Add* block) at each sample time point during the Simulink model execution.

***Mutation Point Creation**:* After model profiling, LION randomly selects k points in the seed Simulink model as mutation points ($k = 10$ in this study). For example, in Fig. 2 LION selects a mutation point between *Abs* and *Add* blocks. LION can select mutation points in both live regions and zombie regions. However, in this study, we only add mutation points in live regions, because mutations in zombie regions might be eliminated if they do not affect the data flow of a Simulink model [17].

At each mutation point, LION inserts a pair of *store-revert* blocks to alter the data flow of the seed Simulink model. Specifically, LION adds two custom blocks at the mutation point, i.e., the *MATLAB Function Store* block and the *MATLAB Function Revert* block. The *MATLAB Function Store* block records variables and their values from

the output port of the *Abs* block. The *MATLAB Function Store* block has two output ports y and z. The port z connects with the input port v of the *MATLAB Function Revert* block, which is used to pass the recorded variables and values to the *Add* block through the output port w. Since all variables and their values from the *Abs* block have been transferred to the *Add* block, the data flow equivalence is kept.

However, Simulink model has the process of datatype inference, where a small modification may significantly change the automatically inferred datatypes of blocks and break the data flow equivalence, especially when live regions are modified. To maintain the equivalence modulo input property, we follow the existing study [1] to add a *Datatype Conversion* block before each block in the Simulink model. These blocks automatically specify the datatypes of blocks in Simulink models based on the datatypes parsed during profiling.

Between a *store-revert* block pair, LION creates mutation space for generating a sequence of blocks (e.g., the *Divide* block and the *First Order Hold* block in Fig. 2). These new blocks take variables and values recorded in the *MATLAB Function Store* block as input (delivered through the output port y), perform operations on the recorded variables, and pass results to the input port u of the *MATLAB Function Revert* block. These results are saved in the *MATLAB Function Revert* block locally and will not affect subsequent blocks.

Compared to existing Simulink compiler testing approaches that mainly mutate in zombie regions, LION significantly increases the space for Simulink model mutation by inserting *store-revert* block pairs at arbitrary points in the seed Simulink model. This strategy adds side effects on the seed Simulink model by changing its data flow. However, these side effects are reverted at the exit of the *MATLAB Function Revert* block to maintain the equivalence modulo input property. Since these *store-revert* block pairs are in live regions and alter the data flow, Simulink compiler has to optimize blocks in the new data flows. It helps more thoroughly test Simulink compiler.

B. *Model Mutation*

In this phase, LION automatically generates block sequences between each *store-revert* block pair using an MCMC sampling strategy for model mutation. MCMC approximates a target distribution by creating a Markov chain, where each state depends on the previous one, and the chain's long-run distribution mirrors the target. This enables statistical inference and is widely used to enhance mutation-based testing [18]. Unlike previous studies that reuse existing code snippets, we use MCMC to generate block sequences in Simulink, a block-diagram-based language with no formal specifications. The key idea is to build a Markov chain to capture the relationships between block connections in valid Simulink models, allowing LION to sample diverse and valid block sequences from a repository.

The model mutation process is outlined in Algorithm 1. LION takes a seed Simulink model M, a repository R_M of valid models, *store-revert* block pairs $\langle blockS, blockR\rangle$, and a parameter N specifying the number of blocks per pair. It outputs a mutated Simulink model M'. LION first analyzes block connections in M using the repository R_M (line 1). The repository provides a variety of candidate blocks for sampling, and since the connections between blocks are datatype-compatible, this enhances mutation success. We use SLforge to generate the repository of valid models. LION builds a block database $blockDB$ from R_M and computes two probabilities: the block density distribution $densePr$ and the transition probability $transPr$. $densePr$ reflects the frequency of each block in R_M, while $transPr$ represents the one-step transition probability from block b to block $\hat{b}$, calculated as the ratio of connections from b to $\hat{b}$ over the total connections from b.

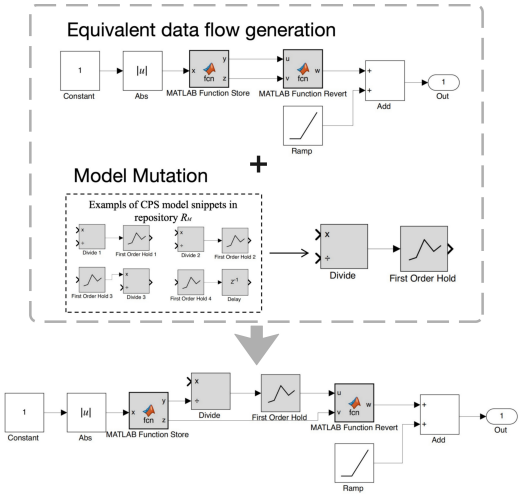


Fig. 2: An example of Simulink model variant generated by LION and the repository.

Algorithm 1 Model Mutation

Require: Seed Simulink model M
Valid Simulink model repository R_M
Store-revert block pairs $\langle blockS, blockR\rangle$
Number of blocks N to generate in each block pair
Ensure: Simulink model variant M'
1: *densePr*, *transPr*, *blockDB* $\leftarrow$ *Parse*(R_M)
2: **for** $\langle blockS_i, blockR_i\rangle \in \langle blockS, blockR\rangle$ **do**
3: $headBlock \leftarrow SelectBlock(blockDB)$
4: $M' \leftarrow AddLine(M, blockS_i, headBlock)$
5: $curBlock = headBlock$
6: **for** $i = 0 \rightarrow N-1$ **do**
7: $candiBlocks \leftarrow ConnectedBlocks(curBlock, blockDB)$
8: $accRates \leftarrow CalcAcc(densePr, transPr, candiBlocks)$
9: $nextBlock \leftarrow ChooseBlock(candiBlocks, accRates)$
10: $M' \leftarrow AddLine(M', curBlock, nextBlock)$
11: $curBlock = nextBlock$
12: **end**
13: $M' \leftarrow AddLine(M', curBlock, blockR_i)$
14: **end**
15: $M' \leftarrow DataTypeInference(M')$
16: **return** (M')

After analyzing R_M, LION starts to generate new blocks between each *store-revert* block pair. Initially, LION randomly selects a block in $blockDB$ as the head block to increase the mutation diversity (line 3). The head block is connected with the *store* block (line 4). Taking the head block as the current block, LION collects all blocks in $blockDB$ that the current block connects with (line 7). LION calculates an acceptance rate $accRate$, representing the probability to accept $\hat{b}$ as the next block when the current block is b (line 8).

$$accRate(b \rightarrow \hat{b}) = \min(1, \frac{\theta(\hat{b})q(b|\hat{b})}{\theta(b)q(\hat{b}|b)}), \tag{1}$$

where $\theta(b)$ and $\theta(\hat{b})$ are the probability density distribution of blocks b and $\hat{b}$, respectively, and $q(b|\hat{b})$ and $q(\hat{b}|b)$ represent their transition probability. The intuition of this formula is, blocks that are frequently used and connected with block b should have a higher chance to be accepted as the next block. The overall significance of the division is to assess the relative likelihood of accepting the candidate block $\hat{b}$, thereby determining whether to accept it

and ensuring that the Markov chain ultimately converges to the target distribution. LION randomly selects the next block taking into consideration of the acceptance rate of each candidate block (line 9), i.e., blocks with higher acceptance rates have a higher chance to be selected. In an extreme case, there could be no block to select (e.g., the current block is a *sink* block). LION will randomly select a block in $blockDB$ as the next block. After selection, LION takes the selected block as the current block for the next round of sampling (line 11). The sampling in each *store-revert* block pair stops when the number of selected blocks reaches a certain number N. The output port of the last block in the sequence is connected with the *revert* block (line 13).

After generating a Simulink model variant, LION propagates datatype information from *store* blocks to the newly generated blocks along the connections using forward propagation (line 15). If the datatype is not supported by a block, we follow SLEMI to add a *Datatype Conversion* block before it [1]. The algorithm ends by returning a Simulink model variant of the seed Simulink model. In this stage, we generate Simulink model variants with different types of blocks to exercise the Simulink compiler thoroughly, as it may use various optimization strategies to optimize these variants.

After the mutation process, LION generates a large number of mutated versions of the seed Simulink models. Next, LION filters these mutants. When mutated Simulink models fail to pass Simulink compilation, LION discards such mutants and retains only those that successfully compile for subsequent differential testing.

C. *Differential Testing*

LION employs EMI (a form of differential testing) to evaluate the Simulink compiler. As illustrated in Fig. 3, given the original inputs of the seed model, equivalent variant models are expected to produce identical outputs. To this end, LION executes the seed model and its variants under the same Simulink configuration (e.g., *Normal* mode or *Accelerator* mode) and compares their outputs. We use the Simulation Data Inspector of Simulink to show this process graphically and intuitively.

Specifically, for a seed Simulink model M and each of its variants M', let $t \in [t_0, t_n]$ denotes a sampling time point within the testing time interval. The output of model M at sampling time t_i under *Normal* mode is denoted as $O^N_M(t_i)$. The output of model M at sampling time t_i under *Accelerator* mode is denoted as $O^A_M(t_i)$. Similarly, the outputs of M' at sampling time t_i in *Normal* and *Accelerator* mode are denoted as $O^N_{M'}(t_i)$ and $O^A_{M'}(t_i)$. If there exists a sampling time t_i such that:

$$O^N_M(t_i) \neq O^N_{M'}(t_i) \quad \text{or} \quad O^A_M(t_i) \neq O^A_{M'}(t_i),$$

then the variant model M' is marked to have triggered a bug. The above process is repeated for all variant Simulink models generated from each seed Simulink model, ultimately identifying all variants marked as triggering bugs.

IV. Evaluation

In this section, we evaluate our approach LION for Simulink compiler testing. First, we compare LION with state-of-the-art approaches to assess their overall effectiveness in finding Simulink compiler bugs. Second, we evaluate how the use of the equivalent data flow generation component and the model mutation component affect the effectiveness of LION. Last, we run LION on the stable widely-used Simulink version to analyze its bug-finding capability in practice. Specifically, we aim to answer the following research questions:

RQ1 *How is the bug-finding capability of LION compared against state-of-the-art approaches?*

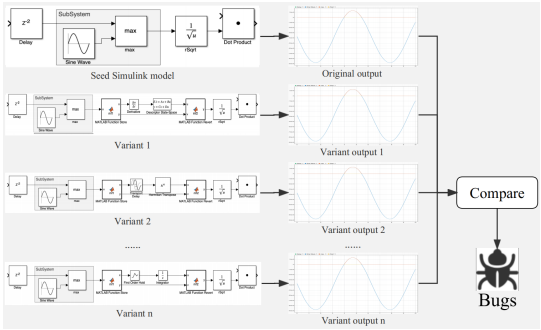


Fig. 3: Schematic diagram of differential testing.

RQ2 *What is the impact of using the equivalent data flow generation component and the model mutation component on the effectiveness of LION?*

RQ3 *Can LION steadily find new bugs in the stable widely-used Simulink version?*

A. *Seed Simulink Models And Testing Strategy*

LION uses seed Simulink models to generate model variants. Building on existing research, LION utilizes both real-world models and SLforge-generated models as seeds. Real-world models are sourced from a corpus of 1,000 publicly available CPS projects [19]. LION filters out models that could not run in our environment, retaining 357 Simulink models. To supplement this limited number, SLforge, a Simulink model generator, is employed to create additional seeds, following prior methods [1]. These models contain 1–5 subsystems with 30–100 blocks per subsystem, resulting in seed models with 100–3,000 blocks. SLforge's generation time increases significantly with block count due to connection corrections, potentially dominating the testing process. Moreover, models exceeding this scale rarely trigger compiler bugs, as shown in preliminary evaluations. This range is chosen to balance efficiency and effectiveness in highlighting LION's ability to generate bug-triggering variants.

In the testing strategy, LION generates one seed Simulink model at a time and feed it to LION to create model variants. These seed models serve as a repository for model mutation, with each generated seed added to the repository. The repository is continuously updated with new valid models. Once variants are generated, LION runs both the seed and its variants with the same Simulink settings. First, LION tests all models in *Normal* simulation mode to find bugs, then in *Accelerator* mode. To isolate equivalence modulo input's impact from differential testing, LION does not compare the outputs between *Normal* and *Accelerator* simulation modes (i.e., the strategy of plain differential testing). When all Simulink models have been executed, LION generates the next seed Simulink model. Testing continues until the predefined period ends.

B. *Evaluation Criterion*

When a bug is detected, LION compares its failed assertion and back-trace with those of previously identified bugs by LION. If they match, LION classifies it as a duplicate. However, some miscompilation bugs do not have the information of failed assertion and back-trace. De-duplicating such bugs is still an open research question.

Initially, these bugs are considered unique and are provided to Simulink developers for further analysis. For each unique bug, LION reduces the corresponding Simulink model by iteratively removing blocks until it identify the minimal model that triggers the bug [18, 20]. Then, we create a bug report for each bug to

explain the symptom of the bug and submit it to the MathWorks' bug reporting platform [6]. We mainly get two types of feedback from the MathWorks support team for a bug report, i.e., *new* and *known*, where "new" means this is a new bug in Simulink and "known" means the bug is duplicate with existing bugs in their bug repository. Since the complete list of bug reports is neither available nor traceable in the MathWorks' bug reporting platform, we publish our reported Simulink models for reproducing our identified bugs [11].

C. Implementation And Settings

We implement LION as a Matlab program. Given a seed Simulink model, we configure LION to generate five Simulink model variants, which is the same setting as the existing study SLEMI [1] and COMBAT [8]. In addition, LION has two unique parameters, i.e., the number of mutation points (i.e., *store-revert* block pairs) to insert k (as shown in Section III-A) and the number of blocks to generate per mutation point N (as shown in Algorithm 1). During mutation, we configure LION to insert 10 mutation points in the seed Simulink model. At each mutation point, LION randomly generates 5 blocks. These parameters were selected based on experimental comparisons.

We performed our experiments with a computer running Windows 10 64-bit system with a 2.10 GHz Intel Core i9 processor and 120 GB of memory. The implementation of LION and experimental results are available at GitHub [11].

D. Comparison With Baselines (RQ1)

To assess the effectiveness of LION, we compare the bug-finding capability of LION with the state-of-the-art approaches:

1) SLforge [5] generates random Simulink models with controllable size and complexity, though it supports only a subset of Simulink specifications due to incomplete documentation.
2) SLEMI [1] applies equivalence modulo input (EMI) techniques to mutate seed Simulink models and generate variants for compiler testing.
3) COMBAT [8] extends SLEMI by modifying zombie regions and adding blocks to enhance variant generation, improving bug detection through EMI testing.

Since SLforge is a Simulink model generator that emphasizes plain differential testing. For each generated Simulink model by SLforge, we run it in both *Normal* and *Accelerator* simulation modes to identify the output inconsistency between the two modes. For SLEMI, COMBAT, and LION, we use SLforge-generated seed models and run each seed and its variants in a single simulation mode (*Normal* or *Accelerator*) to isolate the impact of equivalence modulo input.

We run each approach with an evaluation period of 200 CPU hours [1, 8], including the time to generate Simulink models, mutation (if necessary), and differential testing. We choose two Simulink versions for testing, i.e., Simulink R2018a and Simulink R2021b. The reason is, Simulink R2018a has been used in the related work [1, 8], while Simulink R2021b is the stable widely-used version during this study. We reproduce SLforge, SLEMI and COMBAT with the source code provided in their studies and use the default configurations in the source code or specified in the corresponding papers. We have cross-reviewed the implementation and resolve deployment issues.

Table I presents the bugs found by different approaches on the two Simulink versions. As shown in Table I, the baselines only find 6 or fewer bugs. Specifically, SLforge, SLEMI, and COMBAT found 1, 3, and 6, respectively. In contrast, LION finds 11 bugs on the two Simulink versions in the testing period, that is at least twice what baselines find. SLforge only find 1 bug in 200 CPU hours. SLforge detects a bug in Simulink R2018a where a model worked in normal mode but failed in accelerated mode, a bug also identified by LION and SLEMI due to LION's use of SLforge-generated seed models. While LION and COMBAT identified overlapping bugs, COMBAT's effectiveness was limited to Simulink R2018a, with many bugs fixed in later versions. COMBAT's reliance on zombie regions further hindered its performance in newer Simulink versions.

TABLE I: Bugs Found by LION and Baselines.

	Simulink R2018a	Proper Simulink version(Simulink R2021b)	Total
SLforge	1	0	1
SLEMI	2	1	3
COMBAT	4	2	6
LION	3	8	**11**

TABLE II: Impact of different components in LION.

	LION$_{dis_dflow}$		LION$_{dis_mcmc}$		LION	
	New	Known	New	Known	New	Known
Bugs	0	2	0	5	2	6

LION, on the other hand, was able to uncover additional potential bugs in Simulink by overcoming challenges related to data flow equivalence and mutation diversity. Notably, LION found more bugs in Simulink R2021b, as developers primarily fix bugs in newer versions. If a bug in Simulink R2018a does not impact relatively newer versions, it is rarely confirmed by the developers. In particular, LION finds more bugs by generating new mutation space in live region with different control/data flow characteristics.

E. Impact Of Different Components (RQ2)

LION has two important components: (1) the equivalent data flow generation component, which generates different but equivalent data flows for a seed Simulink model (Section III-A), and (2) the model mutation component, which uses the MCMC sampling strategy to generate diverse blocks (Section III-B). To answer RQ2, we assess the impact of these two components on the effectiveness of LION.

We create two variants of LION, denoted as LION$_{dis_dflow}$ and LION$_{dis_mcmc}$. LION$_{dis_dflow}$ disables the impact of the equivalent data flow generation component. Instead of creating *store-revert* block pairs, we use SLEMI and COMBAT to randomly create a new signal path terminated by a *sink* block in live regions (i.e., the fourth mutation strategy in SLEMI [1]). In the new path, blocks are generated by the MCMC sampling strategy. LION$_{dis_dflow}$ investigates whether the data flow modification strategy in LION is more effective than the existing strategy. LION$_{dis_mcmc}$ disables the impact of the model mutation component by replacing the MCMC sampling strategy with a random sampling strategy, which randomly selects blocks from a repository and adds them into *store-revert* block pairs. We run LION and the additional variants for 200 CPU hours, and measure their bug-finding capability. Since developers mainly work on the stable Simulink version, we test Simulink R2021b in the following RQs.

As shown in Table II, both components impact the effectiveness of LION. The equivalent data flow generation component helps LION thoroughly test Simulink compiler by altering the data flow that has to be optimized in live regions. The model mutation strategy generates Simulink model variants with different types of blocks, which allows Simulink compiler to use various optimization strategies to optimize them. When we disable the impact of a component, the bug-finding capability of LION drops. LION$_{dis_dflow}$ and LION$_{dis_mcmc}$ find 2 and 5 bugs on Simulink R2021b, respectively. LION$_{dis_dflow}$ finds fewer bugs than LION$_{dis_mcmc}$. It means comparing with existing strategy in SLEMI, the *store-revert* block pairs inserted in the seed Simulink model significantly improve the effectiveness of LION.

F. *Effectiveness In Finding New Bugs (RQ3)*

LION is tested on Simulink R2021b, a stable and widely-used version, for two months. Developers prioritize fixing bugs in such versions, making the bugs discovered by LION more relevant and often new. Using the strategy in Section IV-A, all bugs identified by LION are submitted to MathWorks' bug reporting platform for verification ((as shown in Section IV-B). Due to the time required for a full run, comparisons with SLforge, SLEMI, and COMBAT are not conducted.

We manually classify the submitted bugs from three dimensions:

1) The feedback from the MathWorks support team (Fb): Indicating whether the bug was new (N) or known (K).
2) The Discovery Point (Dp): The stage of detection, either during compilation (C) or runtime (R).
3) The Symptom Type (St): Categorized as compiler prompt blur (B), inconsistent compilation prompts (I), time vector errors (T), or data inconsistency errors (D), based on back-tracing and MathWorks' feedback.

As shown in Table III, LION found 16 confirmed bugs in Simulink R2021b in two months, of which 12 bugs are new and 4 bugs are known. Each bug has a unique ID from MathWorks. Simulink models triggering these bugs are in our replication package.

LION can find bugs in both the compilation stage (C) and the runtime stage (R). During compilation, LION found 7 bugs, which are related to compiler prompt blur (B) and inconsistent compilation prompts (I). 9 bugs are found in the runtime stage. They are related to time vectors inconsistent error (T) and data inconsistent error (D). To better illustrate these issues, we present several representative bugs identified by LION. For simplicity, all Simulink models that trigger bugs have been reduced.

TSC 05359223*: Compilation error prompts an unreasonable exception.* This type of bugs means that warning or error messages prompted by Simulink are vague or misleading. As illustrated in Fig. 4, the seed Simulink model initially compiles successfully. However, when we generated Simulink model variants for compilation, Simulink showed an error message for datatype inference "*data type mismatch*: *the internal rule determines the output of 'model/Subsystem/ PID 2dof/. . . is 'ufix32_En22'; however, it is driving a signal of data type 'double'*". This error message may not help trace the bug, since *PID 2dof* is a complex and encapsulated block in Simulink which can be hardly monitored and manipulated. The MathWorks support team has confirmed this bug and created a request for developers to enhance the error message. This bug is classified as compiler prompt blur, because the error message hinders developers for analyzing the bug.

TSC 05416819*: Time Vectors Inconsistent Error.* This type of bugs reflects errors in the sample time inference diagnostic. A typical example is shown in Fig. 5. The cause of this bug lies in Simulink incorrectly passing data from the *Discrete FIR Filter* block to the *Data Type Conversion* block, leading to incorrect time inference in the latter block. This bug has been confirmed. SLforge, SLEMI, and COMBAT may not generate such Simulink models, as it requires complex mutations involving long sequences of blocks.

TSC 05382868*: Data Inconsistent Error.* This type of bugs means that output data of two equivalent Simulink models are different. They are serious bugs since such bugs could inject unexpected behaviors in target Simulink applications. LION finds 8 bugs of this type. An example of Simulink model triggers this type of bug is presented in Fig. 6. Simulink compiler cannot correctly optimize Simulink models in the *Accelerator* simulation mode if a Simulink model has the *max* block with multiple input ports and *NaN* is the first value passed into it. The MathWorks support team considers this bug as serious, since it affects Simulink versions from R2015a to R2021b. The bug was fixed immediately after our submission. However, it was not found in existing studies.

TABLE III: Bugs found by LION in an evaluation period of two months.

#	TSC	Summary	Fb	Dp	St
1	5318668	S-Function too many output arguments	N	C	B
2	5359149	Equivalent mathematical function modules compile different prompts	N	C	I
3	5359223	Compilation error prompts an unreasonable exception	N	C	B
4	5370105	Exception in accelerate mode for if block	N	C	I
5	5382868	Data results different exception of max block	N	R	D
6	5394704	Min block produce incorrect results in accelerator modes with NaN input	K	R	D
7	5403026	Zero-crossing signal exception of the min block	N	C	I
8	5407559	Unknown module acceleration mode is abnormal	N	C	I
9	5411035	The Observer Block acceleration mode is not detected	N	C	I
10	5416819	Time vector inconsistency error of DTC block	N	R	T
11	5422544	Inconsistent Delay module data in different modes	N	R	D
12	5422545	Error in Sum module with onconsistent data result	N	R	D
13	5434051	Sinwave module data is inconsistent in different compilation	K	R	D
14	5436071	Ramp block is affected and produces different data in different modes	N	R	D
15	5445811	Problem of inconsistent data in the RT module	K	R	D
16	5445813	Data inconsistency errors when running the Product module	K	R	D

TSC = Technical Support Case number from MathWorks; Fb = feedback from MathWorks (N = new bug, K = known bug); Dp = discovery point (C = compilation stage, R = runtime stage); St = symptom of bugs (B = compiler prompt blur, I = inconsistent compilation prompts, T = time vectors inconsistent, D = data inconsistent error,).

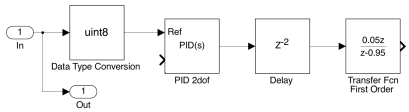


Fig. 4: TSC 05359223: Compilation error prompts an unreasonable exception.

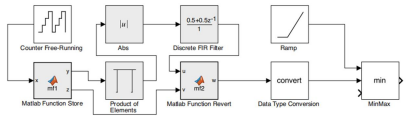


Fig. 5: TSC 05416819: Time Vectors Inconsistent Error.

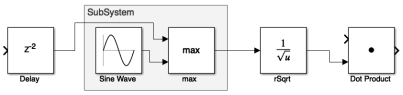


Fig. 6: TSC 05382868: Data inconsistent error.

V. Conclusions and Future Work

In this study, we propose LION, an approach for automatically generating equivalent Simulink models to test Simulink compilers. LION includes components for generating equivalent data flows and diverse model variants, addressing the challenges in compiler testing. LION modifies the data flow of a seed Simulink model while preserving equivalence, then uses an MCMC sampling strategy to generate diverse blocks. These models are used for equivalence modulo input testing. Evaluation results show that LION outperforms state-of-the-art methods in bug detection, finding 16 confirmed bugs in Simulink R2021b, 12 of which were previously unknown. Future work includes testing LION with more CPS development toolchains and analyzing the impact of testing time on bug-finding effectiveness.

ACKNOWLEDGMENT

This work was supported by the key research and development project of Liaoning Province (2024JH2/102400059), the National Natural Science Foundation of China (No.62472062, No.62202079), the Dalian Science and technology Innovation Fund project (No.2024JJ12GX022).

References

[1] S. A. Chowdhury, S. L. Shrestha, T. T. Johnson, and C. Csallner, "Slemi: Equivalence modulo input (emi) based mutation of cps models for finding compiler bugs in simulink," in *Int'l Conf. on Software Eng. (ICSE)*. Washington, USA: IEEE, 2020, pp. 335–346.

[2] MathWorks Inc, "Products and services," http://www.mathworks.com/products/, 2018.

[3] J. Zander, I. Schieferdecker, and P. J. Mosterman, *Model-Based Testing for Embedded Systems*. Boca Raton, USA: CRC press, 2017.

[4] S. A. Chowdhury, T. T. Johnson, and C. Csallner, "Cyfuzz: A differential testing framework for cyber-physical systems development environments," in *Cyber Physical Systems Design, Modeling, and Evaluation*. Cham: Springer International Publishing, 2017, pp. 46–60.

[5] S. A. Chowdhury, S. Mohian, S. Mehra, S. Gawsane, T. T. Johnson, and C. Csallner, "Automatically finding bugs in a commercial cyber-physical system development tool chain with slforge," in *Int'l Conf. on Software Eng. (ICSE)*. New York, NY, USA: ACM, 2018, pp. 981–992.

[6] MathWorks Inc, "Bug report platform," https://www.mathworks.com/support/bugreports/, 2018, (last access: 16/03/2022).

[7] A. Brillout, N. He, M. Mazzucchi, D. Kroening, M. Purandare, P. Rümmer, and G. Weissenbacher, "Mutation-based test case generation for simulink models," *Int'l Symposium on Formal Methods for Components and Objects (FMCO)*, vol. 6286, p. 208–227, 2009.

[8] S.Guo, H.Jiang, Z.Xu, X.Li, Z.Ren, Z.Zhou, and R.Chen, "Detecting simulink compiler bugs via controllable zombie blocks mutation," in *Joint European Software Eng. Conf. and Symposium on the Foundations of Software Eng.(ESEC/FSE)*, 2022, pp. 1061–1072.

[9] C. Sun, V. Le, and Z. Su, "Finding compiler bugs via live code mutation," 2016, p. 849–863. [Online]. Available: https://doi.org/10.1145/2983990.2984038

[10] Y. Chen, T. Su, and Z. Su, "Deep differential testing of jvm implementations," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 2019, pp. 1257–1268.

[11] LION, "Lion replication package," https://github.com/Z-Lighthouse/LION, 2024.

[12] Y. Cheng, Z. Yu, Z. Su, T. Chen, X. Zhang, and Y. Jiang, "Accmos: Accelerating model simulation for simulink via code generation," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[13] Z. Su, Z. Yu, D. Wang, R. Wang, Y. Tao, and Y. Jiang, "Cftcg: Test case generation for simulink model through code based fuzzing," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[14] Z. Yu, Z. Su, Y. Jiang, A. Cui, and R. Wang, "Efficient code generation for data-intensive simulink models via redundancy elimination," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[15] X. Yang, Y. Chen, E. Eide, and J. Regehr, "Finding and understanding bugs in c compilers," in *ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. New York, NY, USA: ACM, 2011, pp. 283–294.

[16] MathWorks Inc, "Signal range coverage tool," https://ww2.mathworks.cn/help/slcoverage/ug/types-of-model-coverage.html, 2022, (last access: 16/03/2022).

[17] C. Sun, V. Le, and Z. Su, "Finding compiler bugs via live code mutation," in *Int'l Conf. on Object-Oriented Programming, Systems, Languages, and Applications (SIGPLAN)*. New York, NY, USA: ACM, 2016, pp. 849–863.

[18] Y. Tang, H. Jiang, Z. Zhou, X. Li, Z. Ren, and W. Kong, "Detecting compiler warning defects via diversity-guided program mutation," *IEEE Trans. on Software Eng. (TSE)*, pp. 1–1, 2021.

[19] S. A. Chowdhury, L. S. Varghese, S. Mohian, T. T. Johnson, and C. Csallner, "A curated corpus of simulink models for model-based empirical studies," in *Proceedings of the 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems*, 2018, pp. 45–48.

[20] H. Jiang, Z. Zhou, Z. Ren, J. Zhang, and X. Li, "Ctos: Compiler testing for optimization sequences of llvm," *IEEE Trans. on Software Eng. (TSE)*, pp. 1–1, 2021.

SDISC: A Spike-Driven Human-Machine Interface with In-Situ Computing for Real-Time Low-Power Interaction

Fangduo Zhu[1†], Jingyi Chen[1, 2†], Jingsong Zhang[1], Xumeng Zhang[1*], Siyuan Ouyang[1],
Chenyang Li[1], Hao Jiang[1], Xiaonan Yang[2*], Qi Liu[1*]
[1]State Key Lab of Integrated Chips and Systems, Frontier Institute of Chip and System, Fudan University, Shanghai, China;
[2]School of Electrical and Information Engineering, Zhengzhou University, Zhengzhou, China;
Email: xumengzhang@fudan.edu.cn; iexnyang@zzu.edu.cn; qi_liu@fudan.edu.cn;

***Abstract*— Feature extraction and classification of bio-signals are crucial in human-machine interface (HMI), yet suffer from high delay and limited energy efficiency using conventional hardware. To mitigate this challenge, we propose an SDISC architecture, a neuromorphic HMI with the innovation from signal encoding, computing-in-memory (CIM) hardware, to algorithm-hardware co-optimization. The following strategies are implemented: (1) A spike-driven feature extractor, achieving >10× sparser dataflow than frame-based method; (2) In-situ computing based on resistive random-access memory (RRAM), enabling energy-efficient (4.09 TOPS/W) spiking neural network (SNN) classifier; (3) A *Spike-Activity-Distillation* algorithm and an *Aid-Loser-Only* recovery scheme to alleviate the non-ideality of RRAM devices, ensuring SDISC maintains high accuracy (∼ 98.0%) in long time inference (>15 days). We further develop an end-to-end SDISC system for real-time EMG-based robot control, achieving a low latency (34 µs) and low power (39.72 µW/sample) interaction on edge.**

***Index Terms*—neuromorphic HMI, spiking neural network, RRAM, computing-in-memory, edge computing**

I. Introduction

In the era of artificial intelligence, the demand for always-on, low-power, and real-time HMI is increasingly emphasized, particularly in fields like virtual reality [1], [2], health monitoring [3] , and prosthetics [4]. In end-to-end HMI systems, the rapid processing of bio-signals (e.g. EMG) under limited battery power is the basis of smooth interaction. State-of-the-art HMI architectures that employ the frame-based paradigm (Fig. 1a) suffer from high energy consumption and compute delay due to redundant data buffering and transfer. Feature extraction (FE) and classification are the two main stages of HMI. The FE module picks up meaningful information from noisy bio-signals and transfers the representation in frames for the subsequent classification module. Although noise removal [5], filtering [4], [6], or peak searching [7] are elegantly realized, current FE modules still occupy a high ratio (41.7% in [7]) of the overall system power. Moreover, since the FE modules are designed to produce dense framed data, the caching process for the classifier is inevitable, inducing high latency (100 ms in [8]). For classification processes, an artificial neural network (ANN) is generally employed, which is well supported by CIM techniques [9], especially those based on non-volatile memories like RRAM [10], attributing to the low standby power and outstanding energy efficiency during compute [11]. However, the frame-based processing flow of ANN causes ineffective waiting between frames in CIM engines, which accumulates substantial delays for real-time HMI, as depicted in Fig. 1b. Besides, the frame controller [12] and the input memory caching [13] introduce additional energy consumption.

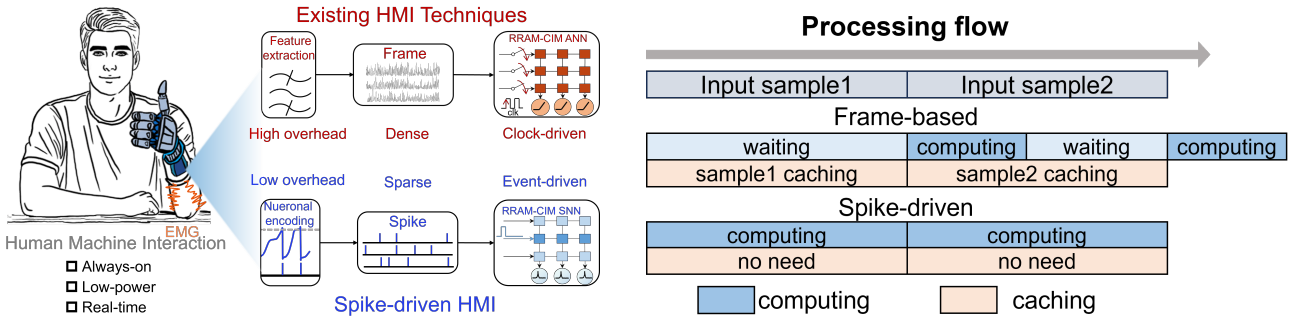
Neuromorphic hardware like Xylo [14] and Speck [15], inspired by biological systems, leverages event-driven spikes to encode signals and perform computations in the spatiotemporal domain, which makes them free of frames and suitable for processing temporal tasks. However, several challenges remain in developing a neuromorphic HMI. For example, common spiking FE modules based on differential encoding [16] cannot be reused in the SNN classifier, increasing the overall circuit overhead (energy and area cost). In addition, the exploitation of CIM techniques in spike-driven paradigm [17] still lacks hardware-algorithm co-design to promote the system robustness. Thus, building a neuromorphic HMI combining the merits of spike-driven and CIM deserves more attention.

In this work, we propose an SDISC, a spike-driven in-situ computing architecture, conduct algorithm-hardware co-optimization, and finally demonstrate an end-to-end SDISC hardware system for EMG-based robot interaction in real-time. The main contributions are summarized as follows:

- We proposed a reconfigurable spike encoding FE module with the wake-up function, achieving 10× sparser dataflow compared to the frame-based paradigm.
- We built an SNN classifier based on the RRAM-CIM strategy, with an energy efficiency of 4.09TOPS/W, achieving 41× energy efficiency improvement to conventional GPU.
- To enhance the reliability and robustness of RRAM-CIM HMI, a *Spike-Activity-Distillation* algorithm and an *Aid-Loser-Only* recovery scheme were proposed, achieving 98.0% accuracy in a long time (>15 days) on-chip inference.

In the experimental results, the SDISC hardware system demonstrates low power consumption of 39.72 µW/sample and low latency of 34 µs on EMG signal processing, achieving 9.4×

†: These authors contributed equally; *: Corresponding authors. This work was supported by the National Natural Science Foundation of China (Nos. T2293732, 62374040, 62488101), the Shanghai RisingStar Program under Grant No. 24QA2700400, the National Key Laboratory of Human-Machine Hybrid Augmented Intelligence, Xi'an Jiaotong University under Grant No. HMHAL-202415.



(a) (b)

Fig. 1. (a) Requirements of the hardware for human-machine interaction (left). Challenges on current HMI (right top) and our solutions (right bottom). (b) Processing flow comparison between frame-based and spike-driven classifiers.

faster than the frame-based paradigm in a real-world EMG-controlled robot maze-walking task.

II. Preliminaries

A. Working principle of spiking neural networks

The processing of bio-signals in HMI is an inherently temporal sequence task. In SNN, the spiking neurons extract meaningful temporal representations with intrinsic dynamics, making them well-suited for such sequential tasks. Among the commonly used neuron models, Leaky Integrate and Fire (LIF) model stands out due to its relatively low computational budget. The state "membrane potential" v and binary spike o at the timestep t of the LIF model evolves as (1).

$$v[t] = (1 - 1/\tau) \cdot v[t-1] + \sum_j w_{ij} o_j[t] - \vartheta o[t-1] \quad (1a)$$

$$o[t] = s(v[t] - \vartheta) \quad (1b)$$

$$s(x) = 0 \text{ if } x < 0, \text{ otherwise } 1 \quad (1c)$$

The $\sum_j w_{ij} o_j[t]$ term is in the same form as weighted summation in ANN. In the absence of input spikes o_j, v leaks with a time constant τ. $s(\cdot)$ denotes spike function, indicating that once v surpasses the neuronal threshold ϑ, an spike o occurs and then v is soft reset to $v - \vartheta$. In this work, we use the "arctan" function [18] for the surrogate gradient of $s(\cdot)$ in training. The f in (2) indicates the output frequency of neurons in a whole time window T , which gives the prediction result for inference, where $g^l(\boldsymbol{x}) = \text{LIF}(\boldsymbol{W}^l \boldsymbol{x})$ denotes the l-th layer, $\circ$ represents a feedforward operation.

$$f = \frac{1}{T} \sum_{t=1}^{T} g^l \circ g^{l-1} \circ \cdots g^1(\boldsymbol{x}) \quad (2)$$

B. In-situ computing with RRAM

As (3) shows, an $M \times N$ RRAM array enables in-situ computing by performing voltage dot conductance operations directly within memory cells. This approach eliminates the need to transfer SNN weights between off-chip memory and compute units, significantly enhancing efficiency [19]. CIM architectures based on RRAM offer distinct advantages, including low standby power and high synaptic weight density, attributed to the nonvolatile and multi-level characteristics of RRAM devices. Therefore, RRAM-CIM engines are not only far more energy efficient than von Neumann architectures in the compute stage of HMI but also maintain exceptionally low static power in idle states [20]. These features make RRAM-CIM engines particularly well-suited for always-on HMI, where low power and fast response are critical.

$$I_n = \sum_{m=1}^{M} G_{m,n} V_m \quad (3)$$

Although the RRAM-CIM engines hold great benefits as classifier, it faces several devices' non-ideal characters that can pose significant challenges to inference accuracy. First, the conductance variations and conductance drift during reading are unavoidable. Second, after a long-duration high-intensity inference, an RRAM cell may jump from a low conductance state to a high conductance state, or vice versa, referring to as the degradation problem and destroying the system accuracy. To alleviate these issues, an algorithm-hardware co-optimization strategy is essential for improving the reliability and robustness of HMI with RRAM-CIM.

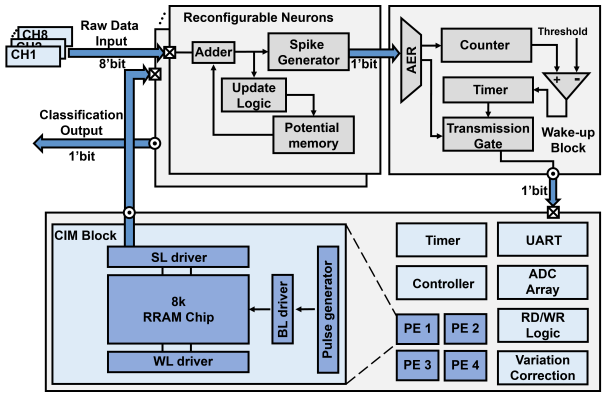
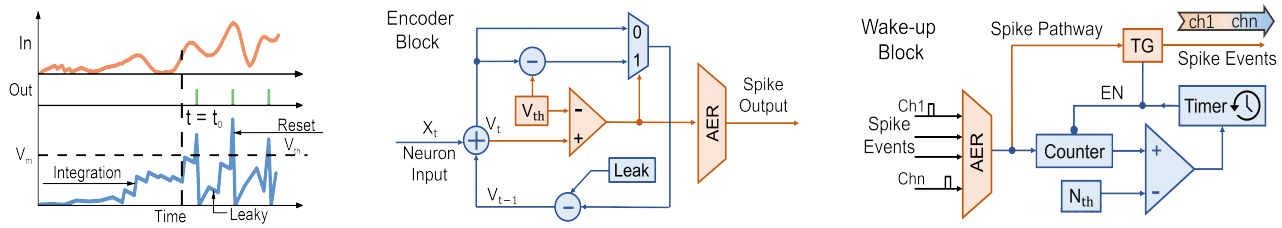


Fig. 2. Overall architecture of spike-driven HMI system.



(a) (b) (c)

Fig. 3. (a) Diagram of PL-LIF neuron executing encoding. (b) Digital circuit implementation of PL-LIF neuron. (c) Wake-up circuit scheme.

III. SDISC Paradigm

In this section, we outline the overall architecture of the proposed SDISC hardware system and the key circuit designs, present the SNN implementation scheme, and finally declare the methodologies of algorithm-hardware co-optimization.

A. Overall hardware architecture

The SDISC system processes bio-signals in 1-bit spikes instead of multi-bit frames, enabling ultra-low latency and power consumption. Fig. 2 shows the overall architecture of SDISC, which consists of a reconfigurable spiking neuron block, a wake-up block, and an RRAM-CIM block. The spiking neuron block handles both spike encoding in the FE and subsequent classification processes. The wake-up block monitors the input spike sparsity and initiates system startup when necessary. The RRAM-CIM block acts as an SNN classifier accelerator, comprising four RRAM processing elements (PE), read/write peripheral circuits and other functional modules. The system workflow begins with EMG data being filtered and encoded into spike events by the spiking neurons. These spike events are then evaluated by the wake-up block to determine whether the system should activate. Once awakened, the CIM block and spiking neurons collaboratively perform SNN classification. This hierarchical design ensures efficient operations, activating compute-intensive processes only when needed, thus optimizing power and latency performance.

B. Low-hardware-cost reconfigurable spiking neurons

Spiking neurons are foundational to sparse encoding and the event-driven paradigm. For hardware implementation, neuron design must balance functionality with hardware complexity. However, the leaky operation of current LIF model in (1a) is of high hardware cost, because the neuron circuit has to perform a multi-bit multiplication for every single input spike. Although some solutions like leaky LUT [21] or right shift [22] operations simplify the computing flow of leaky, they weaken the neuronal capability of temporal representation by restricting the time constant τ to a fixed value. To solve this issue, we propose a parametric linear LIF (PL-LIF) model as (4a), in which the leaky operation is optimized to multi-bit subtraction as the linear subtraction item $-(1/\tau)\cdot\vartheta$ shows.

$$v[t] = v[t-1] - (1/\tau)\cdot\vartheta + \sum_j w_{ij} o_j[t] - \vartheta o[t-1] \quad (4a)$$

$$\tau = \sqrt{1+\alpha^2}/\alpha \quad (4b)$$

To enhance the temporal information extraction capability of PL-LIF neuron, the time constant τ in PL-LIF is reconfigurable for both FE and classifier blocks, as shown in (4b), where the parameter α is trainable. This reconfigurability ensures that the PL-LIF neurons can adapt to diverse temporal patterns while maintaining hardware efficiency.

C. Spiking encoder and wake-up module

SNN inherently eliminates noisy signals that are less frequent in temporal domain, performing filtering for FE. In SDISC, each input channel connects to a PL-LIF neuron, which converts EMG signals into spike trains. As illustrated in Fig. 3a, the PL-LIF neuron effectively filters out meaningless signals ($t < t_0$) , ensuring that only signals with substantial amplitude trigger spikes, thereby enhancing data sparsity. The digital circuit diagram of a PL-LIF is shown in Fig. 3b. To ensure the battery life in always-on HMIs, a wake-up module illustrated in Fig. 3c is designed to isolate and filter out the spikes generated from muscle resting. Encoded spikes from all input channels with address event representation (AER) are monitored. Only when the total spike number counted in a small timewindow exceeds a set value (N_{th}), the encoded

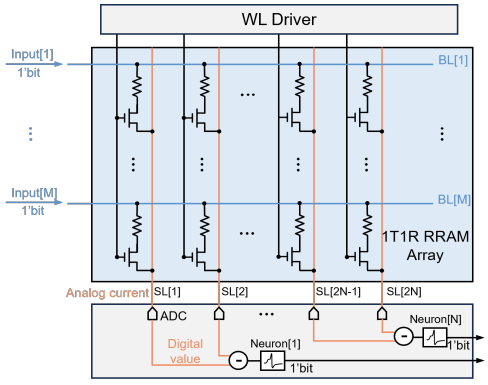


Fig. 4. Inference strategy of SNN on an RRAM-CIM array.

spikes start to pass through the transfer gate (TG) and proceed to the classifier.

D. RRAM-based SNN classifier

To construct an energy-efficient classifier, we train and implement a three-layer fully connected SNN on an RRAM-CIM block. Each layer of the SNN is mapped onto the RRAM PEs, with each PE comprising a 64×128 one-transistor-one-resistor (1T1R) crossbar array. The inference flow of the SNN on the RRAM crossbar is shown in Fig. 4, where input pulses are applied to the bit lines (BLs). When spikes come, only the selected BL rows of RRAM execute in-situ computing, and the current accumulate over the source lines (SL). For all layers, the analog multiple accumulation (MAC) values are converted to digital values, and the differential results of adjacent SLs serve as the input of PL-LIF neurons. In the output layer, the neuron with the highest firing rate represents classification results. This design leverages the advantages of RRAM-based in-memory computing to minimize energy consumption.

To optimize the compute density, we exploit the multi-level conductance programming capability of RRAM. Using quasi-analog [23] mapping with a write-verify process, each differential pair is programmed to represent a weight w_{ij} as (5a) shows.

$$G_{diff} = (G_{max} - G_{min})/w_{max} \cdot w_{ij} + \Delta G \tag{5a}$$

$$\mathcal{L}_w = [(|w| - w_{\max})_+]^2 \tag{5b}$$

G_{diff} is the conductance after mapping, $[G_{min}, G_{max}]$ is the predefined conductance range (20 μS∼120 μS), and ΔG is the write-verify deviation (5 μS). In the training process, we adjust the positive and negative weights to have the same absolute upper bound w_{max} by introducing an extraloss function item $\mathcal{L}_w$ as (5b), where $(\cdot)_+$is equivalent to $\max\{\cdot, 0\}$. This approach ensures a well-balanced distribution of target conductance, preserves the precision of the mapped weights and provides a solid foundation for accurate inference.

E. Algorithm-hardware co-optimization for robust HMI

Despite the progress made, achieving a reliable and robust HMI remains a significant challenge. The inherent non-idealities of RRAM devices and the inevitable noise in analog circuits threaten the systems' robustness. As shown in Fig. 5a, the readout results of multi-state conductance exhibit fluctuations, leading to variations in the corresponding SNN weights at each timestep. This creates a disparity between the pre-trained SNN model and its on-chip implementation. Moreover, under heavy-stress usage for over 15 days, certain unbearable conductance degradation happens (see Fig. 5b). To prevent the accuracy instability caused by these non-idealities, we propose two algorithm-hardware co-design strategies as follows.

***Spike-Activity-Distillation* algorithm.** The key idea is training the network with conductance variation to find optimal τ for PL-LIF neurons so that they can mitigate the negative impact of noise accumulation. For example, if the neuronal input (accumulated noisy MAC variations) over a certain period of time is higher than expected, the time constant τ of PL-LIF should be smaller to get similar firing activity as that under ideal situation without noise. Based on this idea, we design a *Spike-Activity-Distillation* (*SAD*) scheme that contains two-stages. In stage one, a high-accuracy teacher SNN is trained with τ frozen as a hyperparameter. Then it comes to stage two as Algorithm 1 shows. We map the weights of teacher net NET_{Tea} to RRAM and sample the noise weight matrices $\boldsymbol{\mathcal{W}}_\sigma$ for a student net NET_{Stu}. The goal of NET_{Stu} is to adjust its spike activity (output firing rate in this work) closer to the $\text{NET}_{\text{Teacher}}$ by modifying $\boldsymbol{\tau}_{\mathbf{S}}$ layer by layer, enabling it to learn the optimal spike output patterns for classification. *SAD* is hardware-friendly because the SDISC system only need to reconfigure the linear subtraction values of each spiking layer after finishing the two stages. As a result, the NET_{Stu} can tolerate the non-ideal variations of RRAM during inference with a low-cost circuit calibration.

Algorithm 1 Spike Activity Distillation

$$\mathcal{D} = \left\{\left(\boldsymbol{x}^{(n)}, y^{(n)}\right)\right\}_{n=1}^{N}, \boldsymbol{\mathcal{W}}_\sigma = \left\{\left(\boldsymbol{w}_\sigma^{(l,t)}\right)\right\}_{l=1,t=1}^{L,T}$$

Input: NET_{Tea} with $\boldsymbol{\mathcal{W}} = \left\{\left(\boldsymbol{w}^{(l)}\right)\right\}_{l=1}^{L}$ and τ_{T}
Sample number N_s , Epoch E , Validation set V

Output: Time constant $\boldsymbol{\tau}_{\mathbf{S}} = \left\{\tau_s^{(l)}\right\}_{l=1}^{L}$ of NET_{Stu}

Initialize $\boldsymbol{\tau}_{\mathbf{S}} \leftarrow \tau_{\text{T}}$
repeat
  Shuffle the training set $\mathcal{D}$
  for each training data $i \in \mathcal{D}$ **do**
    Feed forward NET_{Tea} and get the spike activity O_T
    while sample number $< N_s$ **do**
      Sample weight set $\boldsymbol{\mathcal{W}}_\sigma$ with device variation
      Load timestep-varied weight $\boldsymbol{w}_\sigma^{(l,t)}$ to NET_{Stu}
      Feed forward NET_{Stu} for spike activity O_S
      Compute the MSE error ϵ between O_S and O_T
      Compute loss $\mathcal{L} = \mathcal{L}_{cross\ entropy} + \lambda\mathcal{L}_{MSE}$
      Back propagation through time and update $\boldsymbol{\tau}_{\mathbf{S}}$
    end while
  end for
until epoch E or accuracy does not increase on V
return Trained $\boldsymbol{\tau}_{\mathbf{S}}$

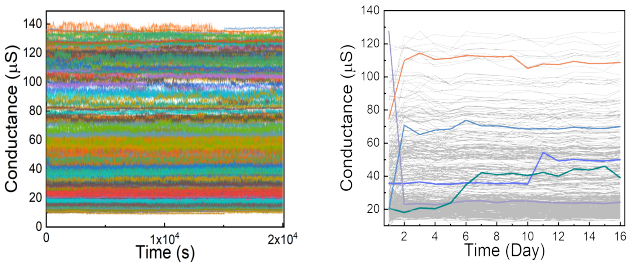


Fig. 5. (a) Measured conductance read variation of 8k RRAM array. (b) Degradation issue occurs after 15 days' inference

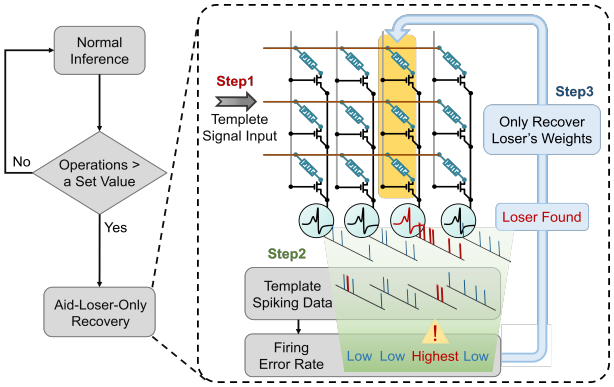


Fig. 6. The ALO recovery scheme.

***Aid-loser-only* recovery scheme.** To address the degradation issue and ensure the long time reliability of the RRAM-CIM block in HMI, we propose an *Aid-loser-only (ALO)* recovery scheme to restore off-track devices. The schematic of the *ALO* is shown in Fig. 6. Specifically, the method is to spot check the output neurons' firing activity with a template data (from the class with the lowest accuracy). The spiking neuron with the highest firing rate deviation compared to itself at the very beginning is defined as the loser that should be aided. It should be noted that only partial RRAM synapses connected to the loser (in the same columns) are recovered rather than the entire RRAM array [12].

IV. System Implementation and Metrics Evaluation

To validate the feasibility of our SDISC architecture, we develop a sensor-end to robot-end HMI system for EMG-based closed-loop control in real time. Additionally, we evaluate the key performance metrics such as signal processing latency, power consumption, and system robustness. The experimental details and result analyses are declared as below.

A. End-to-end SDISC system

Dataset. Myo surface EMG armband is chosen to create a dataset for training. Five distinct hand gestures (fist, finger abduction, wrist extension, wrist flexion, and rest) are recorded, each of which provides 1000 samples (5000 samples in total). Each sample is an 8-channel data segmented to 75ms. Our recorded EMG dataset is available at [24].

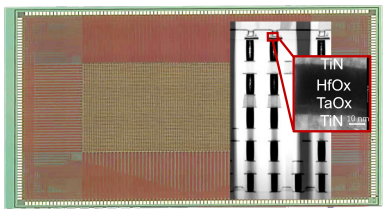


Fig. 7. Die photo and TEM image of a 64×128 1T1R RRAM array. Insert details layer composition.

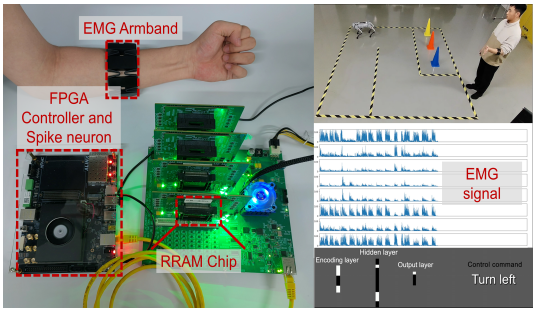


Fig. 8. The diagram of SDISC real-time system.

TABLE I
Latency Comparison

Implementation	Spike-driven on SDISC	Frame-based on RRAM-CIM	Spike-driven on GPU
Compute Latency	34 µs	320 µs	70.4 ms

Experimental Platform. Four 8k RRAM arrays are fabricated with 180 nm BEOL standard process and then driven by a board system with peripheral circuits. The die photo is shown in Fig. 7. A CPU (Intel i9-12900) and a GPU (Nvidia Tesla V100) are baselines for energy efficiency evaluation. In the real-time demonstration, an FPGA board (Xilinx XCZU15EG) serves as the global controller and spiking neurons. To evaluate the power consumption of an integrated SDISC system, we customize spiking neurons under 180nm technology and simulate their performance using the Cadence Virtuoso tool. An ANN with the same network structure as our SNN is implemented on the RRAM-CIM block for comparison. We obtain speedup on a GPU (Nvidia RTX3090) benchmark.

HMI System Implementation. As Fig. 8 illustrates, we demonstrate a spike-driven HMI system working at 50MHz to perform low-latency control in a closed-loop robot maze-walking task. The right side of Fig. 8 displays the real-time scene conditions, raw EMG signals, and inference commands.

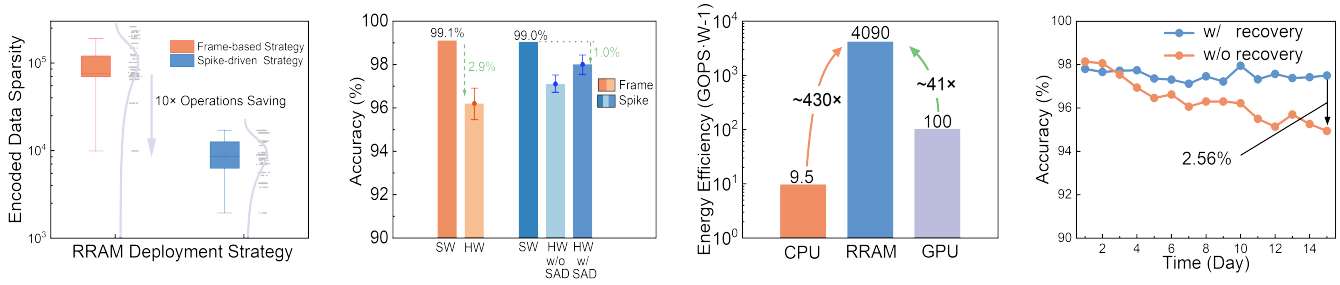
B. Comparison to frame-based processing

For comparison, We employ the same frame-based processing method as [25] to deploy the ANN on the RRAM-CIM block. Our SDISC scheme outperforms in multiple metrics:

Dataflow sparsity. As shown in Fig. 9a, the spike-driven strategy achieves 10× data sparsity compared to the frame-based strategy, significantly reducing operations in the RRAM-CIM block and forming the basis of our low-power design.

Accuracy. Fig. 9b illustrates the implementation results of frame-based and spike-driven inference on RRAM-CIM. Both of them have a subtle accuracy loss when compared to the software results. However, the spike-driven scheme is 0.9% superior to the frame-based strategy on chip, and has a smaller standard deviation (the error bars). This result can be explained by the temporal filtering ability of spiking neurons, which neutralize up and down variations over time.

Compute latency. We test and summarize the compute latency under different implementation schemes. As Table



(a) (b) (c) (d)

Fig. 9. (a) Data sparsity comparison between frame-based and spike-driven strategies. (b) Classification accuracy comparison between frame-based and spike-driven compute paradigm on software/hardware. (c) Energy efficiency comparison between implementations on CPU, GPU and our RRAM-CIM block. (d) Evolution of inference accuracy in two trials of experiment, respectively w/ and w/o *ALO* recovery.

TABLE II
SUMMARY AND BENCHMARKS

Reference	**This work**	**TBioCAS19'** [26]	**BioCAS23'** [5]	**ODIN** [27]	**Loihi** [28]	**IEDM23'** [29]	**JSSC24'** [30]
Data Type	EMG	EMG	EMG	EMG	EMG	Image	Image
Hardware	RRAM CIM	Digital-Analog	FPGA	Digital	Digital	RRAM CIM	RRAM CIM
Input/weight precision	1b/ quasi-analog	1b/ 2b	8b/ 8b	1b/ 4b	1b/ 4b	1b/ 4b	8b/ 8b
Power Consumption	39.72 µW	50 µW	493 mW	315 µW	29.4 mW	N/A	N/A
Energy Efficiency	4.09 (TOPS/W)	N/A	N/A	N/A	N/A	1.547 (TOPS/W)	51.4 (TOPS/W)
Normalized Energy Efficiency	$\geq$ 65.44 (TOPS/W)	N/A	N/A	N/A	N/A	1.547 (TOPS/W)	51.4 (TOPS/W)
Event-driven	✓	✓	×	✓	✓	×	×
End-to-end system	✓	✓	✓	×	×	×	×
Latency	34µs	N/A	44.36 µs	23.5 ms	173 µs	875 µs	472.7 µs

I shows, spike-driven processing equipped with the SDISC system produces control commands within only 34 µs. When implemented on SDISC, the spike-driven scheme provides a 9.4× speedup for classification as the frame-based scheme, which proves the superiority of our processing paradigm. Furthermore, our SDISC hardware system is 2070× faster than GPU when executing the same spike-driven paradigm, indicating the high efficiency of our hardware system design.

C. Energy consumption analysis

The RRAM-CIM block exhibits an outstanding energy efficiency of 4.09 TOPS/W, which is 430× that of CPU and 41× that of GPU, as illustrated in Fig. 9c. Such metric can be normalized to $\geq$65.44 TOPS/W due to the equivalent precision ($\geq$4 bit) in our RRAM implementation. As for the power consumption of the entire HMI system working in our real-world control task, the RRAM+FPGA implementation consumes 32 mW for a sample on average. According to the results of transient simulation and analysis, such power consumption metric can be optimized to 39.72 µW by integrating our low-standby-power RRAM synapses and low-compute-complexity spiking neurons together.

D. System robustness evaluation

Ablation studies are conducted to validate the effectiveness of the proposed *SAD* and *ALO* strategies. As the rightmost two bars in Fig. 9b reveal, the control command accuracy rises up to 98.0% after introducing the *SAD* scheme, which indicates that the SDISC HMI system becomes more reliable. Furthermore, two trials of experiments for 15 days are conducted to test the performance of our recovery scheme in a long-time inference. Fig. 9d indicates the evolution of accuracy in the whole course. The accuracy remains nearly stable with *ALO* recovery, while it drops by up to 2.56% without recovery. Such results prove the improvement of HMI's robustness due to our algorithm-hardware co-optimization.

V. CONCLUSION

In this paper, we propose SDISC, a neuromorphic HMI featuring spike-driven and RRAM-based in-situ computing for real-time and low-power processing of EMG. The SDISC paradigm achieves sparser dataflow, high inference energy efficiency, and low compute time delay. An *SAD* strategy and an *ALO* recovery mechanism counteract the non-idealities of RRAM devices and enhance the reliability of the HMI system. Compared to the representative works in Table II, our HMI system exhibits lower latency (34 µs/sample) and lower power consumption (39.72 µW/sample) in an EMG-based robot maze walking task. This work provides insights for developing always-on HMIs in end-to-end interaction scenarios.

REFERENCES

[1] J. Ryu et al., "AI SoCs for AR/VR User-Interaction," *IEEE International Electron Devices Meeting (IEDM)*, pp. 35.4.1–35.4.4, 2021.

[2] M. Abrash, "Creating the future: augmented reality, the next human machine interface," *IEEE International Electron Devices Meeting (IEDM)*, pp. 1–2, 2021.

[3] U. Shin et al., "A 256-Channel 0.227 μJ/class Versatile Brain Activity Classification and Closed-Loop Neuromodulation SoC with 0.004 mm 2-1.51 μW/channel Fast-Settling Highly Multiplexed Mixed Signal Front-End," *IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, pp. 338–340, 2022.

[4] A. Furui et al., "A myoelectric prosthetic hand with muscle synergy–based motion determination and impedance model–based biomimetic control," *Sci. Robot.*, vol. 4, no. 31, 2019.

[5] Y. Guo et al. "sEMG-Based Wearable HMI System for Real-Time Robotic Arm Control with Edge AI," *IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pp. 1-5, 2023.

[6] J.-H. SeoI et al., "A 1.5 μW end-to-end keyword spotting SoC with content-adaptive frame sub-sampling and fast-settling analog frontend," *IEEE Int. Solid-State Circuits Conf.(ISSCC)* Dig. Tech. Papers, pp. 1-3, 2023.

[7] Y. Chen et al., "A 384-Channel online-spike-sorting IC using unsupervised Geo-OSort clustering and achieving 0.0013 mm^2/Ch and 1.78 μW/ch," *IEEE Int. Solid-State Circuits Conf.(ISSCC)*, pp. 486-488, 2023.

[8] Z. Wang et al., "12.1 a 148nw general-purpose event-driven intelligent wake-up chip for aiot devices using asynchronous spike-based feature extractor and convolutional neural network," *IEEE Int. Solid-State Circuits Conf.(ISSCC)*, vol. 64, pp. 436–438, 2021.

[9] L. Chang et al., "Tdpro: Time-domain-based computing-in memory engine for ultra-low power ecg processor," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 10, pp. 3908–3919, 2023.

[10] W. Ye et al., "A 28-nm RRAM Computing-in-Memory Macro Using Weighted Hybrid 2T1R Cell Array and Reference Subtracting Sense Amplifier for AI Edge Inference," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 10, pp. 2839-2850, 2023.

[11] M. Chang et al., "A 40nm 60.64 TOPS/W ECC-capable compute-in-memory/digital 2.25 MB/768KB RRAM/SRAM system with embedded cortex M3 microprocessor for edge recommendation systems," *IEEE Int. Solid-State Circuits Conf.(ISSCC)*, Vol. 65, pp. 1-3, 2022.

[12] W. Li et al., "A 40-nm MLC-RRAM compute-in-memory macro with sparsity control on-chip write-verify and temperature-independent ADC references," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 9, pp. 2868–2877, 2022.

[13] H. Jiang et al., "A $2.53\mu\mathrm{W}/\mathrm{channel}$ Event-Driven Neural Spike Sorting Processor with Sparsity-Aware Computing-In-Memory Macros," *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5, 2023.

[14] D. Muir et al., "Sub-mw neuromorphic snn audio processing applications with rockpool and xylo," *Embedded Artificial Intelligence. River Publishers*, 69-78, 2023.

[15] M. Yao et al., "Spike-based dynamic computing with asynchronous sensing-computing neuromorphic chip," *Nature Communications*, 15(1), 4464, 2024.

[16] Y. Liu et al., "An 82nW 0.53 pJ/SOP clock-free spiking neural network with 40μs latency for AloT wake-up functions using ultimate-event-driven bionic architecture and computing-in-memory technique," *IEEE Int. Solid-State Circuits Conf.(ISSCC)*. Vol. 65, 2022.

[17] Y. Li et al., "Input-aware dynamic timestep spiking neural networks for efficient in-memory computing," *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

[18] W. Fang et al., "Deep residual learning in spiking neural networks," *Advances in Neural Information Processing Systems* 34:21056-21069, 2021.

[19] C. Xue et al., "A 22 nm 4 Mb 8b-precision ReRAM computing-inmemory macro with 11.91 to 195.7TOPS/W for tiny AI edge devices," *IEEE ISSCC Dig. Tech. Papers*, Feb. 2021, pp. 245–247.

[20] S. Yu et al., "RRAM for compute-in-memory: From inference to training," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 7, pp. 2753–2765, Jul. 2021.

[21] J. Zhang et al., "ANP-I: A 28-nm 1.5-pJ/SOP Asynchronous Spiking Neural Network Processor Enabling Sub-0.1-μJ/Sample On-Chip Learning for Edge-AI Applications," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 8, pp. 2717-2729, 2024.

[22] M. Bensimon et al., "A New SCTN Digital Low Power Spiking Neuron," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 8, pp. 2937-2941, 2021.

[23] H. Zhao et al., "Implementation of Discrete Fourier Transform using RRAM Arrays with Quasi-Analog Mapping for High-Fidelity Medical Image Reconstruction," *IEEE International Electron Devices Meeting (IEDM)*, pp. 12.4.1-12.4.4, 2021.

[24] J. Chen, "SNN-HMI Dataset", *GitHub Repository*, 2024. Available: https://github.com/jermycc/SNN-HMIdataset.

[25] P. Yao et al., "Fully hardware-implemented memristor convolutional neural network," *Nature* 577, 641–646, 2020.

[26] E. Donati et al., "Discrimination of EMG signals using a neuromorphic implementation of a spiking neural network," *IEEE transactions on biomedical circuits and systems*, 13(5), 795-803, 2019.

[27] C. Frenkel et al., "A 0.086-mm2 12.7-pJ/SOP 64k-Synapse 256-Neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS," *IEEE Trans. Biomed. Circuits Syst.*, vol. 13, no. 1, pp. 145–158, Feb. 2019.

[28] Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan. 2018.

[29] X. Zheng et al., "Point-of-Care Testing (POCT) System based on self-Recovery Memoristor Chip with Low Energy Consuption(1.547 TOPS/W) and High Recognition (1142 fram/s)," *IEEE International Electron Devices Meeting (IEDM)*, pp. 1-4, 2023.

[30] H.H. Hsu et al., "A nonvolatile AI-edge processor with SLC–MLC hybrid ReRAM compute-in-memory macro using current-voltage-hybrid readout scheme," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 1, pp. 116–127, 2024.

MemSeer: Leverage Memory Failure Distinctions and Multi-Grained Prediction in Ultra-Scale Heterogeneous X86/ARM Clusters

Yunfei Gu†, Yixuan Liu†, Xinyuan Wu†, Bo Shao†, Chentao Wu†*, Shiyi Li‡, Jieru Zhao†,
Jie Li†, Minyi Guo†, Kunlin Yang§, Wengui Zhang§, Feilong Lin§
†Shanghai Jiao Tong University, Shanghai, China
‡Harbin Institute of Technology, Shenzhen, China
§Huawei Technologies Co., Ltd., China
*Corresponding Authors: wuct@cs.sjtu.edu.cn

***Abstract*—In high-performance ultra-scale cloud computing, heterogeneous clusters consisting of x86 and ARM architecture platforms have become increasingly common to boost performance and energy efficiency. Ensuring high availability in these environments is crucial for meeting service-level agreements. However, DRAM failures, a primary cause of server downtimes, present significant challenges to reliability, availability, and serviceability. This paper provides an in-depth analysis of memory failure characteristics across cross-architecture platforms in large-scale heterogeneous clusters. We introduce MemSeer, an AIOps-integrated tool that utilizes a multi-grained memory failure prediction approach for x86/ARM heterogeneous clusters. MemSeer improves the F1-score by 17.3% and increases recall by an average of 27% across different lead times compared to state-of-the-art methods. These advancements show great promise in reducing memory failures in cluster environments, decreasing VM interruptions by up to 42.7% and averaging 24.2% in real-world implementations.**

***Index Terms*—DRAM Reliability, Memory Failures, Prediction, Heterogeneous Clusters**

I. Introduction

In the rapid development of high-performance cloud computing, heterogeneous clusters are increasingly used to optimize diverse computing resources and enhance performance and energy efficiency [1]. High Availability (HA) is crucial, with cloud providers setting availability targets in Service Level Agreements (SLAs) [2]. However, hardware failures, especially in Dynamic Random Access Memory (DRAM), jeopardize Reliability, Availability, and Serviceability (RAS) [3], risking data integrity as memory scales and workloads grow. Our research at Huawei shows that memory failures contribute to 57% of server downtimes, as illustrated in Figure 1. To mitigate DRAM errors, chips use Error Correction Code (ECC) methods like Single Bit Error Correction and Double Bit Error Detection Code (SEC/DED) [4], Single Device Data Correction (SDDC) [5], and Chipkill ECC [6]. When errors exceed ECC capabilities, they are categorized as ***Correctable Errors (CEs)*** and ***Uncorrectable Errors (UEs)***, with UEs often leading to data loss or system crashes.

To enhance resilience and reduce memory failures, various fault-tolerance strategies are incorporated into BIOS firmware and AIOps systems [7–10]. Additionally, machine learning (ML)-based memory failure prediction schemes have emerged as a promising solution, leveraging historical error logs to extract CEs and UEs information and offer predictive failure analyses [11, 12]. This enables proactive maintenance

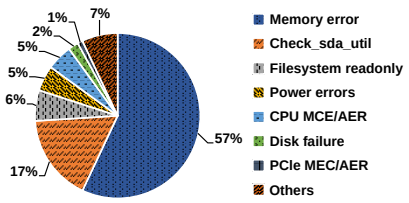


Fig. 1: The breakdown of the causes of server crashes in Huawei clusters.

options such as Dual In-line Memory Module (DIMM) replacement [13, 14], Page Offlining in the operating system by retiring faulty memory regions [15, 16], and migration of Virtual Machines (VMs) across servers before system collapse [17]. Research [14, 18–24] investigates memory faults in cells, rows, and columns to foresee failures. State-of-the-art studies [13–16, 25–27] focus on exploiting more effective indicators for improving memory failure prediction precision and recall. Du *et al.* [14, 25] pioneeringly discuss the bit-level CE patterns and define a new risky CE indicator for the future UEs occurrence prediction. [27] utilizes the concept of CE storms as an important feature to predict the DRAM-caused node unavailability. [26] goes deeper into the bit-level DQ and beats information within the cacheline and presents the hierarchical memory failure prediction (HiMFP) framework and attempts to extend their work across various CPU architectures [28]. However, there are remaining challenges: ❶ Servers with distinct x86 and ARM CPU architectures may exhibit notable differences in DIMM error distribution patterns, affecting predictive performance. ❷ Long-term lead time and fine-grained memory failure predictions perform poorly and offer limited value for actual system fault tolerance.

To address these challenges, we further leverage the spatial and temporal distinctions in memory failure characteristics between x86 and ARM architectures in ultra-scale heterogeneous clusters derived from real-world data centers, including Intel SkyLake, Cascade Lake, and Huawei Kunpeng servers. Subsequently, we introduce MemSeer, a multi-grained memory failure prediction approach that forecasts failures at the DIMM, server, and micro (page and row) levels, showing promise for VM migration and page offlining to prevent system crashes.

Our contributions are summarized as follows:

- We present an in-depth exploration of memory failure logs across ultra-scale heterogeneous clusters, including x86 and ARM architectures, drawing from a dataset comprising about

2 million DIMMs from over 120,000 servers.

- We propose MemSeer and employ corresponding feature engineering to realize the leading multi-grained memory failure prediction, including DIMM-level, server-level and micro-level (i.e., page-level, row-level).
- We conduct comprehensive experiments to evaluate MemSeer's predictive performance and cost benefits from page offlining and live VM migrations based on multi-grained failure prediction. It reduces VM interruptions by 24.2% on average in real-world implementations.

II. Background

A. Memory Organization and Memory Access

The memory system in a server is hierarchical, with DIMMs connected to the CPU through the Integrated Memory Controller (IMC) via channels, as demonstrated in Figure 2. A DIMM consists of multiple DRAM chips organized in ranks for parallelism, with each rank containing 16 data-storing chips and 2 ECC chips. Each chip has multiple banks organized into two-dimensional arrays of cells, indexed by rows and columns, with each cell containing several bits based on the DRAM bit width [21, 25, 26]. In a DDR4 DIMM, data is transmitted through the Data Bus (DQ wires), sending 8 beats of data, including ECC error codes, to the IMC across 18 chips. Each beat encompasses 18 x4 cells (4-bit width, which means the DRAM chip is organized to provide 4 bits of data per access), totaling 72 bits (64 bits for data and 8 for ECC). ECC bits identify error locations in DQs and beats [25, 26]. A typical DDR4 DRAM die (8 Gbit) features banks with 64K rows and 1K columns [21]. Due to Die, Rank, and Channel Interleaving technologies in BIOS [29], a row can accommodate data from 2 to 48 4KB pages by distributing data across multiple channels, ranks, and dies for parallel access.

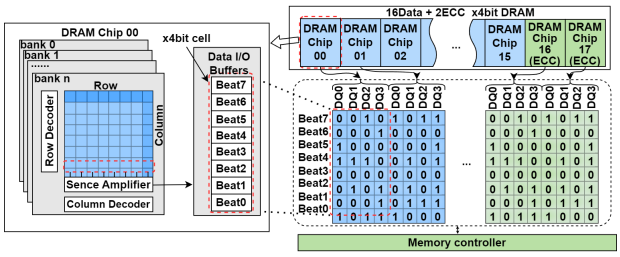


Fig. 2: DIMM components and memory access.

B. Memory Faults, Errors and Failures

Memory Faults refer to the underlying causes of errors, including soft faults and hard faults. **(1)** ***Soft Faults*** occur randomly due to strikes by high-energy particles, cosmic rays, etc. However, they may self-recover during subsequent read or write operations. **(2)** ***Hard Faults*** refer to irreversible damages in hardware. They develop on the micro-level DRAM circuits, with granularity at the level of rows, columns, banks, or ranks.

Memory Error is a general symptom indicating abnormal behavior in memory, representing instances where one or multiple bits in DRAM read do not match the data that was originally stored. From the perspectives of the Operating System (OS) or AIOps, UEs can be categorized as follows: **(1)** ***Critical UE***: Fatal Errors that lead to service interruptions, causing OS kernel panic, process interruptions, and server crashes. **(2)** ***Detected and Unconsumed UE***: Errors detected by the IMC during memory access, which have not yet been loaded into the CPU execution path [10, 30]. **(3)** ***Patrol Scrubbing UE***: Errors discovered through the memory patrol scrubbing mechanism, which periodically initiates memory requests to devices at a predefined frequency, triggering a hardware interrupt and notifying the software layer of the corresponding memory address [10, 31].

III. Observations and Motivation

We analyze memory failure logs from Huawei ultra-scale clusters over three years (2020 Jan. to 2022 Dec.). This dataset includes around 2 million DIMMs from various manufacturers, such as Samsung, Hynix, Micron, Kingston, and Ramaxel. The logs feature timestamps, error types, events, and address details for each CE and UE[1]. We extract bit-level information for each CE from the read retry register[2]. A CE storm suppression mechanism is enabled to mitigate node unresponsiveness from excessive CE interruptions and CPU resource exhaustion. Detailed statistics and error rates are provided in Table I, anonymizing manufacturer information due to NDA policies.

TABLE I: Overview of the cross-arch. DRAM error dataset extracting from real-world ultra-scale heterogeneous clusters.

CPU arch.	DIMM mfr.	DIMMs #	DIMMs w/ CEs	DIMMs w/ CE & UEs	DIMMs w/ UEs[1]	DIMMs w/ UEs in %
x86	A	530,289	6,716	466	502	7.47%
	B	337,856	4,571	560	587	12.84%
	C	355,424	7,786	1043	1,087	13.96%
	D	22,984	416	51	56	13.4%
	Total	1,223,569	19,489	2,069	2,232	11.45%
ARM	A	319,996	6,332	182	305	4.81%
	B	187,908	3,394	79	145	4.27%
	C	244,353	3,774	72	116	3.07%
	Total	752,257	13,500	333	566	4.19%
Total		1,975,826	32,989	2,402	2,788	8.45%

[1] We focus on DIMMs with (denoted as w/) CEs, excluding ***Sudden UEs*** (UEs that occur without any prior CE history). For each DIMM with a UE, we truncate its error log to the first UE occurrence, which is a common method used for preprocessing memory error data [25].

A. Observations

Observation 1: DIMM static properties, capacities and bit-width, have opposite UE rates in x86 and ARM architectural platforms due to distinctions in ECC. We break down the three DIMM specifications' UE rates statistics in Figure 3. On the x86 platform, the UE rates for the three specifications are around 10%. However, on the ARM platform, the failure rate for 8-bit width is 2× to 3× higher than that of 4-bit width. This phenomenon may be attributed to the fact that ARM's SDDC ECC implementation provides near Chipkill-level protection for x4 DIMMs For x8 DIMMs, ARM slightly outperforms x86 by dedicating more bits to ECC verification rather than metadata. This suggests that the static properties are one of the important indicators influencing the predictive performance across x86 and ARM architectural servers.

[1]Logs from ARM servers contain address records, while x86 server logs are excluded due to distinct logging paths in AIOps.

[2]Bit-level data is taken from the `retry_rd_err_log_parity` register in Skylake and Cascade Lake Intel Xeon CPU but is unavailable for Huawei Kunpeng 920 processor due to the lack of these registers.

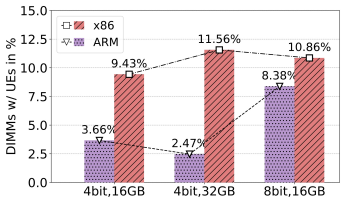


Fig. 3: Ratio of DIMMs w/ UEs in different bit widths and capacity.

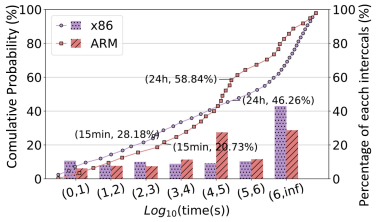


Fig. 4: $CDF(UE)$ and UE probability distribution vs. lead time.

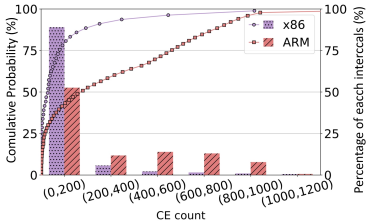


Fig. 5: $CDF(UE)$ and UE probability distribution vs. CE count.

Observation 2: The temporal distribution of CE exhibits significant discrepancies between ARM and x86 platforms. We classify UEs as predictable or unpredictable based on the time interval between the last CE and the first UE occurrence; if this interval exceeds a threshold, the UE is deemed unpredictable. Figure 4 presents the probability distribution and Cumulative Distribution Function (CDF) of predictable UEs across x86 and ARM platforms, with the y-axis showing the proportion of predictable UEs and the x-axis representing the logarithm of prediction lead time. At shorter lead times (up to 6 hours), ARM architecture servers demonstrate slightly higher CDF probability values (2% to 8%) for predictable UEs compared to x86 servers. However, this trend reverses at the longer lead time, with x86 architecture showing a higher proportion of predictable UEs (above 12%) at the 24-hour mark. We conclude that platform differences result in varying predictability of UEs at different lead times, potentially impacting predictive performance.

Observation 3: The spatial distribution of UEs versus CE counts suggests that variations in server CPU architecture influence CE-UE progression and predictability. Figure 5 shows the CDF and probability distribution of DIMMs with CEs leading to UEs in x86 and ARM servers. We observe that on the x86 platform, most UEs occur with few CEs, while the ARM platform shows the opposite trend, where UEs require a significant accumulation of CEs. This phenomenon may result from: varying effectiveness of CE storm suppression; or differing page offline capabilities. This correlation suggests that CE counts may be a potential predictive indicator for UE occurrences on the x86 platform. However, the predictive value of CE count for UE occurrence is substantially diminished in ARM systems.

B. Motivation

❶ Our observations reveal notable differences in memory failure patterns between x86 and ARM platforms, including CE and UE spatial and temporal distributions, progression characteristics, and predictability. This indicates that conventional x86-trained prediction models may underperform on ARM systems. ❷ Memory is crucial for system reliability, and precise predictive performance in long prediction lead time and memory failure granularity is essential in maximizing fault tolerance benefits. We propose MemSeer, a multi-grained memory failure prediction system that distinguishes memory failures in x86/ARM heterogeneous clusters.

IV. Design of MemSeer

The overview of MemSeer is illustrated in Figure 6. It realizes multi-grained memory failure prediction in DIMM, server, and micro (page and row) levels.

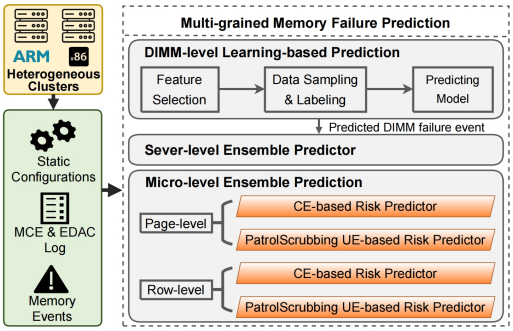


Fig. 6: The overview of MemSeer framework design. The AIOps system collects memory failure logs from various sources, such as configurations, mcelog, and memory logs, and sends them to the data lake.

A. DIMM-level Learning-based Prediction

For DIMM-level prediction, we first preprocess the data by: ❶ Ignoring memory error logs after the first UE in a DIMM, as the DIMM will be replaced and may bias model training [13]. ❷ Excluding DIMMs with ***Sudden UEs***.

1) Feature Engineering: Based on the literature review (see in §.I) and our observations, we extract four feature types for DIMM-level memory failure prediction: Static Features, Spatial Features, Temporal Features, and Bit-Level Error Features.

a) Static Features. These features represent the fixed DIMM characteristics of physical DRAM components.

- **CPU Model**: Different CPU models may use distinct ECC algorithms, fault tolerance technology, and MCE registers.
- **Bit Width**: The number of bits accessible per DRAM access.
- **Capacity**: Memory capacity is pertinent to DRAM density.
- **Rank Count**: The number of ranks in memory impacts the parallelism of the memory system.
- **Vendor**: Different manufacturing processes may involve distinct reliability characteristics.

b) Spatial Features. We design spatial features to capture the spatial distribution characteristics of CEs and UEs in each component at micro-levels.

- **Faulty_Components_Count**(M_i): The count of components M_i with CEs, $M_i \in \{Bank,\ Column,\ Row,\ Cell\}$.
- **CE**$\{max, avg\}$**_Count**(M_i): The maximum and average occurrence count of CEs in each single component M_i, where $M_i \in \{Bank,\ Column,\ Row,\ Cell\}$.
- **Count_Error**(E_i): The count of various errors $E_i \in \{CEs, CEs_withoutAddr$ [3]$, PatrolScrubbing_UEs\}$.

c) Temporal Features. The temporal distribution characteristics of CEs, including the quantity of CEs over a time period t_i and the frequency features of CE occurrences.

[3]CEs reported without physical addresses due to reporting limitations.

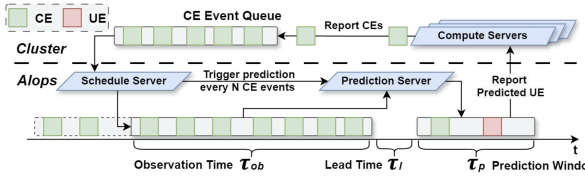


Fig. 7: The logic for DIMM-level memory failure prediction workflow.

- **Min_T_CE**(θ, t_i): The minimum time interval T for θ [4] CEs to appear in t_i [5].
- **CE_Count**(t_i): The quantity of CEs within the time t_i.
- **DIMM_Life_Span**: The time interval between the sampling time and the DIMM's operational timestamp.
- **DIMM_Error_Span**: The time interval between sampling time and first CE occurrence in a DIMM.
- **Error_Avg**: It is equal to the total number of CEs divided by the error span, reflecting the accumulation rate of CEs.

d) Bit-Level Error Features [6]**.** They characterize the distribution of error bits in DRAM DQs and Beats over a 120-hour window (t_i [7]). These features can effectively predict UE occurrences based on memory access principles [26].

- **Distance**$\{min, max\}(B_i, t_i)$: At the bit level, the distance between the two nearest/farthest components B_i that are affected by bit errors, where $B_i \in \{DQs,\ Beats\}$.
- **Bit_Level_Faulty_Count**(B_i, t_i): The number of bit-level faulty components B_i in the occurrence of a bit error, where $B_i \in \{DQs,\ Beats\}$.
- **Adjacent_Error**(B_i, t_i): Record the cumulative CE occurrences for each bit-level component $B_i \in \{DQs,\ Beats\}$.

2) Data Sampling, Labeling and Prediction: Observations 2 and 3 in §.III-A show fewer CE reports and a sparse distribution. We propose a CE queue event-driven sampling and prediction scheme (see Figure 7) instead of fixed time windows. A message queue in the computing cluster collects CE occurrences, which are uploaded to the Schedule Server to update the CE dataset in the observation window of τ_{ob} (6, 72, or 120 hours). When reported CEs for a DIMM reach the threshold N, the data from its observation window will be sent for UE prediction. We also consider lead time (τ_l) and prediction window (τ_p) for labeling DIMM samples as follows:

- **Positive Sample**: If the occurrence time t_{UE} of a UE falls within the sum of the time interval of the observation window and lead time, i.e., $t_{UE} \in [t_{CE_i} + \tau_l,\ t_{CE_i} + \tau_l + \tau_p]$.
- **Negative Sample**: If no UE happens within $[t_{CE_i} + \tau_l,\ t_{CE_i} + \tau_l + \tau_p]$, this sample is labeled as a negative sample.

In our experiments, DIMM-level prediction with $\tau_{ob} = 120$ hours and $\tau_p = 30$ days brings the best performance.

3) Prediction Model: DIMM-level prediction is a binary classification problem. We leverage three tree-based ML approaches, including Extreme Gradient Boosting (XGBoost) [32], Random Forest (RF) [33], and Light Gradient-Boosting Machine (LightGBM) [34]. We also investigate three neural network-based ML methods, including Self-Attention Long-Short Term Memory neural network (SA-LSTM) [35], Fully-connected Neural Network (FCN) [36], and Recurrent Neural Network (RNN) [37]. These algorithms can capture complex feature correlations and are easy to implement in real-world clusters due to their lightweight footprint. We compare their predictive performance in §.V-B1 and select XGBoost as the DIMM-level predictor due to its best experimental results.

[4] θ is the number of CEs and is set to 3, 5, and 7.
[5] For temporal features, t_i is set to 6 min, 6, 24, 72, and 120 hours.
[6] Available only on x86 architecture platforms (explained in §.III).
[7] For bit-level error features, we set t_i to 120 hours

Algorithm 1: Micro-level Ensemble Prediction

```
/* Rule-based Page-level Risk Predictor (CP)   */
Input: The number of CEs that occur in Cell i: CE_cnt(Cell_i); Threshold of
       a cell experiencing CE occurrences: X;
Output: CE-based Risky Pages
for each page Page_j do
    for each cell Cell_i in Page_j do
        if CE_cnt(Cell_i) ≥ X then
            Return Page_j as a CE-based Risky Page;

/* PSUE-based Page-level risk Predictor (PP)   */
Input: PatrolScrubbing UE (PSUE) count in Cell i: PSUE_cnt(Cell_i);
Output: PatrolScrubbing-based Risky Pages
for each page Page_j do
    for each cell Cell_i in Page_j do
        if PSUE_cnt(Cell_i) ≠ 0 then
            Return Page_j as a PatrolScrubbing-based Risky Page;

/* CE-based Row-level risk Predictor (CR)      */
Input: The number of cells in Row i that experience CE:
       Cell_CE_cnt(Row_i)); The furthest distance between the first and the
       last CEs in Row i: Len_CE(Row_i); Thresholds θ_Cell and θ_Len;
Output: CE-based Risky Rows
for each row Row_i do
    if Cell_CE_cnt(Row_i) ≥ θ_Cell and Len_CE(Row_i) ≥ θ_Len then
        Return Row_i as a CE-based Risky Row;

/* PSUE-based Row-level risk predictor (PR)    */
Input: the frequency of cells that occur PatrolScrubbing UE in Row i:
       Cell_PSUE_cnt; Thresholds θ_Cell and θ_Len;
Output: PatrolScrubbing-based risky row
for each row Row_i do
    if Cell_PSUE_cnt(Row_i) ≥ 2 then
        Return Row_i as a PatrolScrubbing-based Risky Row;
```

B. Server-level Ensemble Prediction

We estimate the likelihood of UEs in DIMMs within a specific lead time. Based on prior research [26, 38], we assume that UEs in server DIMMs are independent and identically distributed (i.i.d.). To derive server-level forecasts from DIMM predictions, we apply the At Least One Rule, using a logical OR on the binary failure predictions (0: Negative, 1: Positive) for all DIMMs, as shown in Equation 1. Here, $C(S_i)$ represents the predicted output for server i, and $C(D_{ij})$ indicates the prediction for DIMM j in server i, with $j \in \{1, \ldots, N\}$ and N being the total DIMMs in server i. A server-level memory failure prediction triggers VM migration to avoid downtime, requiring at least 5 minutes for log transfer and migration [26, 27].

$$C(S_i) = 1 - \prod_{j=1}^{N} (1 - C(D_{ij})) \tag{1}$$

C. Micro-level Ensemble Prediction

Unlike DIMM-level prediction, learning methods are ineffective for micro-level predictions. Therefore, micro-level prediction considers all CEs and addressable Patrol Scrubbing UEs for identifying risks at the page and row levels, using ensemble predictions based on threshold rules. We design four predictors in Algorithm 1, detailed as follows:

1) Rule-based Page-level Risk Identification: Classical page-level prediction uses the $X/24$ page offlining policy [39], identifying pages with more than X faults in 24 hours for offline status. It is suggested that repeated CE occurrences in the same cell pose a high risk in correlating with UE events [27, 40]. Additionally, Patrol Scrubbing UEs notably increase the risk of subsequent UEs in the same cell. Consequently, we incorporate both CEs and Patrol Scrubbing UEs in our page-level risk identification design (Algorithm 1 lines 1 to 8).

2) Rule-based Row-level Risk Identification: Based on the literature [15, 16], CE can predict UE occurrences within the same column, and there's a correlation between UE and CE events in the same row. The row-level predictor in [16] uses CE frequency, CE distribution, and bit-level error features for identifying high-risk rows. Lacking bit-level logs in our ARM platform data (see in §.III), we focus on CE counts and the spatial distribution of CE and UEs at the row-level predictor (Algorithm 1 lines 9 to 14).

3) Micro-level Risk Ensemble Prediction: We propose page and row-level rule-based predictors, excluding others due to high costs in memory failure policies. To minimize costs, we optimize ensemble mechanisms for efficient UE prediction. The hyperparameters are calibrated to $X = 4$, $\theta_{Cell} = 5$, and $\theta_{Len} = 64$ (assuming a row length of 1024). The evaluation of micro-level risk prediction performance is discussed in §.V-B3.

V. Evaluation

A. Experiment Setup

1) Dataset Description: We replay historical logs from the dataset explored in §.III to simulate an online production environment, with the dataset split by server architecture. We manipulate labels to evaluate DIMM, micro, and server-level predictions. The relationships among datasets, their utilization, and evaluation metrics are detailed in Table II.

TABLE II: Utilization of dataset.

Datasets	Bit-level logs	UE Addr	Utilization of Datasets	Evaluation Metrics
Dataset1 (x86)	✓	✗	DIMM-level prediction	R, P, F1
			Server-level prediction	R, P, F1, VMIRR
Dataset2 (ARM)	✗	✓	DIMM-level UE prediction	R, P, F1
			Micro-level prediction	Poff cost per UE
			Server-level prediction	R, P, F1, VMIRR

2) Evaluation Metrics: To comprehensively evaluate our proposed approach, we employ three categories of metrics:

a) Predictive Measures: We utilize standard classification metrics to assess the predictive performance:

- **Precision** = $\frac{TP}{TP+FP}$, the higher the better.
- **Recall** = $\frac{TP}{TP+FN}$, the higher the better.
- **F1-score** = $\frac{2\times Precision\times Recall}{Precision+Recall}$, the higher the better.

TP (True Positives): correctly predicted failures. FP (False Positives): false alarms. FN (False Negatives): missed failures.

b) VM Interruption Reduction Rate (VMIRR): VMIRR is a popular metric to quantify the effectiveness of server-level memory failure prediction by live VM migrations [26, 27]. V_a is the average number of VMs per server. p_c is the proportion of cold VM migrations due to alarms and resource unavailability, and the number of VM interruptions with UE prediction is $p_c \cdot V_a(TP+FP) + V_aFN$.

- **VMIRR** = $\frac{V_a(TP+FN)-(p_c\cdot V_a(TP+FP)+V_a\cdot FN)}{V_a\cdot(TP+FN)}$ $= (1-\frac{p_c}{Precision})\cdot Recall$ (the higher the better).

c) Page Offlining Measures: Based on established literature [15, 16, 19, 39–41], we evaluate the effectiveness of page offlining using two key metrics:

- **Avoided UEs**: Quantifies the number of prevented UEs.
- **Page offling (Poff) per UE**: Measures the memory capacity overhead required to prevent each UE.

B. Experimental Results

We show the experimental results to comprehensively evaluate the proposed MemSeer's ❶ prediction performance (§.V-B1.a) and §.V-B1.b), ❷ feature importance (§.V-B1.c) and lead time (§.V-B1.d) sensitivity analyses, and ❸ the fault tolerance value brought by multi-grained failure prediction (§.V-B2 and §.V-B3).

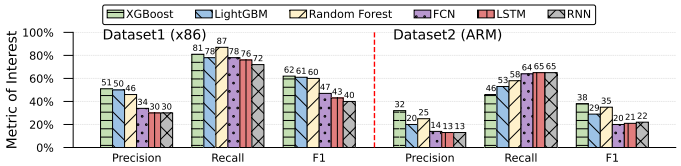


Fig. 8: Different model attempts in MemSeer for DIMM-level prediction.

1) DIMM-level Prediction Evaluation: ***a) Comparison of various prediction models:*** We compare various models discussed in §.IV-A3 for the DIMM-level predictor. Figure 8 shows the predictive performance of models on x86 and ARM architectures with a 1-second lead time and a 0.5 probability threshold. XGBoost leads with a precision of 0.51, recall of 0.81, and F1-score of 0.62 on dataset 1, while on dataset 2, it scores 0.32, 0.46, and 0.38, respectively. LightGBM achieves the highest recall of 0.87 and 0.58 for x86 and ARM datasets. Tree-based models consistently outperform NN models by 12% to 20% in precision and F1-score, though differences among models within each category are under 5%. The predictive performance of each model on x86 and ARM datasets has decreased by around 25% in F1-score, due to the absence of bit-level CE logs in ARM architecture, which has a non-negligible impact on predictive performance.

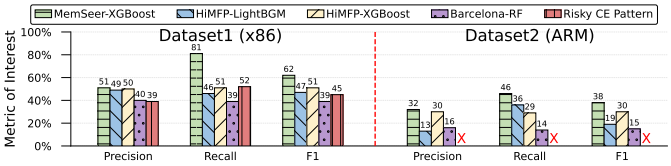


Fig. 9: Performance comparison with existing approaches.

b) Comparison with existing approaches: To evaluate DIMM-level UE prediction, we reproduce leading methods including Barcelona-Random Forest (RF) [13], HiMFP-lightGBM [26], and Risky_CE Pattern [25], fine-tuning their best models on our dataset from ultra-scale heterogeneous clusters. We compare MemSeer-XGBoost against these methods using metrics from our x86 and ARM datasets. Results in Figure 9 show XGBoost surpassing LightGBM, the best in HiMFP [26], in our heterogeneous clusters. Our method excels on both platforms, improving F1-score by 11% to 30% on dataset 1 and 8% to 20% on dataset 2. In dataset 1 (x86), our recall is at least 30% higher, while in dataset 2 (ARM), our F1-score is over 8% better, with HiMFP showing minimal difference. Risky_CE cannot be applied to ARM due to the lack of bit-level data.

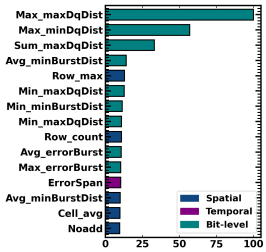


Fig. 10: Feature importance of x86 datase

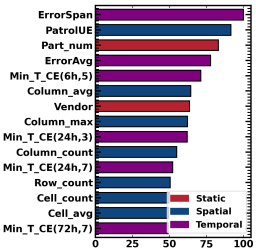


Fig. 11: Feature importance of ARM dataset

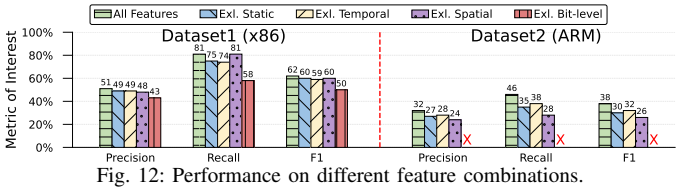


Fig. 12: Performance on different feature combinations.

c) Sensitivity I: Feature Importance Analysis and Ablation Experiments: We conduct an ablation study to evaluate the importance of static, spatial, temporal, and bit-level features in our proposed model, MemSeer-XGBoost. Results are shown in Figure 12, and feature importance is illustrated in Figures 10 and 11 for x86 and ARM datasets, respectively. The results from the evaluation of the x86 dataset align with the findings of the previous study [26]. Bit-level features demonstrate the highest importance, with bit-level features having the highest importance, leading to decreases in precision, recall, and F1-score by 8%, 23%, and 12% when excluded. In contrast, the ARM dataset exhibits a more even distribution of importance scores among feature types, with spatial features showing noticeable effects, resulting in precision, recall, and F1-score reductions of 8%, 18%, and 8%, respectively, when excluded.

d) Sensitivity II: Lead time. In Figure 13, we assess our method's performance in predicting DIMM-level memory failures across seven lead times based on the memory recovery strategy. We analyze page offlining within 1 minute, VM migration from 2 to 5 minutes, and an optimal 5-minute VM migration to mitigate data loss and maintain SLA during system crashes. Even with heavy workloads, a 1-hour lead time allows for manual diagnosis and DIMM replacement. Our results show that as lead time increases, all metrics decline, particularly recall, which drops by 25% and 10% for x86 and ARM, respectively. With a 6-hour lead time, we achieve 0.43 precision and 0.56 recall for x86, and 0.27 precision and 0.36 recall for ARM, resulting in F1-scores of 0.48 and 0.31. The increase in lead time hampers the model's ability to capture temporal dependencies, reducing predictive accuracy. These findings highlight the implications of varying lead times for memory failure tolerance in real-world applications.

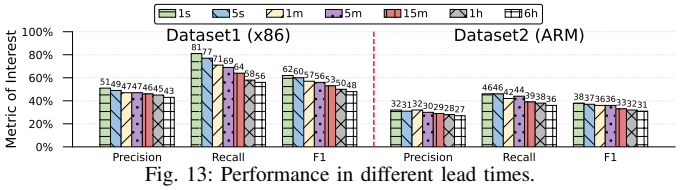


Fig. 13: Performance in different lead times.

2) Server-level Prediction and VMIRR Evaluation: In this prediction (introduced in §.IV-B), TP indicates correct predictions of servers with UEs, FP is when a normal server is incorrectly predicted to have UEs, and FN is when a UE occurs but is not predicted. We emulate VM live migration times across different loads with lead times of 5 minutes, 15 minutes, and 1 hour. We also analyze UE variations at these lead times, as shown in Figure 4. Table III presents our server-level prediction performance and VMIRR on x86 and ARM platforms. For x86, we achieve a VMIRR of 40.2% at a 5-minute lead time, with heavy VM loads reducing migrations by 28.1%. ARM platforms reach a VMIRR of 20.7% at best. Compared to HiMFP-XGBoost, MemSeer improves VMIRR by 15.2% to 34.9% on x86 and about 1.3× on ARM.

TABLE III: Performance of server-level prediction and benefits of live VM migration on x86 and ARM platforms in Precision / Recall / F1-score / VMIRR.

Dataset	Lead time	MemSeer-XGBoost	HiMFP-XGBoost
Dataset1 (x86)	5min	**0.41 / 0.75 / 0.53 / 40.2%**	0.40 / 0.57 / 0.47/ 29.8%
	15min	0.39 / 0.70/ 0.50 / 34.7%	0.37 / 0.61 / 0.46 / 28.6%
	1h	0.36 / 0.64 / 0.46 / 28.1%	0.36 / 0.56 / 0.44 / 24.4%
Dataset2 (ARM)	5min	**0.24 / 0.65 / 0.35 / 20.7%**	0.24 / 0.26 / 0.25 / 8.2%
	15min	0.23 / 0.58 / 0.33 / 15.5%	0.24 / 0.24 / 0.24 / 6.8%
	1h	0.22 / 0.53 / 0.31 / 12.3%	0.23 / 0.21 / 0.22 / 5.4%

TABLE IV: Average numerical results of comparative micro-level prediction-guided page offline policies. We design four selected micro-level predictor ensembles (MemSeer-Comb1 to MemSeer-Comb4) using different combinations of CR, CP, PR and PP predictors outlined in Algorithm 1 in §.IV-C, where a '✓' indicates the inclusion of a predictor and an 'x' indicates its exclusion.

Method	Multi-level Ensemble				Page Offlining Measures		
	CR	CP	PR	PP	Cost (pages)	Avoid UEs	Cost/UE (pages)
CR[16]	✓	×	×	×	10,452	472	23
CP[39]	×	✓	×	×	59,306	389	153
PR	×	×	✓	×	1,714	745	3
PP	×	×	×	✓	10,104	819	13
MemSeer-Comb1	×	×	✓	✓	10,317	840	13
MemSeer-Comb2	×	✓	✓	✓	68,722	962	72
MemSeer-Comb3	✓	×	✓	✓	20,329	994	21
MemSeer-Comb4	✓	✓	✓	✓	73,372	1,050	70

3) Micro-level Prediction and Page Offline Evaluation: We analyze micro-level page offlining benefits using Dataset 2 (ARM), as it alone has UE address logs for micro-level failure prediction. We replicate state-of-the-art OS page offlining policies [16, 39], which are also included in our approach (referred to as CR and CP). Results are shown in Table IV. CP's cost per UE is significantly higher than the baseline, but the cost per UE for PR and PP remains lower than the baseline. Therefore, in scenarios where the priority is on the number of avoided UEs, we choose method MemSeer-Comb3, which includes three predictors outside of CP; compared to the baseline, avoided UEs increased by 110.6%, and the cost per UE is 92.4% of the original. In scenarios where precision is prioritized, we choose method MemSeer-Comb1, which improves recall by 78.0% compared to the baseline, and the cost per UE is 55.5% of the baseline.

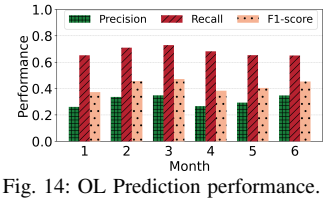


Fig. 14: OL Prediction performance.

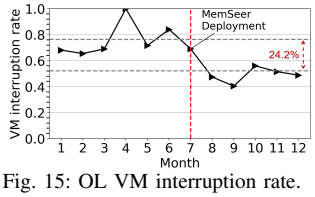


Fig. 15: OL VM interruption rate.

4) Online (OL) Results: MemSeer has been used in the online real-world heterogeneous cluster for over six months, enhancing service availability. The online precision and recall of MemSeer are shown in Figure 14. Moreover, Figure 15 illustrates the VM interruption rate before and after MemSeer's

deployment. The Y-axis is relative due to confidentiality. The interruption rate decreased by an average of 24.2%.

VI. Conclusion

This paper first explores the memory failure distinctions in ultra-scale x86 and ARM heterogeneous clusters, introducing MemSeer for multi-grained memory failure prediction. It achieves 51% precision and 81% recall for x86, and 32% precision and 46% recall for ARM, reducing VM interruptions by 40.2% and 20.7%, respectively. It also significantly enhances page offlining. MemSeer online implementation in real products reduces VM interruptions by an average 24.2% in clusters.

VII. Acknowledgments

We thank all the anonymous reviewers for their valuable comments. This work was supported in part by the National Key R&D Program of China (Grant No. 2023YFB4502900).

References

[1] J. Jiang *et al.*, "Cronus: Fault-isolated, secure and high-performance heterogeneous computing for trusted execution environment," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 124–143.

[2] P. Patel *et al.*, "Service level agreement in cloud computing," 2009.

[3] A. Yao *et al.*, "A memory ras system design and engineering practice in high temperature ambient data center," in *2020 19th IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm)*. IEEE, 2020, pp. 1379–1388.

[4] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based sec-ded-daec code," in *25th IEEE VLSI Test Symposium (VTS'07)*. IEEE, 2007, pp. 349–354.

[5] A. Note, "Intel® e7500 chipset mch intel® x4 single device data correction (x4 sddc) implementation and validation," 2002.

[6] X. Jian *et al.*, "Low-power, low-storage-overhead chipkill correct via multi-line error correction," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–12.

[7] Y. Luo *et al.*, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014, pp. 467–478.

[8] H. Dong *et al.*, "Memvisor: application level memory mirroring via binary translation," in *2012 IEEE International Conference on Cluster Computing*. IEEE, 2012, pp. 562–565.

[9] X. Guo *et al.*, "Sanitizer: Mitigating the impact of expensive ecc checks on stt-mram based main memories," *IEEE Transactions on Computers*, vol. 67, no. 6, pp. 847–860, 2017.

[10] S. Scargall and S. Scargall, "Reliability, availability, and serviceability (ras)," *Programming Persistent Memory: A Comprehensive Guide for Developers*, pp. 333–346, 2020.

[11] I. Giurgiu *et al.*, "Predicting dram reliability in the field with machine learning," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*, 2017, pp. 15–21.

[12] P. Nikolaou *et al.*, "Modeling the implications of dram failures and protection techniques on datacenter tco," in *Proceedings of the 48th International Symposium on Microarchitecture*, 2015, pp. 572–584.

[13] I. Boixaderas *et al.*, "Cost-aware prediction of uncorrected dram errors in the field," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[14] X. Du and C. Li, "Predicting uncorrectable memory errors from the correctable error history: No free predictors in the field," in *The International Symposium on Memory Systems*, 2021, pp. 1–10.

[15] X. Du and C. Li, "Combining error statistics with failure prediction in memory page offlining," in *Proceedings of the International Symposium on Memory Systems*, 2019, pp. 127–132.

[16] X. Du *et al.*, "Fault-aware prediction-guided page offlining for uncorrectable memory error prevention," in *2021 IEEE 39th International Conference on Computer Design (ICCD)*. IEEE, 2021, pp. 456–463.

[17] F. Zhang *et al.*, "A survey on virtual machine migration: Challenges, techniques, and open issues," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018.

[18] X. Du and C. Li, "Memory failure prediction using online learning," in *Proceedings of the International Symposium on Memory Systems*, 2018, pp. 38–49.

[19] X. Du *et al.*, "Predicting uncorrectable memory errors for proactive replacement: An empirical study on large-scale field data," in *2020 16th European Dependable Computing Conference (EDCC)*. IEEE, 2020, pp. 41–46.

[20] F. Yu *et al.*, "Dram failure prediction in large-scale data centers," in *2021 IEEE International Conference on Joint Cloud Computing (JCC)*. IEEE, 2021, pp. 1–8.

[21] Z. Cheng *et al.*, "An in-depth correlative study between dram errors and server failures in production data centers," in *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2022, pp. 262–272.

[22] J. Bogatinovski *et al.*, "First ce matters: On the importance of long term properties on memory failure prediction," in *2022 IEEE International Conference on Big Data (Big Data)*. IEEE, 2022, pp. 4733–4736.

[23] L. Mukhanov *et al.*, "Workload-aware dram error prediction using machine learning," in *2019 IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2019, pp. 106–118.

[24] X. Wang *et al.*, "On workload-aware dram failure prediction in large-scale data centers," in *2021 IEEE 39th VLSI Test Symposium (VTS)*. IEEE, 2021, pp. 1–6.

[25] C. Li *et al.*, "From correctable memory errors to uncorrectable memory errors: what error bits tell," in *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2022, pp. 01–14.

[26] Q. Yu *et al.*, "Himfp: Hierarchical intelligent memory failure prediction for cloud service reliability," in *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2023, pp. 216–228.

[27] P. Zhang *et al.*, "Predicting dram-caused node unavailability in hyperscale clouds," in *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2022, pp. 275–286.

[28] Q. Yu *et al.*, "Investigating memory failure prediction across cpu architectures," 2024. [Online]. Available: https://arxiv.org/abs/2406.05354

[29] M. Hillenbrand *et al.*, "Multiple physical mappings: Dynamic dram channel sharing and partitioning," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, 2017, pp. 1–9.

[30] D. Technologies, "Memory errors and dell emc poweredge yx4x server memory ras features," Tech. Rep., 2020. [Online]. Available: https://dl.dell.com/manuals/common/dellemc_poweredge_yx4x_memoryras_v1_1.pdf

[31] Y. Han *et al.*, "Data-aware dram refresh to squeeze the margin of retention time in hybrid memory cube," in *2014 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2014, pp. 295–300.

[32] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[33] S. J. Rigatti, "Random forest," *Journal of Insurance Medicine*, vol. 47, no. 1, pp. 31–39, 2017.

[34] G. Ke *et al.*, "Lightgbm: A highly efficient gradient boosting decision tree," *Advances in neural information processing systems*, vol. 30, 2017.

[35] W. Li *et al.*, "Bidirectional lstm with self-attention mechanism and multi-channel features for sentiment classification," *Neurocomputing*, vol. 387, pp. 63–77, 2020.

[36] T. N. Sainath *et al.*, "Convolutional, long short-term memory, fully connected deep neural networks," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. Ieee, 2015, pp. 4580–4584.

[37] L. R. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, no. 64-67, p. 2, 2001.

[38] B. Schroeder *et al.*, "Dram errors in the wild: a large-scale field study," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 1, pp. 193–204, 2009.

[39] C. H. Costa *et al.*, "A system software approach to proactive memory-error avoidance," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2014, pp. 707–718.

[40] A. A. Hwang *et al.*, "Cosmic rays don't strike twice: Understanding the nature of dram errors and the implications for system design," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 111–122, 2012.

[41] J. Meza *et al.*, "Revisiting memory errors in large-scale production data centers: Analysis and modeling of new trends from the field," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2015, pp. 415–426.

Enabling On-Tiny-Device Model Personalization via Gradient Condensing and Alternant Partial Update

Zhenge Jia, Yiyang Shi, Zeyu Bao, Zirui Wang, Xin Pang, Huiguo Liu, Yu Duan, Zhaoyan Shen*, Mengying Zhao
School of and Computer Science and Technology, Shandong University, China
Corresponding Author Email: *shenzhaoyan@sdu.edu.cn

***Abstract*—On-device training enables the model to adapt to user-specific data by fine-tuning a pre-trained model locally. As embedded devices become ubiquitous, on-device training is increasingly essential since users can benefit from the personalized model without transmitting data and model parameters to the server. Despite significant efforts toward efficient training, on-device training still faces a major challenge: The prohibitive cost of multi-layer backpropagation strains the limited resources of tiny devices. In this paper, we propose an algorithm-system co-optimization framework *TinyMP* that enables self-adaptive on-tiny-device model personalization. To mitigate backpropagation costs, we introduce *Gradient Condensing* to condense the gradient map structure, significantly reducing the computational complexity and memory consumption of backpropagation while preserving model performance. To further reduce computation overhead, we propose *Alternant Partial Update*, a mechanism that locally and alternatively selects essential parameters to update without requiring retraining or offline evolutionary search. Our framework is evaluated through extensive experiments using various CNN models (e.g., MobileNetV2, MCUNet) on embedded devices with minimal resources (e.g., OpenMV-H7 with less than 1MB SRAM and 2MB Flash). Experimental results show that our framework achieves up to 2.4× speedup, 80.8% memory saving, and 30.3% accuracy improvement on downstream tasks, outperforming SOTA approaches.**

I. Introduction

To date, most ultra-low power deep learning applications at the edge are still trained "off device" (typically on the cloud/edge server) while the tiny edge device performs the inference. The real-world applications always come with dynamic scenarios (e.g., personalized data features) leading to significant accuracy drops. To address the issue, tiny edge devices need to transmit model parameters with user-specific data to the server to personalize the model to achieve satisfying performances. Data uploading jeopardizes privacy over sensitive information and high-volume transmission leads to a severe battery lifetime degradation of the edge device.

On-device training enables the edge device to fully exploit its computing power to perform local adaptation after deployment. By fine-tuning locally, the deep neural network (DNN) can learn to dynamically adjust to new tasks or shifting data distributions over time. This approach is particularly advantageous for ultra-low-power deep learning applications, where personalization tasks are relatively lightweight [1], [2]. In such scenarios, only a small subset of model parameters needs to be updated within just a few training epochs to achieve satisfactory performance [3]. Compared with the conventional server-based personalization approaches, on-device training eliminates the need for high-energy and potentially privacy-compromising data transmission. This makes it a compelling solution for applications where efficiency, privacy, and adaptability are paramount [4].

However, on-device training on tiny devices presents significant challenges and fundamentally differs from training in cloud or edge environments [5], [6]. Tiny devices such as microcontrollers are typically constrained by limited computing power (e.g., 1.25 DMIPS/MHz for Cortex-M4 [7]) and memory capacity (often under 1MB SRAM and 2MB Flash [8], [9]). These resource limitations make even on-device inference difficult to sustain. The challenges are further exacerbated during training, which requires additional computational and memory resources for backpropagation. Moreover, existing deep learning frameworks for tiny devices (e.g., TensorFlow Micro [10] and X-Cube-AI [11]) are primarily optimized for inference tasks rather than training.

To enable on-tiny-device training, the fundamental approach is to adopt resource-efficient neural network architecture such as MobileNetV2 [12] and MCUNet [3] with less trainable parameters, which directly cut down memory and computation costs. Based on the efficient network architecture, efforts are made from two perspectives: Partial Update (PU) and Efficient Backpropagation (EB). PU reduces memory usage (e.g., activation storage) and computational costs (e.g., gradient calculation) by selectively updating critical parts of the network. The attempts in PU include updating the linear layers of DNN [13], training the selected parameters based on offline or online searching [1], [14], [15], updating bias alone [16], etc. EB focuses on optimizing the backpropagation computation process to lower resource costs. The attempts include gradient quantization to save memory [17] and gradient map pooling to reduce the number of operations [18], etc.

Despite efforts to enable on-device training, three key challenges impede the practical adoption of model personalization on tiny devices: (1) Identifying important model parameters using searching-based methods often imposes a computational workload that exceeds the capabilities of tiny devices; (2) Parameter importance is dynamic, and static partial update strategies risk overfitting as they fail to adapt to the changing parameter learning state over time; (3) Efficient backpropagation often compromises gradient precision to reduce resource usage, leading to significant accuracy degradation.

To address these challenges, we propose *TinyMP*, an algorithm-system co-optimization framework that enables self-adaptive on-tiny-device model personalization entirely on tiny devices. TinyMP empowers tiny devices to perform self-calibrating important parameter selection and efficient model personalization without gradient precision compromise.

Specifically, we propose Gradient Condensing (GC) to approximate the gradient w.r.t. the kernel size by creating a condensed gradient map with fewer elements. This reduces the number of operations required for gradient calculation through convolutional layers while minimizing the memory needed for forward activations. Notably, the approximated gradients are bounded and maintain high precision. To further optimize computational and memory efficiency, we introduce Alternant Partial Update (APU), which dynamically identifies important parameters by periodically measuring parameter importance at runtime, skipping gradient computation for less critical parameters. We evaluate TinyMP using various CNN models (e.g., MobileNet, MCUNet) deployed on tiny devices (e.g., microcontrollers), adapting to a wide range of downstream tasks. Experimental results show that TinyMP achieves up to 2.4× speedup, 80.8% memory saving, and 30.3% accuracy improvement over downstream tasks, outperforming SOTA approaches. Our contributions are summarized as follows:

- We propose an algorithm-system co-optimization framework TinyMP that enables deep model personalization entirely on tiny devices.
- We devise Gradient Condensing (GC), a mechanism that minimizes the computational overhead and memory usage of backpropagation while maintaining gradient precision.
- We introduce Alternant Partial Update (APU), a runtime mechanism for identifying important parameters to update, which reduces computational overhead and mitigates overfitting.
- TinyMP outperforms SOTA approaches across various downstream tasks, delivering superior accuracy, reduced training latency, and lower memory overhead on tiny embedded devices.

II. Background and Motivation

To date, the standard approach for ultra-low-power deep learning applications involves training a model offline, compressing it, and deploying it to the device for inference only. However, this approach often yields poor performance due to discrepancies between real-world data distributions and the training data. To address this, model parameters and user-specific data are typically uploaded to edge or cloud servers for model updates, after which the personalized model is redeployed to the device [17], [19]. However, this process must often be repeated in dynamic application scenarios, leading to excessive energy consumption, long execution delays, and potential privacy risks, which are particularly problematic for resource-constrained embedded devices [18].

On the other hand, ultra-low-power deep learning applications generally involve relatively simple tasks, where model personalization can achieve satisfactory performance without requiring extensive user-specific data or training epochs. In this context, fully leveraging the computational capabilities of tiny devices to perform efficient on-device training offers a more practical and effective solution. This approach enables on-demand, privacy-preserving model personalization while eliminating the drawbacks of conventional server-device communication. The existing efforts toward on-device training can be summarized into two categories:

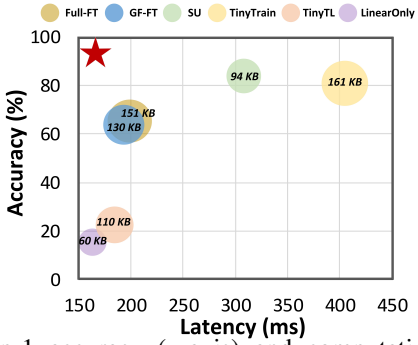


Fig. 1: Top-1 accuracy (y-axis) and computation overhead (x-axis) of existing on-device training methods deployed on practical MCU platforms. The radius of the circles illustrates the peak memory usage. Detailed setup is in Section IV-A.

Partial Update: Based on the efficient model architecture, updating the partial network is widely adopted to reduce the memory footprint of backpropagation. [19] fine-tunes the model with a static number of the last few layers, denoted as *Full-FT*. Authors in [13] propose a method (*LinearOnly*) to only update the linear layers of the model. Authors in [16] propose *TinyTL* to update only bias and the attached branches of the neural network. [1] proposes Sparse Update (*SU*) based on an offline evolutionary search to determine the layers and channels to be updated. However, these approaches require iterative training with extensive user-specific data on the server to reach satisfying performance, which is not capable to dynamic application scenarios. Another solution is to perform online important parameter searching. [14] propose a dynamic neuron searching strategy based on the difference changes of the corresponding activations. [15] propose *TinyTrain*, an online and multi-objective on-device searching method by dynamically setting importance score on channels based on the product of the corresponding gradient and activations. Though they claim to be online searching approaches, they are still far beyond practical deployment on tiny devices. The storage of activation and gradient for searching can easily exceed the capacity of the tiny devices, and the importance computation incurs a long delay.

Efficient Backpropagation: A more fine-grained approach is to optimize the gradient computation process in backpropagation. Authors in [17], [20] quantize gradient after backpropagation to reduce memory usage while works in [21] reduce the cost of each arithmetic operation. However, these approaches only focus on compressing gradient maps while unable to reduce the number of operations. [18] proposes a gradient filtering based fine-tuning approach (denoted as *GF-FT*) to reduce the number of computations and memory consumption by approximating the gradient through max-pooling the gradient map and corresponding filter and activations. However, these approaches inevitably result in an accuracy drop due to the lower precision of the gradient.

Fig. 1 compares the Top-1 classification accuracy (y-axis)

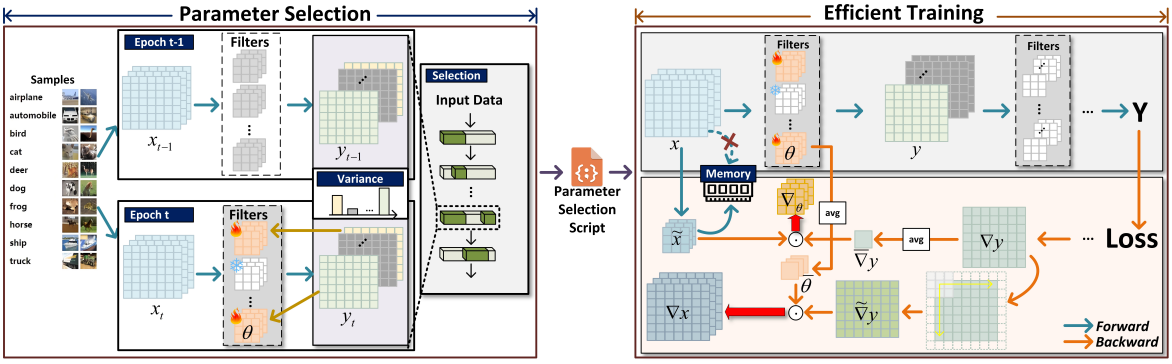


Fig. 2: An overview of TinyMP framework: (a) Parameter Selection: Determine important parameters via Alternant Partial Update (APU); (b) Efficient Training: Memory- and computation-efficient backpropagation via Gradient Condensing (GC).

and computational overhead in terms of latency (x-axis) for various on-device training methods implemented on a microcontroller (MCU) platform, with circle sizes representing peak memory usage (detailed setup in Section IV-A). Existing methods demonstrate significant trade-offs: approaches like *TinyTL*, *LinearOnly*, and *GF-FT* achieve shorter latencies but suffer from low accuracy, while methods such as *TinyTrain* and *SU* deliver higher accuracy at the expense of substantial latency and memory consumption. The red star indicates the desired optimization goal of achieving high accuracy, low latency, and minimal memory usage, which existing methods fail to balance effectively. This highlights the need for a novel framework that enables efficient online parameter searching and lightweight backpropagation to achieve real-time model personalization on highly constrained devices such as MCU.

III. On-Tiny-Device Model Personalization

A. System Overview

TinyMP consists of two modules, Parameter Selection (PS) and Efficient Training (ET), enabling on-tiny-device model personalization shown in Fig. 2. Alternant Partial Update (APU) is proposed for the PS module to enable online important parameter searching and alternant update. Gradient Condensing (GC) is proposed in the ET module to conduct memory- and computation-efficient backpropagation with a bounded approximated gradient.

B. Gradient Condensing

We first introduce GC, which targets the most computation- and memory-heavy operation, *i.e.,* convolution.

Problem Setup: It starts with considering a neural network stacking N convolutional layers. In the *forward propagation*, for a convolution layer r, the input $x \in \mathbb{R}^{C_{in,r}\times H\times W}$ convolves with the filters $\theta \in \mathbb{R}^{C_{out,r}\times C_{in,z}\times K_H\times K_W}$ and return the output $y \in \mathbb{R}^{C_{out,z}\times H'\times W'}$, where H' and W' are calculated based on the convolution stride and padding. The output y will be utilized as the input feature map to the next convolution layer until the loss L is obtained. As for *backward propagation*, for a convolution layer r, we first need to obtain the gradient of the loss w.r.t the output (i.e., $\nabla_y = \frac{\partial L}{\partial y}$), which is typically computed in the subsequent layer. The gradient of the loss w.r.t. the input (i.e., $\nabla_x = \frac{\partial L}{\partial x}$) is computed by convolving g_y with the rotated convolutional filters θ^{rot} (i.e., $\nabla_x = \nabla_y \circledast \theta^{rot}$). The gradient of the loss w.r.t. the convolution filters (i.e., $\nabla_\theta = \frac{\partial L}{\partial \theta}$) is computed by convolving input x with ∇_y (i.e., $\nabla_\theta = x \circledast \nabla_y$).

GC of the ET module in Fig. 2 shows the optimizations for both forward and backward paths to reduce the memory and computation overhead. The core idea is to approximate the dot product of convolution by condensing the corresponding regions and transforming it as the product of the regions' average and sum. Specifically, to generate ∇_θ of layer r, feature map x will be condensed during forward propagation as follows:

$$\hat{x}[c,h,w] = \sum_{i=0}^{H'-1}\sum_{j=0}^{W'-1} x_{pad}[c, s_H\cdot h+i, s_W\cdot w+j], \quad (1)$$

where s_H and s_W are the strides and x_{pad} is the padded feature map under the same setting as $\nabla_\theta = x \circledast \nabla_y$. The feature map to be saved during inference aligns with the spatial dimensions of the filter (i.e., $\hat{x} \in \mathbb{R}^{C_{in,z}\times K_H\times K_W}$), which can reduce a significant amount of memory usage as the kernel size is small. During backpropagation, the corresponding gradient of y is condensed as follows:

$$\bar{\nabla}_y = \frac{1}{H'W'}\sum_{i=0}^{H'-1}\sum_{j=0}^{W'-1} \nabla_y[c,i,j], \quad (2)$$

where the gradient map of each channel is condensed into a scalar with its average (i.e., $\bar{\nabla}_y \in \mathbb{R}^{C_{out,z}\times 1\times 1}$). The gradient of the filters is then approximated as the element-wise product of the condensed $\hat{x}$ and $\bar{\nabla}_y$ as $\nabla'_\theta = \hat{x} \odot \bar{\nabla}_y$.

To generate the gradient of x of layer r, the gradient of y will be condensed during backpropagation as follows:

$$\hat{\nabla}_y[c,h,w] = \sum_{i=0}^{K_H-1}\sum_{j=0}^{K_W-1} \nabla_{y_{pad}}[c, s_h\cdot h+i, s_w\cdot w+j], \quad (3)$$

where s_h and s_w are the strides and $\nabla_{y_{pad}}$ is the padded gradient of y under the same setting as $\nabla_x = \nabla_y \circledast \theta^{rot}$. The condensed gradient map ∇_y aligns with the spatial dimensions of x (i.e., $\hat{\nabla}_y \in \mathbb{R}^{C_{out,z}\times H\times W}$). The corresponding filters are condensed as follows:

$$\bar{\theta} = \frac{1}{K_H K_W}\sum_{i=0}^{K_H-1}\sum_{j=0}^{K_W-1} \theta[c_{in}, c_{out}, i, j], \quad (4)$$

where each kernel of the filters is condensed into a scalar (i.e., $\bar{\theta} \in \mathbb{R}^{C_{in,z} \times C_{out,z} \times 1 \times 1}$). The gradient of x is then approximated as the element-wise product of $\nabla'_x = \hat{\nabla}_y \odot \bar{\theta}$.

Computation Overhead Analysis: Considering the vanilla backpropagation process for the convolution layer r, the computation cost (number of FLOPs) obtaining ∇_x can be approximated as $2 \times C_{out,z} \times H \times W \times C_{in,z} \times K_H \times K_W$ while the computation cost obtaining ∇_θ can be approximated as $2 \times C_{in,z} \times K_H \times K_W \times C_{out,z} \times H' \times W'$. The computation overhead obtaining ∇_x and ∇_θ is significantly reduced by the condensed spatial dimension of $\hat{x}$ and $\hat{\nabla}_y$. The total cost of gradient computation can be mitigated as:

$$\text{Cost} = \underbrace{2C_{in,r}C_{out,r} \cdot K_H K_W}_{\nabla'_\theta \approx \hat{x} \odot \hat{\nabla}_y} + \underbrace{2 \cdot C_{in,r}C_{out,r} \cdot HW}_{\nabla'_x \approx \hat{\nabla}_y \odot \bar{\theta}}. \quad (5)$$

There is an extra overhead of condensing x during inference as well as ∇_y and θ during backpropagation. Given the large number of channels and small size of the convolution kernel in typical on-device neural networks, the condensing operation only leaves little impact on computation efficiency (in terms of on-board latency shown in Section IV).

Memory Usage Analysis: The backpropagation process of full model update needs to consider the residential storage of the intermediate value (e.g., feature maps x) and the model parameters (e.g., θ) of all layers. There is also temporal storage for gradients (e.g., ∇_y, ∇_x and ∇_θ) to update the certain layer. Since each convolution layer requires compute gradients for its filter update, the memory usage of storing feature map x is high. With GC, the feature map saved in memory is condensed to $\hat{x} \in \mathbb{R}^{C_{in,r} \times K_H \times K_W}$ of each convolution layer, which can save even more memory space on the layer with a smaller convolution kernel size of θ.

Error Analysis: We show that the approximation error on the gradient introduced by GC can be properly bounded. The mean of the squared error induced by the approximated filter gradient by ∇'_θ is bounded as follows:

$$\frac{\sum_{C_{in}} \sum_{K_H} \sum_{K_W} (\nabla_\theta - \hat{\nabla}_\theta)^2}{C_{in} K_H K_W} \leq \sigma_{max}(\nabla_y) \cdot \sigma_{max}(x), \quad (6)$$

where $\sigma_{max}(\nabla_y)$ is the maximum variance of ∇_y and $\sigma_{max}(x)$ is the maximum variance of x. It is because GC replaces the dot product of two vectors (i.e., two regions in x and ∇_y under convolution) with the product of the vector average and the sum. The mean of the squared error induced by ∇'_x is bounded as follows:

$$\frac{\sum_{C_{in}} \sum_{H} \sum_{W} (\nabla_x - \nabla'_x)^2}{C_{in} HW} \leq \sigma_{max}(\nabla_y) \cdot \sigma_{max}(\theta), \quad (7)$$

where $\sigma_{max}(\theta)$ is the maximum variance of a certain kernel of the filter. The error could be minimized if the components exhibit low variability within the regions being condensed. The error could be further minimized by the prevalent on-device network architecture with a smaller kernel size and highly structured feature map. These properties ensure that the error in gradient has only a limited influence on the quality of backpropagation. We evaluate the advantages in terms of accuracy gain in the experimental section.

C. Alternant Partial Update

We introduce Alternant Partial Update (APU) to further reduce the computation and memory overhead of backpropagation. The core idea of APU is to enable online parameter searching and dynamic partial update by alternatively selecting a subset of filters of each layer to update during training, as shown in Fig. 2. APU dynamically identifies the most significant filters based on their contribution trend to the model's feature representation, and incurs only during inference without gradient computation and extensive intermediate value storage in [15].

To access the importance of a filter $\theta^r_c \in \{\theta^r_1, \theta^r_2, ..., \theta^r_{C_{out}}\}$ of the convolution layer r, the network is fed with inputs $d \in \mathcal{D}_{ip}$ (where $\mathcal{D}_{ip}$ is a small set of input data different from training or validation dataset). The corresponding output feature map at the training epoch t is defined as $y^t_{d,\theta^r_c} \in \mathbb{R}^{H' \times W'}$. For two consecutive training epochs (i.e., time t and $t-1$), APU computes the Mahalanobis-like similarity between all the outputs of the filter θ^r_c for the whole set $\mathcal{D}_{ip}$ as follows:

$$\delta^t_{\theta^r_c} = \sum_{d \in \mathcal{D}_{ip}} \sqrt{\frac{(\mu_{y^t_{d,\theta^r_c}} - \mu_{y^{t-1}_{d,\theta^r_c}})^2}{\sigma^2_{y^t_{d,\theta^r_c}} + \sigma^2_{y^{t-1}_{d,\theta^r_c}}}}, \quad (8)$$

where μ and σ are the mean and variance of the output feature map respectively. The variations of $\delta^t_{\theta^r_c}$ indicate the stability of the certain filter. To quantify the changes of variation between two epochs, the relative variation of the filter is computed as $\Delta^t_{\theta^r_c} = ||\delta^t_{\theta^r_c} - \delta^{t-1}_{\theta^r_c}||$, where the value of $\Delta^t_{\theta^r_c}$ reflects the latest learning progress of the corresponding filter. The importance of the filter θ^r_c is defined as $I^t_{\theta^r_c} = \Delta^t_{\theta^r_c} - \gamma \cdot I^{t-1}_{\theta^r_c}$, where γ is a momentum factor allowing the importance to carry with the temporal trend. A high $I^t_{\theta^r_c}$ indicates that the filter θ^r_c at epoch t does not reach a stable state and still needs to be updated towards an optimal subspace. On the other hand, a lower $I^t_{\theta^r_c}$ indicates that the filter reaches a relatively stable state and does not need further update steps. During inference, APU will choose a subset of filters for the convolution layer r as follows:

$$F^t_r = \{\theta^r_c \in \theta \mid I^t_{\theta^r_c} \geq \tau_\alpha\}, \quad (9)$$

where τ_α is the threshold such that there are $\alpha\%$ of the filters with a higher importance score.

APU enables the network to skip the gradient computation of less important sub-tensors (unchosen filters) of convolution layers. As a result, the computation cost of obtaining ∇'_θ can be further reduced to $2 \cdot \alpha C_{out,r} C_{in,r} \cdot K_H K_W$. The temporal memory required for storing the gradient of filters could be lessened to reduce peak memory usage. Compared with offline searching approaches [3], APU does not require any prior knowledge of the target dataset and thus eliminates the heavy evolutionary search process. In addition, APU enables the network to dynamically select the parameters to update during training, avoiding potential overfitting problems in

Procedure 1: On-Tiny-Device Model Personalization

Input Pre-trained model ϕ, Set of layers to update R, Epochs T, Datasets for personalization and importance score $\mathcal{D}_p$, $\mathcal{D}_{ip}$.

for $t = 1, ..., T$ **do**
  /* --- *Dynamic Parameter Selection via APU* --- */
  for *each layer* $r \in R$ **do**
    Compute $\mu_{y^t_{d,\theta^r_c}}, \sigma_{y^t_{d,\theta^r_c}}$ using the samples in D_{ip}
    $\mathrm{F}^t_r \leftarrow$ by Eqn.8 and Eqn. 9
  end
  /* --- *Perform Fine-tuning via GC* --- */
  for *each layer* $r \in R$ **do**
    /* --- *Forward Propagation Stage* --- */
    Save in memory for $\hat{x} \leftarrow$ by Eqn.1
    /* --- *Backward Propagation Stage* --- */
    Average output feature map gradient $\bar{\nabla}_y$ of $\theta_c \in F^t_r$ by Eqn.2
    Condense output feature map gradient $\hat{\nabla}_y \leftarrow$ by Eqn.3
    Average each filter $\bar{\theta} \leftarrow$ by Eqn.4
    $\nabla'_\theta \leftarrow \hat{x} \odot \bar{\nabla}_y$
    $\nabla'_x \leftarrow \hat{\nabla}_y \odot \bar{\theta}$
  end
end

static parameter selection configuration. Compared with online searching approaches [15], APU does not incur an extra high memory usage by storing only the mean and variance of feature maps for similarity calculation. Procedure 1 illustrates the whole on-tiny-device model personalization process.

IV. Experiments

A. Experimental Setup

Dataset and Data Preparation: We use ImageNet [22] and CIFAR100 [23] as the training dataset to obtain pre-trained deep models. For downstream tasks, we select Flowers (denoted as FL) [24], CU-Birds (denoted as CUB) [25], FEMNIST (denoted as FM) [26], and Omniglot (denoted as OG) [27] to evaluate the out-of-domain adaptability. We further select MiniImageNet (denoted as MI) [28] and CIFAR10 (denoted as C10) [23] for the pre-trained model trained with CIFAR100 and ImageNet respectively.

Baseline: We compare TinyMP with the following seven baselines: (1) *Pre-Train* does not perform any on-device personalization; (2) *Full-FT* [19] fine-tunes the model with a static number of the last few layers; (3) *LinearOnly* [13] only updates the linear layers of the model; (4) *GF-FT* [18] is a SOTA approach to apply gradient pooling on the convolution layer to conduct efficient backpropagation; (5) *TinyTL* [16] updates the residual modules and bias while freezing the backbone model; (6) *SU* [1] is a SOTA approach for on-device training that statically chooses important parameters to update before deployment; (7) *TinyTrain* [15] is a SOTA approach to dynamically choose important parameters to update.

Implementation Details: We employ two efficient neural network architectures (i.e., MCUNet [3] and MobileNetV2 [12]) as the base model. The network structure is optimized with minor modifications to fit the input dimensions and recourse constraints. For each downstream task, we randomly split samples by 1:9 of each class from each dataset as personalization set $\mathcal{D}_p$ and testing set $\mathcal{D}_{test}$. We fine-tune the models on all these personalization set for 50 epochs following [1]. We extract 2 samples of each class from $\mathcal{D}_p$ to form $\mathcal{D}_{ip}$ to conduct the important parameters searching during forward propagation of fine-tuning. We use resolution 64 for all datasets and the parameters to be updated within the last 16 layers of all models for fair comparison.

We implement approaches on OpenMV-H7 [9] (with Cortex M7) to evaluate the practical performances of on-device training. The board runs with ∼400 mW and is equipped with 2 MB flash and 1 MB SRAM. We measure the latency and peak memory usage of all approaches. For speed evaluation purposes, the accuracy is evaluated on a server (Intel i9 9900K CPU with an RTX4090) using PyTorch (2.3.0).

TABLE I: Top-1 accuracy of models trained on ImageNet.

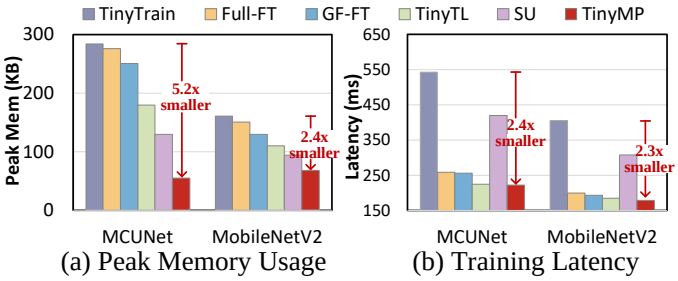
Model	Method	OG	CUB	FL	FM	C10	Avg.
MCUNet	Pre-Train	1.75	1.02	0.76	1.24	0.78	1.11
	LinearOnly [13]	73.0	68.0	89.6	75.2	60.8	73.3
	TinyTL [16]	31.5	21.5	52.0	79.6	34.8	43.9
	Full-FT [19]	83.8	64.2	91.5	84.1	66.2	78.0
	GF-FT [18]	78.0	61.0	89.6	82.9	65.4	75.4
	SU [1]	**87.0**	65.0	92.9	82.6	68.5	79.2
	TinyTrain [15]	85.8	64.2	93.3	84.3	67.8	79.1
	TinyMP	83.5	**66.1**	**93.5**	**84.7**	**69.1**	**79.4**
Mobile NetV2	Pre-Train	1.00	0.43	0.99	0.81	1.00	0.85
	LinearOnly [13]	61.0	62.0	81.9	74.0	57.2	67.2
	TinyTL [16]	32.0	13.0	45.5	79.8	35.7	41.2
	Full-FT [19]	89.0	70.3	91.0	85.7	70.0	81.2
	GF-FT [18]	88.8	69.4	90.9	84.6	68.9	80.5
	SU [1]	**89.8**	68.1	92.2	85.5	69.9	81.1
	TinyTrain [15]	88.3	**70.8**	93.7	85.4	66.0	80.8
	TinyMP	89.5	68.6	**94.0**	**86.5**	**70.3**	**81.8**

TABLE II: Top-1 accuracy of models trained on CIFAR100.

Model	Method	OG	CUB	FL	FM	MI	Avg.
MCUNet	Pre-Train	0.82	0.95	0.85	0.58	0.81	0.80
	LinearOnly [13]	20.3	11.7	42.7	40.7	23.6	27.8
	TinyTL [16]	20.5	11.3	52.1	76.2	35.6	39.1
	Full-FT [19]	59.3	23.6	63.3	82.3	40.6	53.8
	GF-FT [18]	43.0	20.9	61.3	80.9	39.7	49.2
	SU [1]	71.8	25.5	**72.9**	**86.2**	41.7	**59.6**
	TinyTrain [15]	53.8	25.1	65.1	82.9	38.4	53.0
	TinyMP	**73.3**	**26.7**	71.1	82.8	**42.6**	59.3
Mobile NetV2	Pre-Train	1.25	1.11	0.99	3.59	0.77	1.54
	LinearOnly [13]	15.3	5.37	18.2	46.7	14.3	20.0
	TinyTL [16]	22.5	11.4	55.8	79.9	48.1	43.5
	Full-FT [19]	65.3	28.3	73.4	**85.1**	44.9	59.4
	GF-FT [18]	63.8	22.8	65.1	84.8	43.9	56.1
	SU [1]	83.8	29.2	72.5	84.9	45.1	63.1
	TinyTrain [15]	80.8	29.5	73.5	84.8	44.0	62.5
	TinyMP	**89.0**	**32.6**	**78.0**	84.8	**49.7**	**66.8**

B. Experimental Results

Accuracy: We first present the Top-1 classification accuracy of *TinyMP* and other baselines. As shown in Table I and Table II, *Pre-Train* presents the lowest accuracy among all the baselines over each target dataset, demonstrating the necessity of on-device model personalization. *LinearOnly* outperforms *Pre-Train* with a marginal improvement while still achieving relatively low accuracy compared with other baselines. These results suggest that solely updating auxiliary components (e.g., linear layers, bias, residual module) is insufficient to adapt models to real-world applications involving out-of-domain data distributions. For methods that focus on updating the



(a) Peak Memory Usage (b) Training Latency

Fig. 3: Peak memory usage and training latency of the on-device personalization methods.

model backbones, *Full-FT* and *GF-FT* achieve higher accuracy by fine-tuning the entire backbone. However, both methods are resource-intensive, requiring significant memory and computational power, which makes them impractical to include more layers to be updated for highly resource-constrained environments like MCU. *SU* achieves even better accuracy by pre-selecting the optimal parameters using large quantities of user-end data and extensive searches. Despite its accuracy gains, *SU*'s reliance on extensive offline computation limits its applicability for real-time, on-device scenarios. While *TinyTrain* offers an online alternative, it suffers from high latency as it iteratively evaluates all parameters by saving and computing the product of all layers' gradient and activation, which makes it less efficient.

TinyMP consistently delivers the highest average accuracy across most downstream tasks, surpassing baseline methods in almost all cases. For models pre-trained on ImageNet, *TinyMP* achieves up to 24.7% and 30.3% higher accuracy across specific tasks than *Full-FT* and *GF-FT*, respectively. When compared to *SU*, *TinyMP* delivers an even better average top-1 accuracy across different downstream tasks, with only a marginal 0.3% difference on MCUNet pre-trained on CIFAR100. When compared to *TinyRTrain*, *TinyMP* delivers a better average accuracy performance (up to 6.3%) across all downstream tasks. *TinyMP* achieves these accuracy performances while operating in a memory- and computation-efficient manner, making it a practical solution for on-device personalization.

Memory Usage and Training Latency: We further evaluate the practical on-device performances of *TinyMP* and other baselines on the tiny device (i.e., OpenMV-H7). Fig. 3 presents the measured peak memory usage and training latency of MCUNet and MobileNetV2. In terms of peak memory usage shown in Fig. 3(a), TinyMP demonstrates superior memory efficiency, achieving reductions of 2.4× to 5.2× compared to SOTA methods such as *TinyTrain*, *SU*, and *GF-FT*. This substantial reduction highlights *TinyMP*'s capability to conserve memory resources, enabling deployment on devices with extremely limited memory. For training latency shown in Fig. 3(b), *TinyMP* also outperforms existing SOTA methods by achieving speedups ranging from 1.1× to 2.4× for MCUNet, 1.1x to 2.3× for MobileNetV2. These results underscore *TinyMP*'s ability to enable fast and efficient on-device personalization, essential for real-time applications in edge scenarios. Importantly, *TinyMP* achieves these efficiency gains without sacrificing accuracy, or even surpassing baseline methods in performance.

TABLE III: Average accuracy of models personalized under different scheme settings.

Method	MCUNet	MobileNetV2
L1 Norm	75.3	79.0
L2 Norm	74.0	75.9
APU-Only	81.3	84.1
GC-Only	77.5	80.5
TinyMP (20%)	79.5	82.8
TinyMP (80%)	79.1	81.1

Overall, *TinyMP* demonstrates a strong balance between accuracy, computational efficiency, and memory usage, addressing the challenges of real-time model personalization on resource-constrained edge devices. While existing methods like *SU* or *TinyTrain* excel in certain downstream tasks, their offline requirements or high computation costs limit their usability in real-world applications. *TinyMP* emerges as a robust solution, enabling efficient, high-accuracy model adaptation without exceeding the resource constraints of tiny devices.

Ablation Study: Table III presents the results of an ablation study evaluating the average accuracy of MCUNet and MobileNetV2 (pre-trained on ImageNet) under different scheme settings, evaluating the contributions of the key components of *TinyMP*. Dynamic parameter selection using APU-only achieves significantly higher accuracy (81.3% for MCUNet and 84.1% for MobileNetV2) compared to static parameter selection methods like L1 Norm and L2 Norm. The performances demonstrate APU's ability to mitigate overfitting and reduce computational costs for backpropagation. The GC-only scheme achieves competitive accuracy (77.5% for MCUNet and 80.5% for MobileNetV2), closely matching *Full-FT*'s performances (78.0% of MCUNet and 81.2% of MobileNetV2 shown in Table I) while leveraging gradient approximation to improve efficiency. TinyMP demonstrates robust performance when freezing 20% and 80% of convolutional filters, achieving only a small accuracy drop when freezing more filters. These results validate TinyMP's ability to efficiently select and update important parameters, balancing accuracy and computational efficiency in resource-constrained environments.

V. Conclusion

In this paper, we propose the on-tiny-device model personalization (TinyMP) framework for resource-constrained embedded devices. TinyMP aims to achieve self-adaptive on-device model training by introducing Alternant Partial Update to enable online parameter selection and Gradient Condensing to conduct memory- and computation-efficient backpropagation. Experimental results show that TinyMP outperforms SOTA on-device approaches in terms of accuracy while achieving better practical performances over downstream tasks.

VI. Acknowledgment

This work is supported in part by Taishan Scholars Youth Expert Program (Grant No. TSQN202408049), Young Talent of Lifting Engineering for Science and Technology in Shandong, China (Grant No. SDAST2024QTA078).

REFERENCES

[1] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-device training under 256kb memory," *Advances in Neural Information Processing Systems*, vol. 35, pp. 22 941–22 954, 2022.

[2] Z. Jia, Z. Wang, F. Hong, L. Ping, Y. Shi, and J. Hu, "Learning to learn personalized neural network for ventricular arrhythmias detection on intracardiac EGMs," in *IJCAI-21*, 2021, pp. 2606–2613.

[3] J. Lin, W.-M. Chen, Y. Lin, C. Gan, S. Han *et al.*, "Mcunet: Tiny deep learning on iot devices," *Advances in neural information processing systems*, vol. 33, pp. 11 711–11 722, 2020.

[4] F. Svoboda, J. Fernandez-Marques, E. Liberis, and N. D. Lane, "Deep learning on microcontrollers: A study on deployment costs and challenges," in *Proceedings of the 2nd European Workshop on Machine Learning and Systems*, 2022, pp. 54–63.

[5] D. Xu, M. Xu, Q. Wang, S. Wang, Y. Ma, K. Huang, G. Huang, X. Jin, and X. Liu, "Mandheling: Mixed-precision on-device dnn training with dsp offloading," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, 2022, pp. 214–227.

[6] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[7] STMicroelectronic, "Arm® Cortex®-M4 in a nutshell," 2024. [Online]. Available: https://www.st.com/content/st_com/en/arm-32-bit-microcontrollers/arm-cortex-m4.html

[8] STMicroelectronics, "Discovery kit with STM32F469NI MCU," 2020. [Online]. Available: https://www.st.com/en/evaluation-tools/32f469idiscovery.html

[9] OpenMV, "OpenMV-H7: A Python programmable machine vision camera," 2018. [Online]. Available: https://cdn.sparkfun.com/assets/a/3/f/d/9/OpenMV-H7_Datasheet.pdf

[10] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, "Tensorflow lite micro: Embedded machine learning for tinyml systems," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.

[11] STMicroelectronics, "STM32 solutions for Artificial Neural Networks," 2020. [Online]. Available: https://www.st.com/content/st_com/en/ecosystems/stm32-ann.html

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520.

[13] H. Ren, D. Anicic, and T. A. Runkler, "Tinyol: Tinyml with online-learning on microcontrollers," in *2021 international joint conference on neural networks (IJCNN)*. IEEE, 2021, pp. 1–8.

[14] A. Quélennec, E. Tartaglione, P. Mozharovskyi, and V.-T. Nguyen, "Towards on-device learning on the edge: Ways to select neurons to update under a budget constraint," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 685–694.

[15] Y. D. Kwon, R. Li, S. Venieris, J. Chauhan, N. D. Lane, and C. Mascolo, "Tinytrain: Resource-aware task-adaptive sparse training of dnns at the data-scarce edge," in *Forty-first International Conference on Machine Learning*, 2024.

[16] H. Cai, C. Gan, L. Zhu, and S. Han, "Tinytl: Reduce memory, not parameters for efficient on-device learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 285–11 297, 2020.

[17] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "Qsgd: Communication-efficient sgd via gradient quantization and encoding," *Advances in neural information processing systems*, vol. 30, 2017.

[18] Y. Yang, G. Li, and R. Marculescu, "Efficient on-device training via gradient filtering," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 3811–3820.

[19] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[20] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signsgd: Compressed optimisation for non-convex problems," in *International Conference on Machine Learning*. PMLR, 2018, pp. 560–569.

[21] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, "Scalable methods for 8-bit training of neural networks," *Advances in neural information processing systems*, vol. 31, 2018.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[23] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[24] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian conference on computer vision, graphics & image processing*. IEEE, 2008, pp. 722–729.

[25] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "Caltech-ucsd birds," Tech. Rep. CNS-TR-2011-001, 2011.

[26] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[27] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "The omniglot challenge: a 3-year progress report," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 97–104, 2019.

[28] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," *Advances in neural information processing systems*, vol. 29, 2016.

LIO-DPC: Accurate and Fast LiDAR-Inertial Odometry with Dynamic Pose Chain

Yuexin Mu[1], Ao Ren[1], Duo Liu[1], Zihao Zhang[1], Haojie Lu[1], Longyi Zhou[1]
Huachen Tan[1], Kan Zhong[1], Yujuan Tan[1], and Chaoxia Qin[1]

***Abstract*—LiDAR-inertial odometry is widely used in robotics navigation, autonomous driving, and drone operation to provide precise, low-latency motion estimation. Filter-based methods are fast but suffer from significant cumulative errors. Graph optimization methods reduce cumulative errors through loop closure detection but are computationally expensive. In this work, we propose LIO-DPC, a framework that combines the benefits of the filter-based approach and graph-based approach. First, we propose a dynamic pose chain optimization method. It generates an initial pose chain using the fast filter. This is followed by applying computationally efficient local graph optimization to a set of local pose chains to generate refined relative poses, which are then used to update the motion estimation. Second, we propose a loop sparsification approach to select representative loops that are both temporally and spatially proximate, to reduce the computational complexity in graph optimization and minimize loop errors. Extensive experiments demonstrate that LIO-DPC achieves real-time performance and outperforms state-of-the-art methods in accuracy.**

***Index Terms*—LiDAR-inertial odometry, SLAM, state estimation**

I. Introduction

LiDAR-inertial odometry is a method that combines LiDAR data and inertial measurement unit (IMU) data for self-motion estimation [1]. Since LiDAR can directly acquire depth information from the environment, LiDAR-inertial odometry is not affected by ambient lighting conditions [2]. Moreover, integrating high-frequency motion measurements from the IMU can effectively address issues of motion distortion in LiDAR point clouds and degradation problems caused by indistinct environmental geometric information [3]. Therefore, LiDAR-inertial odometry is widely employed in intelligent robots, autonomous vehicles, unmanned aerial vehicles, and other terminal devices, providing accurate and low-latency self-motion estimation information.

In current research on LiDAR-inertial odometry, methods can mainly be divided into two categories: filtering-based methods and graph-based methods [4], [5]. Filtering-based methods, including FAST-LIO [6] and LINS [7], utilize the previous state and inputs of the sensors to estimate the current state of the robot in real time. However, these methods suffer from cumulative errors that increase over time or with the distance traveled, potentially leading to inaccurate or even failed state estimations. Graph-based methods, including LIO-SAM [8] and LIPS [9], optimize the state of the robot by constructing factor graphs, typically using measurement factors of the LiDAR, prediction factors of the IMU, and factors of the loop closures. The introduction of loop closure factors can effectively correct cumulative errors along the entire loop trajectory, making graph-based methods more accurate in large-scale environments and long-term state estimation. However, as the scale of optimization increases, the computational load of these methods also rises significantly, leading to decreased real-time performance.

A recent study [10] has explored combining filtering and graph-based approaches by introducing graph optimization after loop closure detection. Although this mitigates cumulative errors, it introduces a significant delay, as the filter state cannot be updated until graph optimization is complete. This trade-off highlights a key challenge: how to retain the fast updates of filtering methods while leveraging the accuracy improvements of graph optimization. Addressing this issue is crucial for advancing LIO in real-time applications.

In this work, we propose LIO-DPC, a novel LIO framework that addresses the aforementioned challenges through two key innovations: dynamic pose chain and loop sparsification. We deem that the key to combining the advantages of the filtering methods with the graph optimization methods lies in decoupling the two and reducing the computation complexity of the graph optimization, allowing the two methods to operate in parallel. To achieve this, we propose a dynamic pose chain—a sequence of relative poses that can be dynamically updated. Initially, filtering methods provide high-frequency relative pose estimates to construct the chain. Upon detecting a loop closure, the system builds a local pose graph using both the loop closure and the dynamic pose chain. This graph is optimized to refine the relative pose estimates, which are subsequently used to update the chain. This design ensures real-time performance while reducing cumulative errors through loop closure correction.

Furthermore, the system often accumulates multiple loop closures upon entering the pose graph optimization stage. These loop closures are spatially and temporally closed and hold errors. Optimizing all of these loop closures not only consumes excessive computational resources but also risks introducing additional inaccuracies. To address these issues, we propose the loop sparsification strategy to select only the most impactful and high-quality loop closures for optimiza-

[1]The authors are with the Chongqing University, Chongqing, China, yuexinmu@stu.cqu.edu.cn, ren.ao@cqu.edu.cn.
Corresponding authors: Ao Ren and Duo Liu.
The code will be available at https://github.com/YuexinMu/lio_dpc.git.

tion. Specifically, an evaluation function is designed to account for both the scale of the pose graph optimization and their respective registration scores. Ultimately, the loop closure with the highest comprehensive score is selected for local pose graph optimization, thereby reducing computational overhead and minimizing errors.

Our contributions can be summarized as follows:

- We propose a dynamic pose chain method that enables the parallel operation of filtering and graph optimization. While maintaining the high-frequency pose updates of filters, it leverages loop closure detection and graph optimization to enhance system accuracy.
- We propose a loop quality evaluation metric to assess temporally and spatially adjacent loops, aiming to select high-quality loop closures for optimization.
- We develop a LiDAR-inertial odometry system (LIO-DPC), and experimental results on multiple datasets demonstrate that our method achieves a significant improvement in accuracy compared to state-of-the-art approaches while maintaining high real-time performance.

II. Related Works

The LIO system estimates the state of robots by integrating relative transformations between consecutive LiDAR scans and IMU measurements. The iterative closest point (ICP) method [11] iteratively minimizes the distance between point clouds. However, motion distortion in LiDAR data due to sensor movement can cause significant drift if relying solely on point cloud matching. Therefore, LiDAR is typically combined with other sensors like IMU for state estimation [12]–[16]. Current LIO systems employ either filtering-based or graph optimization-based methods to fuse LiDAR and IMU data for robot state estimation.

Several works have employed Kalman Filters to fuse LiDAR and IMU data for state estimation [17]–[22]. Tang et al. [23] used an Extended Kalman Filter (EKF), while Zhen et al. [24] adopted an Error State Kalman Filter (ESKF) to enhance robustness and accuracy. LINS [7] utilized an Iterative Error State Kalman Filter (IESKF) for improved speed over graph-based methods but still faced computational burdens, especially when calculating the Kalman gain due to numerous LiDAR measurements. FAST-LIO [6] addressed this by introducing a novel Kalman gain formulation. FAST-LIO2 [2] further improved accuracy and speed by eliminating feature extraction and directly registering raw LiDAR data using the ikd-Tree structure. Despite these computational improvements, filtering-based LiDAR-inertial odometry systems still cannot effectively mitigate error accumulation during operation.

Graph-based methods enhance odometry performance by incorporating various factors into factor graphs, though this increases computational load. LIPS [9] includes continuous IMU pre-integration and 3D plane factors from LiDAR measurements. IN2LAMA [25] uses upsampled pre-integrated IMU data to undistort LiDAR scans, forming a batch manifold optimization with LiDAR factors, IMU bias factors, and sensor time offsets. LIO-SAM [8] employs IMU pre-integration to undistort LiDAR data for joint optimization in a factor graph. LILI-OM [26] utilizes hierarchical pose graph optimization and introduces a novel feature extraction method for solid-state LiDARs with irregular scanning patterns. Additionally, Setterfield et al. [27] directly integrate feature correspondences from LiDAR measurements into the factor graph.

To leverage the speed of filtering methods and the error reduction benefits of graph optimization with loop closure detection, Tang et al. [10] attempted to construct factor graphs and update filter states upon loop detection. However, the slow graph optimization process negates the real-time advantages of filtering due to the required waiting time.

III. Method

A. System Overview

The system workflow of LIO-DPC is shown in Fig.1. The system first processes LiDAR and IMU data through a filter to quickly estimate the pose of the current moment relative to the previous one. The incremental update module uses this relative pose along with the existing dynamic pose chain to perform incremental updates, providing high-frequency pose estimates while constructing the initial dynamic pose chain. When the loop detection module identifies a loop closure, it passes it to the loop sparsification module, which selects the highest-scoring loop based on the loop scores and sends it to the local pose graph optimization module. The local pose graph optimization module constructs a local pose graph, optimizes it using the least squares method, and computes the optimized relative poses. Subsequently, the batch update module utilizes these optimized relative poses to perform batch updates on the dynamic pose chain, obtaining more accurate pose estimates. Since both incremental updates and batch updates operate only on the dynamic pose chain, they do not interfere with each other and can be executed entirely in parallel. By employing this approach, LIO-DPC achieves fast and precise pose estimation.

B. Dynamic Pose Chain

The dynamic pose chain is a sequentially updated set of relative poses that decouples filtering methods from graph optimization methods, allowing them to operate independently while effectively combining their respective advantages. Initially, the filtering method performs incremental updates, constructing the initial dynamic pose chain and providing real-time pose estimates. When the system detects a loop closure, it constructs a local pose graph based on the dynamic pose chain for optimization. This process yields more accurate local poses and performs batch updates on the dynamic pose chain, reducing accumulated errors and enhancing the accuracy of pose estimation. Since the filtering method and the graph optimization method operate independently on the dynamic pose chain, they do not interfere with each other. Consequently, the system update frequency matches the filtering method, while still providing substantial accuracy improvements.

We can represent the pose of the robot using $\mathbf{x}$:

$$\mathbf{x} = \mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in SE(3); \quad \mathbf{R} \in SO(3), t \in \mathbb{R}^3 \tag{1}$$

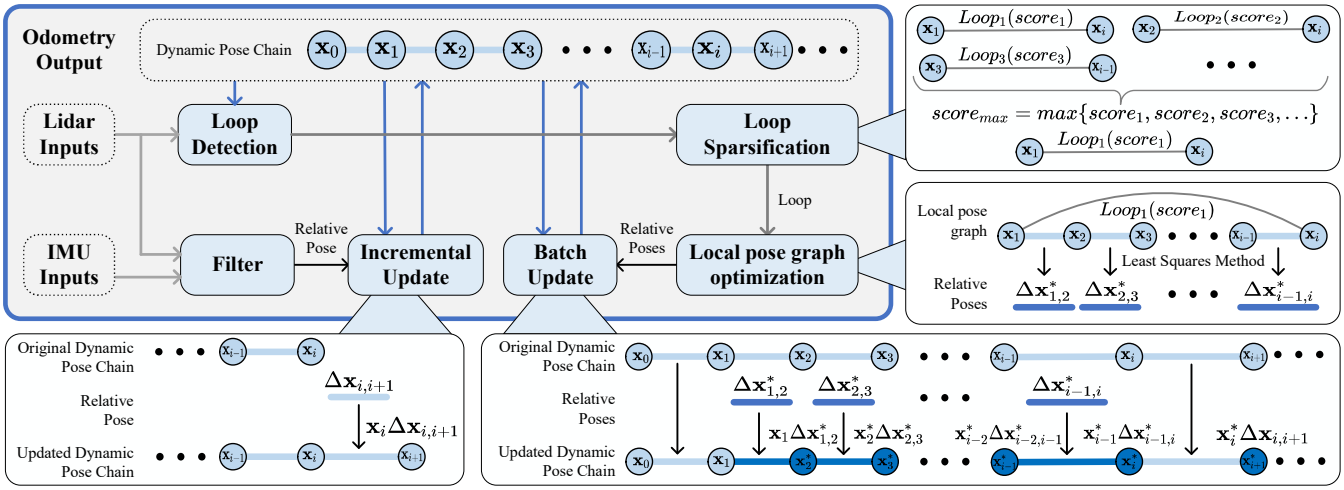


Fig. 1. System overview of LIO-DPC.

where $\mathbf{R}$ is the rotation matrix, and $\mathbf{t}$ is the position vector. $SE(3)$ denotes the special Euclidean group.

We denote the relative pose of the robot by $\Delta\mathbf{x}$. Assuming the poses of the robot at times i and j are known as $\mathbf{x}_i$ and $\mathbf{x}_j$, the relative pose from time i to time j is represented as:

$$\Delta\mathbf{x}_{i,j} = \mathbf{x}_i^{-1}\mathbf{x}_j = \begin{bmatrix} \mathbf{R}_i^\top \mathbf{R}_j & \mathbf{R}_i^\top(\mathbf{t}_j - \mathbf{t}_i) \\ \mathbf{0}^\top & 1 \end{bmatrix} \tag{2}$$

where $\mathbf{R}_i^\top$ is the transpose of $\mathbf{R}_i$.

We define the dynamic pose chain $\mathcal{C}_{0,n}$ as:

$$\mathcal{C}_{0,n} = (\mathbf{x}_0, \{\Delta\mathbf{x}_{0,1}, \Delta\mathbf{x}_{1,2}, \ldots, \Delta\mathbf{x}_{n-1,n}\}); \quad n \in \mathbb{N} \tag{3}$$

where $\mathbf{x}_0$ is the starting pose of the dynamic pose chain, and the set$\{\Delta\mathbf{x}_{0,1}, \Delta\mathbf{x}_{1,2}, \ldots, \Delta\mathbf{x}_{n-1,n}\}$ represents the sequence of continuous relative poses from $\mathbf{x}_0$ to $\mathbf{x}_n$. The initial dynamic pose chain is defined as $\mathcal{C}_{0,0} = (\mathbf{x}_i, \emptyset)$, indicating only the initial pose with no sequence of relative poses.

1) Incremental Update: Incremental updating involves adding new relative poses to the existing dynamic pose chain and computing the absolute pose at the latest time relative to the initial time. Since this process increases the length of the dynamic pose chain, it is called incremental updating. The pose estimated directly by the filter contains accumulated historical errors; therefore, we only use the relative pose estimated by the filter from the previous moment to the current moment. Because this time interval is short and the error is relatively small. Then, we calculate the current pose based on the previous pose in the dynamic pose chain and the relative pose. Since the pose in the dynamic pose chain can improve accuracy through stock updates, the pose estimation obtained in this way is more precise. Additionally, when the system detects a loop closure, the dynamic pose chain constructed through incremental updating can quickly build a local pose graph and optimize its relative pose. This not only enhances accuracy but also does not affect the update speed of the filter.

Assuming the system receives new LiDAR and IMU data, the relative pose calculated by the filter is $\Delta\mathbf{x}_{n-1,n}$. According to the definition of the relative pose chain in (3), we can add the new relative pose to the existing dynamic pose chain, completing the incremental update:

$$\mathcal{C}_{0,n} = \mathcal{C}_{0,n-1} + \Delta\mathbf{x}_{n-1,n} \tag{4}$$

Simultaneously, computing the absolute pose at the current moment relative to the initial moment can be achieved by:

$$\mathbf{x}_n = \mathbf{x}_{n-1}\Delta\mathbf{x}_{n-1,n} \tag{5}$$

LIO-DPC uses (4) to perform incremental updates on the dynamic pose chain. This allows for independent optimization of existing relative poses in the chain without impacting filter updates. Using (5), we ensure that the overall real-time performance of the system and the speed of the filtering updates remain consistent, thereby achieving real-time performance comparable to filtering methods.

2) Local Pose Graph Optimization: Although the dynamic pose chain integrates relative poses generated by filtering, it can accumulate significant errors over time. To minimize these errors, we employ local pose graph optimization to obtain more accurate relative poses. Specifically, we construct a local pose graph using loops and segments of the dynamic pose chain and optimize the relative poses within it. These optimized relative poses then serve as inputs to the batch update module.

Assuming the local pose graph optimization module detects a loop from time i to j, where $i, j \in [1, n]$ and $i < j$, we represent the poses of the robot as nodes and the relative poses between them as edges. Using the dynamic pose chain $\mathcal{C}_{0,n}$ and the loop edge $\Delta\mathbf{x}_{i,j}$, we can construct a pose graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ that includes $m(m = j - i + 1)$ nodes and m edges. The sets of nodes $\mathcal{V}$ and edges $\mathcal{E}$ are defined as:

$$\begin{cases} \mathcal{V} = \{\mathbf{x}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_{j-1}, \mathbf{x}_j\} \\ \mathcal{E} = \{\Delta\mathbf{x}_{i,i+1}, \ldots, \Delta\mathbf{x}_{j-1,j}, \Delta\mathbf{x}_{j,i}\} \end{cases} \tag{6}$$

Since $\mathcal{G}$ includes loop constraints, it can be optimized using the least squares method, with the error function defined as:

$$\mathbf{F}(\mathbf{x}) = \sum_{k,l \in \mathcal{G}} \mathbf{e}(\mathbf{x}_k, \mathbf{x}_l, \Delta\mathbf{x}_{k,l})^\top \Omega_{k,l} \mathbf{e}(\mathbf{x}_k, \mathbf{x}_l, \Delta\mathbf{x}_{k,l}) \tag{7}$$

where $\Omega_{k,l}$ represents the information matrix between nodes k and l, and $\mathbf{e}(\mathbf{x}_k, \mathbf{x}_l, \Delta\mathbf{x}_{k,l})$ represents the error between the edge $\Delta\mathbf{x}_{k,l}$ and node poses. To simplify notation, we denote

the error function as $\mathbf{e}_{k,l}(\mathbf{x})$. The optimization objective is to minimize this error function:

$$\begin{aligned}\mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmin}} \mathbf{F}(\mathbf{x}) \\ &= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{(k,l)\in\mathcal{G}} \mathbf{e}_{k,l}(\mathbf{x})^\top \Omega_{k,l} \mathbf{e}_{k,l}(\mathbf{x})\end{aligned} \tag{8}$$

With the starting point $\mathbf{x}_i$ fixed, the optimized set of node poses can be obtained through least squares optimization:

$$\mathcal{V}^* = \{\mathbf{x}_i, \mathbf{x}^*_{i+1}, \ldots, \mathbf{x}^*_{j-1}, \mathbf{x}^*_j\} \tag{9}$$

Correspondingly, the optimized relative poses between adjacent nodes can be computed:

$$\mathcal{E}^* = \{\Delta\mathbf{x}^*_{i,i+1}, \ldots, \Delta\mathbf{x}^*_{j-1,j}\} \tag{10}$$

The incorporation of loop constraints means these optimized relative poses have less cumulative error compared to the original poses before optimization.

3) Batch Update: We use the relative poses obtained from the local pose graph optimization module to update the existing dynamic pose chain and compute the absolute pose of the latest moment relative to the initial moment. We refer to this process as batch updating, which modifies the relative poses in the existing dynamic pose chain to effectively reduce cumulative errors. Since the batch update process operates solely on the dynamic pose chain, it does not affect the filtering process. Thus, it enhances the accuracy of pose estimation for the system while maintaining the speed of filter updates.

According to the definition of the dynamic pose chain in (3), given the chains $\mathcal{C}_{a,b}$ and $\mathcal{C}_{b,c}$:

$$\begin{cases} \mathcal{C}_{a,b} = (\mathbf{x}_a, \Delta\mathbf{x}_{a,a+1}, \ldots, \Delta\mathbf{x}_{b-1,b}) \\ \mathcal{C}_{b,c} = (\mathbf{x}_b, \Delta\mathbf{x}_{b,b+1}, \ldots, \Delta\mathbf{x}_{c-1,c}) \end{cases} \tag{11}$$

they can be merged to form:

$$\mathcal{C}_{a,c} = (\mathbf{x}_a, \Delta\mathbf{x}_{a,a+1}, \ldots, \Delta\mathbf{x}_{b,b+1}, \ldots, \Delta\mathbf{x}_{c-1,c})$$

We define the merging operation as:

$$\mathcal{C}_{a,c} = \mathcal{C}_{a,b} \oplus \mathcal{C}_{b,c}; \quad a \le b \le c \tag{12}$$

Similarly, we can define the disassembly formula for the dynamic pose chain:

$$\mathcal{C}_{a,b} = \mathcal{C}_{a,c} \ominus \mathcal{C}_{b,c}; \quad a \le b \le c \tag{13}$$

Based on the optimized relative poses from (10), we can further construct the dynamic pose chain between moments i and j as:

$$\mathcal{C}^*_{i,j} = (\mathbf{x}_i, \{\Delta\mathbf{x}^*_{i,i+1}, \ldots, \Delta\mathbf{x}^*_{j-1,j}\}) \tag{14}$$

Using (12) and (13), we can perform a batch update on the original dynamic pose chain:

$$\begin{aligned}\mathcal{C}^*_{0,n} &= \mathcal{C}_{0,n} \ \ominus \ \mathcal{C}_{i,j} \ \oplus \ \mathcal{C}^*_{i,j} \\ &= (\mathbf{x}_0, \ \{\Delta\mathbf{x}_{0,1}, \ \ldots, \ \Delta\mathbf{x}_{i-1,i}, \\ &\qquad \Delta\mathbf{x}^*_{i,i+1}, \ \ldots, \ \Delta\mathbf{x}^*_{j-1,j}, \\ &\qquad \Delta\mathbf{x}_{j,j+1}, \ \ldots, \ \Delta\mathbf{x}_{n-1,n}\})\end{aligned} \tag{15}$$

where $\mathcal{C}_{0,n}$ and $\mathcal{C}_{i,j}$ respectively represent the original dynamic pose chain and the local dynamic pose chain. $\mathcal{C}^*_{0,n}$ represents the dynamic pose chain after batch updating.

Consequently, the latest pose of the robot relative to the initial moment, $\mathbf{x}_n$, can be calculated using the recursive relationship:

$$\mathbf{x}_n = \mathbf{x}_{n-1}\Delta\mathbf{x}_{n-1,n} \tag{16}$$

By expanding this recursive relationship, $\mathbf{x}_n$ can be expressed as a product of the initial pose and a series of relative pose transformations:

$$\begin{aligned}\mathbf{x}_n &= \mathbf{x}_0\Delta\mathbf{x}_{0,1}\Delta\mathbf{x}_{1,2}\cdots\Delta\mathbf{x}_{n-1,n} \\ &= \mathbf{x}_0 \prod_{k=1}^{n} \Delta\mathbf{x}_{k-1,k}\end{aligned} \tag{17}$$

Thus, after the batch update, the optimized current pose of the robot can be computed using (17) and updated dynamic pose chain:

$$\mathbf{x}^*_n = \mathbf{x}_0 \prod_{k=1}^{i} \Delta\mathbf{x}_{k-1,k} \prod_{k=i+1}^{j} \Delta\mathbf{x}^*_{k-1,k} \prod_{k=j+1}^{n} \Delta\mathbf{x}_{k-1,k} \tag{18}$$

It is important to note that due to the non-commutativity of pose transformations, the symbol $\prod$ in the formulas represents the right multiplication of relative poses $\Delta\mathbf{x}_{k-1,k}$ in chronological order from $k = 1$ to $k = n$.

LIO-DPC reduces cumulative errors by applying local pose graph optimization and batch updates to the dynamic pose chain, enhancing pose accuracy. As this process operates independently of the filter, it preserves the filter's real-time update capability. This design improves overall precision without sacrificing real-time performance.

C. Loop Sparsification

During the batch updating process of the dynamic pose chain, multiple loop closures may exist when entering pose graph optimization due to the high speed of loop closure detection. Since these loop closures are temporally and spatially close and may contain errors, optimizing all of them does not significantly improve the results but consumes excessive computational resources and introduces additional errors. Therefore, we aim to optimize only the loop closure that contributes the most to the overall optimization and has the highest quality. Specifically, we evaluate loop closures using two metrics: their impact on the overall optimization scale and their own point cloud registration scores.

We can represent a loop closure as:

$$\mathcal{L} = (\mathbf{r}, \mathbf{p}, \Delta\mathbf{x}) \tag{19}$$

where $\mathbf{r}$ is the time difference between the current pose and the previous pose involved in the loop closure, $\mathbf{p}$ is the point cloud registration score, and $\Delta\mathbf{x}$ is the relative pose transformation associated with the loop closure. Larger $\mathbf{r}$ values correspond to longer loop closure paths, resulting in more extensive pose graphs for optimization. Higher $\mathbf{p}$ values indicate superior registration quality with reduced inherent loop closure errors.

Therefore, we comprehensively evaluate loop closures using these two metrics, $\mathbf{r}$ and $\mathbf{p}$.

Suppose there are m loop closure candidates when entering the graph optimization module. Since $\mathbf{r}$ and $\mathbf{p}$ have different orders of magnitude, we need to normalize $\mathbf{r}$ and $\mathbf{p}$ across these m loop closures.

$$\begin{cases} \bar{\mathbf{r}}_i = \frac{\mathbf{r}_i}{\sum_{j=1}^{m} \mathbf{r}_j}; \quad i \in [1, m] \\ \bar{\mathbf{p}}_i = \frac{\mathbf{p}_i}{\sum_{j=1}^{m} \mathbf{p}_j}; \quad i \in [1, m] \end{cases} \tag{20}$$

where $\bar{\mathbf{r}}_i$ and $\bar{\mathbf{p}}i$ represent the normalized pose graph optimization scale score and the normalized loop closure registration score for the i-th loop closure, respectively. The normalization is performed using the sums $\sum_{j=1}^{m} \mathbf{r}_j$ and $\sum_{j=1}^{m} \mathbf{p}_j$, which are the total scores over all m loop closures. We propose a weighting coefficient $\mathbf{k} \in [0, 1]$ to balance the contributions of the two scores and compute the final score for each loop closure as follows:

$$\begin{cases} \mathcal{S}_i = k * \bar{\mathbf{r}}_i + (1 - k) * \bar{\mathbf{p}}_i; \quad k \in [0, 1], i \in [1, m] \\ \mathcal{S}_{last} = max\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m\} \end{cases} \tag{21}$$

here $\mathcal{S}_i$ represents the final score of the i-th loop closure.

Using this as a criterion, we select the loop closure with the highest score, $\mathcal{S}_{\text{last}}$, from the m loop closures and send its corresponding loop closure to the final pose graph optimization module.

IV. Experiments

A. Experimental Setup

Datasets. We conducted experiments using four publicly available datasets from UTBM [28], URBAN [29], NCLT [30], and LIOSAM (from the LIO-SAM [8] dataset), with detailed information provided in Table I. These challenging datasets are widely used for benchmarking state-of-the-art methods. Due to variability in ground truth data caused by factors such as weather conditions and GPS quality, we selected 13 sequences with reliable ground-truth trajectories from these datasets. These datasets cover various scenarios, including autonomous driving, urban environments, and campus settings, with the longest distance reaching 5.1 km and the longest duration being 111 minutes.

TABLE I
Details of All Dataset Sequences

	Name	Duration (min:sec)	Distance (km)
utbm1	20180719	15:26	4.98
utbm2	20180717	15:59	4.99
utbm3	20180713	16:59	5.03
utbm4	20190418	11:59	5.11
urban1	20190428	8:07	2.00
urban2	20210517	13:05	3.64
urban3	20210521	25:36	4.51
nclt1	20120115	111:46	4.01
nclt2	20120429	43:17	1.86
nclt3	20120511	84:32	3.13
nclt4	20120615	55:10	1.62
nclt5	20130110	17:02	0.26
lio-sam1	park	9:11	0.66

Implementation Details. We evaluated the accuracy and efficiency of our proposed LIO-DPC method by comparing it with state-of-the-art LiDAR-inertial odometry techniques. The comparison includes filtering-based methods FAST-LIO2 [2] and FASTER-LIO [31], as well as factor graph optimization-based methods LILI-OM [26] and LIO-SAM [8]. All experiments were performed on a computer equipped with an AMD R9-5950X CPU (16 cores at 3.4GHz) and 128GB of RAM. Performance was evaluated using the Root Mean Square Error (RMSE) of the Absolute Pose Error (APE) over the entire trajectory. Notably, LIO-SAM could not operate on UTBM datasets due to missing attitude quaternion data (denoted by $-$), and significant drift led to algorithm failures in some sequences (denoted by $\times$).

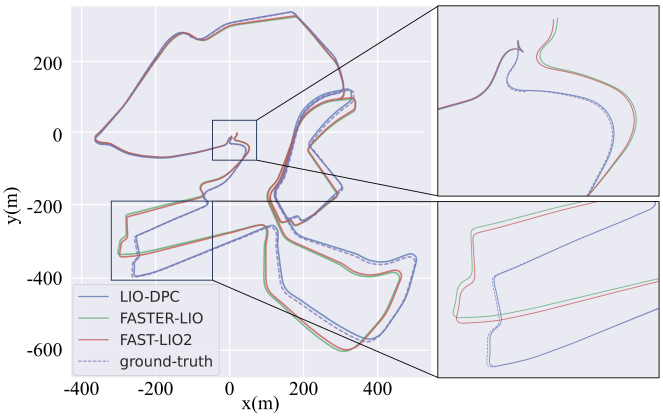


Fig. 2. Ground-Truth and Estimated Trajectories on the UTBM Dataset: Dashed Lines Indicate Ground-Truth, Solid Lines Represent Estimated Results.

B. Accuracy Evaluation

The RMSE results, shown in Table II, demonstrate that LIO-DPC consistently achieves lower RMSE compared to other state-of-the-art methods across all datasets. Compared to FAST-LIO2 and FASTER-LIO, LIO-DPC utilizes batch updates, significantly reducing cumulative errors and improving system accuracy. Additionally, unlike LILI-OM, LIO-DPC employs loop sparsification to filter out low-quality loop closures, effectively minimizing loop-related errors.

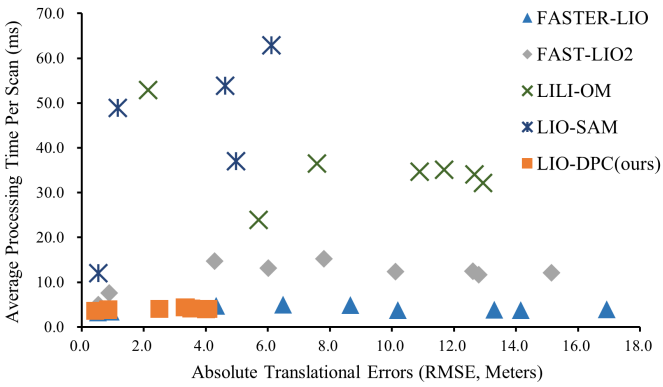


Fig. 3. Comparison of Absolute Translational Errors (RMSE, Meters) and Average Processing Time Per Scan (Milliseconds).

A trajectory comparison within the UTBM dataset sequence is shown in Fig.2, highlighting the significant drift experienced

TABLE II
ABSOLUTE TRANSLATIONAL ERRORS (RMSE, METERS) FOR SEQUENCES WITH HIGH-QUALITY GROUND TRUTH

	utbm1	utbm2	utbm3	utbm4	urban1	urban2	urban3	nclt1	nclt2	nclt3	nclt4	nclt5	liosam1
FAST-LIO2	15.15	12.61	12.80	10.11	4.28	7.81	6.01	1.89	1.56	2.51	2.00	0.89	0.53
FASTER-LIO	16.93	13.30	14.15	10.19	4.33	8.66	6.49	2.06	1.34	2.27	1.79	0.93	0.52
LILI-OM	10.89	12.66	12.94	11.68	35.51	7.58	5.70	×	×	×	×	2.14	21.86
LIO-SAM	–	–	–	–	1.16	6.12	4.62	×	×	×	×	4.98	0.54
LIO-DPC(ours)	**3.36**	**3.53**	**3.32**	**2.50**	**0.86**	**4.08**	**4.01**	**1.88**	**1.30**	**1.41**	**1.57**	**0.85**	**0.44**

TABLE III
AVERAGE PROCESSING TIME PER SCAN (MILLISECONDS)

	utbm1	utbm2	utbm3	utbm4	urban1	urban2	urban3	nclt1	nclt2	nclt3	nclt4	nclt5	liosam1
FAST-LIO2	12.14	12.47	11.70	12.37	14.67	15.18	13.10	9.24	9.38	8.80	9.77	7.57	4.95
FASTER-LIO	**3.96**	**3.81**	**3.75**	**3.80**	4.76	4.88	4.95	4.82	**4.41**	**4.00**	**4.17**	**3.42**	**3.26**
LILI-OM	34.70	34.05	32.16	35.14	36.66	36.47	23.93	×	×	×	×	52.89	53.88
LIO-SAM	–	–	–	–	48.88	62.88	53.86	×	×	×	×	37.01	12.02
LIO-DPC(ours)	4.35	4.14	4.33	4.04	**3.98**	**4.04**	**3.96**	**4.40**	4.43	4.26	4.41	3.74	3.55

by FAST-LIO2 and FASTER-LIO during operation. In contrast, LIO-DPC effectively mitigates these issues through loop closure detection and graph optimization, which systematically eliminate cumulative errors and maintain higher accuracy.

C. Processing Time Evaluation

As shown in Table III, we evaluate processing performance by measuring the average time per LiDAR frame. Compared to LILI-OM and LIO-SAM, LIO-DPC leverages a dynamic pose chain with incremental updates, achieving significantly better real-time performance. Despite integrating loop closure and local pose graph optimization, LIO-DPC maintains a processing speed comparable to the state-of-the-art filtering method FASTER-LIO. Moreover, batch optimization and loop closure allow LIO-DPC to achieve notably higher accuracy than FASTER-LIO. The dynamic pose chain enables parallel filtering and optimization, ensuring both efficiency and accuracy. Fig. 3 further illustrates that LIO-DPC achieves competitive speed while significantly outperforming other methods in accuracy across various datasets.

TABLE IV
ACCURACY EVALUATION ABLATION EXPERIMENT(RMSE, METERS)

	LIO-DPC	LIO-DPC (without sparsification)	LIO-DPC (without optimization)
utbm1	**3.36**	7.36	17.00
utbm2	**3.53**	4.95	13.26
utbm3	**3.32**	3.41	14.10
utbm4	**2.50**	5.72	11.80
urban1	**0.86**	1.46	4.58
urban2	**4.08**	6.19	8.40
urban3	**4.01**	5.01	6.89

D. Ablation Experiments

In this section, we conduct two ablation studies within the LIO-DPC framework to evaluate the effects of loop closure detection and loop sparsification on system accuracy and computational efficiency. We define LIO-DPC (without optimization) as the system without loop detection and local pose graph construction, and LIO-DPC (without sparsification) as the system without loop sparsification. Experiments were performed on the UTBM and URBAN datasets using these variants alongside the original LIO-DPC method. The accuracy results are presented in Table IV, and the total graph optimization time results are shown in Fig.4.

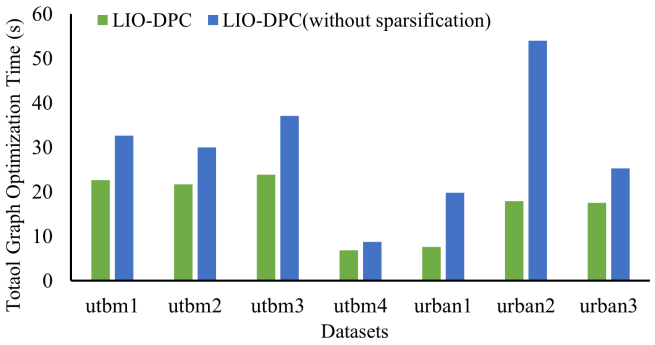


Fig. 4. Total Graph Optimization Time in Seconds for Different Dataset Sequences.

Compared to LIO-DPC (without optimization), the original LIO-DPC significantly improves system accuracy. This demonstrates that integrating filtering methods with pose graph optimization via the dynamic pose chain is both practical and efficient. Furthermore, compared to LIO-DPC (without sparsification), the original LIO-DPC not only achieves higher accuracy but also significantly reduces the total computation time of graph optimization. This indicates that our loop quality evaluation metric effectively selects high-quality loops, reducing the computational burden of graph optimization and minimizing loop closure errors.

V. CONCLUSIONS

We propose LIO-DPC, a LiDAR-inertial odometry system that integrates filtering and graph optimization via a dynamic pose chain. The pose chain is incrementally updated in real-time through filtering and batch refined with graph optimization, enabling fast and accurate pose estimation. To enhance efficiency, we introduce a loop quality metric that selects only high-quality loops for optimization, reducing computation while improving accuracy. Real-world experiments show that LIO-DPC achieves higher accuracy than state-of-the-art methods while maintaining real-time performance.

VI. ACKNOWLEDGMENT

This work was supported by the Fundamental Research Funds for the Central Universities under Grant (Project No. 2024CDJGF-032, 2023CDJXY-039, 2024CDJGF003, 2024CDJGF-019, 2023CDJKYJH026).

REFERENCES

[1] Y. Wu, T. Guadagnino, L. Wiesmann, L. Klingbeil, C. Stachniss, and H. Kuhlmann, "Lio-ekf: High frequency lidar-inertial odometry using extended kalman filters," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 13 741–13 747.

[2] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.

[3] H. Ye, Y. Chen, and M. Liu, "Tightly coupled 3d lidar inertial odometry and mapping," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 3144–3150.

[4] D. Lee, M. Jung, W. Yang, and A. Kim, "Lidar odometry survey: recent advancements and remaining challenges," *Intelligent Service Robotics*, vol. 17, no. 2, pp. 95–118, 2024.

[5] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[6] W. Xu and F. Zhang, "Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3317–3324, 2021.

[7] C. Qin, H. Ye, C. E. Pranata, J. Han, S. Zhang, and M. Liu, "Lins: A lidar-inertial state estimator for robust and efficient navigation," in *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2020, pp. 8899–8906.

[8] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2020, pp. 5135–5142.

[9] P. Geneva, K. Eckenhoff, Y. Yang, and G. Huang, "Lips: Lidar-inertial 3d plane slam," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 123–130.

[10] J. Tang, X. Zhang, Y. Zou, Y. Li, and G. Du, "A high-precision lidar-inertial odometry via kalman filter and factor graph optimization," *IEEE Sensors Journal*, vol. 23, no. 11, pp. 11 218–11 231, 2023.

[11] P. J. Besl and N. D. McKay, "Method for registration of 3-d shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.

[12] T. Shan, B. Englot, C. Ratti, and D. Rus, "Lvi-sam: Tightly-coupled lidar-visual-inertial odometry via smoothing and mapping," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 5692–5698.

[13] W. Ding, S. Hou, H. Gao, G. Wan, and S. Song, "Lidar inertial odometry aided robust lidar localization system in changing city scenes," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4322–4328.

[14] S. Zhao, Z. Fang, H. Li, and S. Scherer, "A robust laser-inertial odometry and mapping method for large-scale highway environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1285–1292.

[15] Z. Wang, L. Zhang, Y. Shen, and Y. Zhou, "D-liom: Tightly-coupled direct lidar-inertial odometry and mapping," *IEEE Transactions on Multimedia*, vol. 25, pp. 3905–3920, 2023.

[16] X. Zuo, P. Geneva, W. Lee, Y. Liu, and G. Huang, "Lic-fusion: Lidar-inertial-camera odometry," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 5848–5854.

[17] N. Carlevaris-Bianco, M. Kaess, and R. M. Eustice, "Generic node removal for factor-graph slam," *IEEE Transactions on Robotics*, vol. 30, no. 6, pp. 1371–1385, 2014.

[18] A. Cunningham, M. Paluri, and F. Dellaert, "Ddf-sam: Fully distributed slam using constrained factor graphs," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3025–3030.

[19] V. Bichucher, J. M. Walls, P. Ozog, K. A. Skinner, and R. M. Eustice, "Bathymetric factor graph slam with sparse point cloud alignment," in *OCEANS 2015 - MTS/IEEE Washington*, 2015, pp. 1–7.

[20] J. Vallvé, J. Solà, and J. Andrade-Cetto, "Graph slam sparsification with populated topologies using factor descent optimization," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1322–1329, 2018.

[21] P. Chauchat, A. Barrau, and S. Bonnabel, "Factor graph-based smoothing without matrix inversion for highly precise localization," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 1219–1232, 2021.

[22] S. Bai, W. Wen, L.-T. Hsu, and P. Yang, "Factor graph optimization-based smartphone imu-only indoor slam with multi-hypothesis turning behavior loop closures," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–21, 2024.

[23] J. Tang, Y. Chen, X. Niu, L. Wang, L. Chen, J. Liu, C. Shi, and J. Hyyppä, "Lidar scan matching aided inertial navigation system in gnss-denied environments," *Sensors*, vol. 15, no. 7, pp. 16 710–16 728, 2015.

[24] W. Zhen, S. Zeng, and S. Soberer, "Robust localization and localizability estimation with a rotating laser scanner," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 6240–6245.

[25] C. Le Gentil, T. Vidal-Calleja, and S. Huang, "In2lama: Inertial lidar localisation and mapping," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6388–6394.

[26] K. Li, M. Li, and U. D. Hanebeck, "Towards high-performance solid-state-lidar-inertial odometry and mapping," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5167–5174, 2021.

[27] T. P. Setterfield, R. A. Hewitt, A. T. Espinoza, and P.-T. Chen, "Feature-based scanning lidar-inertial odometry using factor graph optimization," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3374–3381, 2023.

[28] Z. Yan, L. Sun, T. Krajník, and Y. Ruichek, "Eu long-term dataset with multiple sensors for autonomous driving," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 10 697–10 704.

[29] L.-T. Hsu, F. Huang, H.-F. Ng, G. Zhang, Y. Zhong, X. Bai, and W. Wen, "Hong kong urbannav: An open-source multisensory dataset for benchmarking urban navigation algorithms," *NAVIGATION: Journal of the Institute of Navigation*, vol. 70, no. 4, 2023.

[30] N. Carlevaris-Bianco, A. K. Ushani, and R. M. Eustice, "University of michigan north campus long-term vision and lidar dataset," *The International Journal of Robotics Research*, vol. 35, no. 9, pp. 1023–1035, 2016.

[31] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao, "Faster-lio: Lightweight tightly coupled lidar-inertial odometry using parallel sparse incremental voxels," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4861–4868, 2022.

Configurable DSP-Based CAM Architecture for Data-Intensive Applications on FPGAs

Yao Chen*, Feng Yu*, Di Wu†, Weng-Fai Wong*, Bingsheng He*
*National University of Singapore, Singapore, †University of Science and Technology of China, Hefei, China
{yaochen, wongf, dcsheb}@nus.edu.sg, yuf@u.nus.edu, ddiwu@mail.ustc.edu.cn

***Abstract*—Content-addressable memory (CAM) is a type of fast memory unique in its ability to perform parallel searches of stored data based on content rather than specific memory addresses. They have been used in many domains, such as networking, databases, and graph processing. Field-programmable gate arrays (FPGAs) are an attractive platform for implementing CAMs because of their low latency, reconfigurability, and energy-efficient nature. However, such implementations also face significant challenges, including high resource utilization, limited scalability, and suboptimal performance due to the extensive use of look-up tables (LUTs) and block RAMs (BRAMs). These issues stem from the inherent limitations of FPGA architectures when handling the parallel operations required by CAMs, often leading to inefficient designs that cannot meet the demands of high-speed, data-intensive applications. To address these challenges, we propose a novel configurable CAM architecture that leverages the digital signal processing (DSP) blocks available in modern FPGAs as the core resource. By utilizing DSP blocks' data storage and logic capabilities, our approach enables configurable CAM architecture with efficient multi-query support while significantly reducing search and update latency for data-intensive applications. The DSP-based CAM architecture offers enhanced scalability, higher operating frequency, and improved performance compared to traditional LUT and BRAM-based designs. In addition, we demonstrate the effectiveness of our proposed CAM architecture with a triangle counting application on real graphs. This innovative use of DSP blocks also opens up new possibilities for high-performance, data-intensive applications on FPGAs. Our proposed design is open-sourced at: https://github.com/Xtra-Computing/DSP_CAM/.**

***Index Terms*—Content addressable memory, digital signal processor, FPGA, scalability, high-performance**

I. Introduction

Content-addressable memory (CAM) performs fast content matching across all stored entries in a single operational cycle [8], [12], [16]. Unlike traditional memory, which requires sequential access or indexing methods that introduce significant latency, CAM's processing pattern significantly accelerates search-intensive operations common in data-intensive applications such as graph analytics, network processing, and database query acceleration [5].

Field-programmable gate arrays (FPGAs) are widely adopted as accelerator platforms for data-intensive applications, driving the demand for CAM designs on it. Due to the reconfigurability advantage, the CAM implementations on FPGA emulate all CAM types, including binary, ternary, and range-matching CAMs.

Current CAM implementations on FPGAs are typically based on Lookup Tables (LUTs), Block RAMs (BRAMs), or hybrid designs combining both, but all face significant challenges, especially in scalability and performance, as shown in Figure 1. The main challenge of LUT-based CAMs is on how to achieve very good search performance while efficiently utilizing the LUT resources [7], [18]. While LUTRAM-based designs optimize LUT usage, they impose additional preprocessing overhead for input data, complicating updates and limiting their suitability for dynamic applications [10], [16], [20]. BRAM-based CAMs, on the other hand, leverage the large storage capacity of Block RAMs (BRAM) [3], [10], but BRAM blocks are designed for sequential access, requiring additional logic for parallel comparisons, which reduces search performance and achievable clock frequency [4], [17]. Hybrid-resource CAMs attempt to balance the flexibility of LUTs with the storage efficiency of BRAMs but encounter complex update processes due to the overhead of managing multiple resource types [4], [9]. The Digital Signal Processing (DSP) slices in FPGAs have been explored as the primary resource for CAM design due to their superior achievable clock frequency compared to LUT and ability to be configured as a logical operation unit. Repurposing the DSP slice for CAM may also help us to reduce the reliance on LUTs and BRAMs, alleviating resource constraints with enhanced performance. The existing DSP-based CAM design has demonstrated benefits, such as higher operating frequency and optimized data pathways. However, it remains challenged by prolonged search latency, rendering it less suitable for data-intensive applications [14]. Moreover, the intrinsic computational demands associated with executing parallel comparisons within CAMs exacerbate existing challenges, rendering these designs suboptimal for data-intensive applications that necessitate scalability, frequent updates, and multiple concurrent query searches.

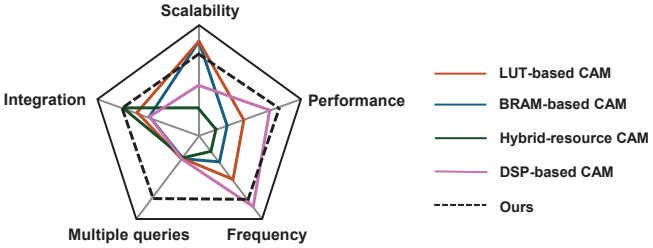


Fig. 1: The Characteristics of current FPGA-based CAM designs. Scalability denotes the achieved CAM size. Performance represents the normalized search and update latency. Frequency is the maximum achievable clock frequency. Integration demonstrates the level of easy integration to an application. Multiple queries stand for the concurrent support to multiple input queries.

This paper presents a novel configurable design of high-performance CAM with DSP slices as the primary resource. The proposed CAM architecture addresses the need for flexible and efficient data updates and searches in data-intensive applications and also provides easy integration to accelerator systems. The main contributions are listed as follows.

- **Configurable DSP-based CAM architecture**: We introduce a fully parameterized CAM design that mainly uses DSP slice and can be customized at different granularities across cell, block, and unit levels.
- **Multi-query support**: Our CAM architecture supports multiple content searches in parallel with an optimized search logic.
- **Advantageous experimental results**: Our proposed design on FPGA demonstrates superior performance in different scales.

†This work was done when Di Wu was a visiting student at NUS.

TABLE I: A survey of recent CAM designs on FPGA

Design Name	Category	Platform	Max CAM Size	Frequency (MHz)	Resource Utilization			Latency (cycles) ‖	
					LUT	BRAM	DSP	Update	Search
Scale-TCAM [10]	LUT	XC7V2000T	4096 × 150 bits	139	322648 *	0	0	33	-
DURE [16]	LUT	Xilinx Virtex-6	1024 × 144 bits	175	35807	0	0	65†	1†
BPR-CAM [15]	LUT	XC6VLX760	1024 × 144 bits	111	15260	0	0	-	2
Frac-TCAM [20]	LUT	XC7V2000T	1024 × 160 bits	357	16384	0	0	38	-
HP-TCAM [19]	BRAM	Xilinx Virtex-6	512 × 36 bits	118	5326	56	0	-	5
PUMP-CAM [17]	BRAM	XC6VLX760	1024 × 140 bits	87	7516	80	0	129	-
IO-CAM [13]	BRAM	Intel Arria V 5ASTD5	8192 × 32 bits	135	19017 ‡	2112§	0	-	-
REST-CAM [4]	Hybrid	Xilinx Kintex-7	72 × 28 bits	50	130	1	0	513	5
Preußer, et.al [14]	DSP	XCVU9P	1000 × 24 bits	350	2843	0	1022	-	42
Ours	DSP	U250	9728 × 48 bits	235	72178	4	9728	6	8

‖ The latency is measured for a single end-to-end operation. * The number of LUT is calculated by the number of slices (80662) multiplied by four. † The update and search latencies are measured on a single CAM block with dimensions of 512 x 36. ‡ The number of used ALM in Intel FPGA. § The number of used M10K in Intel FPGA. - indicates the value is not reported in the literature.

- **System case study**: We demonstrate the effectiveness and integrability of our proposed CAM design in the context of triangle counting acceleration system in graphs, showcasing improved efficiency and scalability.

The remainder of this paper is organized as follows. Section II identifies the inefficiencies of current CAM on FPGAs for data-intensive applications and motivates our proposed design. Section III presents our proposed configurable DSP-based CAM architecture. Section IV evaluates our design and demonstrates its advantage over the existing ones. A case study with triangle counting on real-world graphs is presented in Section V and Section VI concludes this paper.

II. Background

CAMs can be categorized into Binary CAM, Ternary CAM (TCAM), and Range-Matching CAM (RMCAM). Binary CAMs perform exact-match searches using binary data (0s and 1s), making them suitable for applications like cache memory tag matching where precise data retrieval is essential [2], [3], [12], [18]. Ternary CAMs (TCAMs) introduce a third state, the "don't care" (X) condition, allowing for partial or wildcard matching, and is widely used in IP routing or packet redirection. Range-matching CAMs are designed to match input data within specific numerical ranges, which is valuable in applications like database indexing and firewall rule matching. Recent representative designs are compiled in Table I. FPGAs possess a unique capability to emulate all forms of CAM designs due to their inherent logical representation functionality. However, the pronounced variations in resource types available within FPGAs necessitate categorizing CAM designs based on the primary resources utilized: LUTs, BRAMs, DSPs, and combinations thereof.

A. Challenges When Facing Data-Intensive Applications

Data-intensive applications often require frequent data updates and rapid data searches, which raise challenges for the underlying CAM architecture. The challenges observed include:

Limited Scalability. In traditional CAM implementations that adopt LUT and BRAM [10], [13], [15]–[17], as the size of the CAM increases — both the number of entries and the width of each entry — the number of required LUT or BRAM grows exponentially. Consequently, the implemented timing drops significantly, which become impractical for data-intensive tasks like graph processing.

High Search and Update Latency. Many CAM architectures [4], [10], [16], [17], [20] are optimized for read-intensive operations with infrequent updates, which is not suitable for applications that require frequent data modifications. Frequent updates result in increased latency and create bottlenecks that reduce the system's data processing efficiency. The slow update rates and the complexity of update logic hinder the performance of applications that need immediate reflection of data changes, such as dynamic graph algorithms.

Multiple Concurrent Queries. Multi-queries are an essential processing pattern in data-intensive applications. However, CAMs on FPGAs rely heavily on resources like Lookup Tables (LUTs) to perform simultaneous comparisons across all stored entries, which makes them incapable of dealing with multiple search requests from different input data. Even in the most recent design that adopts DSP as the processing resource, multi-query support remains unexplored.

System Integration Complexity. Data-intensive applications on FPGAs commonly necessitate domain-specific architectures. In such contexts, CAM implementations must coexist with other system components to achieve overall accelerator functionality. Consequently, the configurability and integrability of the CAM microarchitecture should be explicitly considered during the design process.

B. Design Motivations

The challenges outlined above motivate a reevaluation of CAM design on FPGA for data-intensive applications.

Leveraging DSP Blocks for better scalability. DSP blocks are specialized units optimized for high-speed arithmetic and logic operations. By repurposing the DSP slice for CAMs, we may reduce reliance on LUTs and BRAMs, alleviating resource constraints and enhancing performance.

Balancing update and search latency. Data-intensive applications require more frequent data updates, which necessitates a more balanced update and search logic path design.

Multi-query support. Runtime dynamic data path control in the CAM unit is necessary to address the multiple search requests for data-intensive applications.

Configurable architecture. The variation of the applications necessitates a flexible CAM architecture that can accommodate these differing needs without completely redesigning the entire architecture.

In summary, using DSP blocks as the primary resource in our CAM design effectively addresses resource utilization and scalability issues found in traditional FPGA-based CAMs. Multi-query support adds flexibility to accommodate dynamic data and application demands, enhancing both efficiency and responsiveness. Furthermore, a parameterized architectural design facilitates adaptation for broader application acceptance.

III. Configurable DSP-based CAM Architecture

With the motivations above, our configurable CAM design on FPGA adopts the DSP slice as the primary resource and is designed

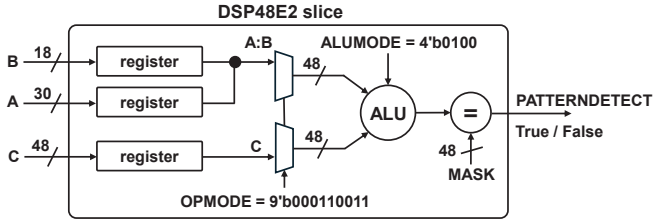


Fig. 2: DSP-based CAM cell.

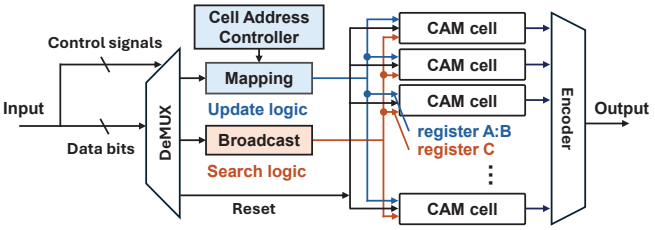


Fig. 3: Architecture of DSP-based CAM block.

hierarchically in a fully parameterized manner. The CAM cell, configured using a DSP slice, provides basic data storage and data comparison functionalities. Multiple CAM cells are grouped to form a CAM block, which integrates data update and search logic, serving as the core component for executing essential CAM operations. Our proposed configurable CAM unit is constructed by combining multiple CAM blocks with additional logic for data updates and parallel searches. This CAM unit can operate as multiple independent CAMs, with each CAM sized at the granularity of a CAM block, providing exceptional flexibility. The CAM unit also supports multiple search queries across multiple grouped CAM blocks, significantly enhancing processing throughput and parallelism.

A. Configurable CAM cell with DSP

The DSP48E2 [6] slice is designed to provide arithmetic processing capacity, which contains an arithmetic unit, logic and comparison unit, shift and preprocessing unit, registers, control, and configurable logic, cascading and interconnection. Turning the DSP slice into a CAM cell requires the configuration of the DSP block into logic processing mode. Specifically, the operational mode of the DSP slice is determined by configuring the ALUMODE and OPMODE registers, as shown in Figure 2, enabling the slice to perform XOR operations between the two 48-bit registers:

$$O = (A : B) \oplus C \tag{1}$$

With this configuration, registers A and B in each DSP48E2 slice are concatenated to provide a maximum 48-bit storage space for input data, while register C is used to hold the search key during search operations. For Ternary CAM (TCAM) and Range-Matching CAM (RMCAM) functionality, the additional MASK function is employed to handle partial matches and range checks. By setting specific bits in the mask to "don't care" (X) status, the DSP performs post-processing after the XOR operation to identify matches based on the masked conditions. However, in Range-Matching CAM, the representation is limited to ranges where the start and end values are powers of 2. This limitation arises from the bit-level granularity of the mask control, restricting the range specifications' flexibility. Table II summarizes the MASK definitions and their effects for binary, ternary, and range-matching CAMs. The mask is also used for the data bit width control, where the unused data bits are masked out to simplify the data path.

TABLE II: MASK value for CAM type configuration

Type	MASK value	Behavior
BCAM	All bits are zero	All bits are compared
TCAM	Active bits = 0, ignored bits = 1	Bits with MASK = 1 are "don't care" bits
RMCAM	Relevant bits = 0, others = 1	Bits with MASK = 0 are selected range

B. CAM block microarchitecture

A DSP-based CAM cell can not perform the update and search operations as it only provides the data storage and comparison logic. The design of a CAM block involves organizing multiple CAM cells along with additional control logic to support search and update operations. Specifically, a CAM block in our design includes a configurable number of CAM cells, a DeMUX, an update and search logic, and an output Encoder, as shown in Figure 3.

The input bus for the CAM block comprises both data bits and control signals that include update, search, and reset. When a new input data packet arrives, the DeMUX routes the data bits to the update logic or the search logic based on the control signals. For update operations, the update logic uses a mapping function based on the Cell Address Controller to direct each data word (with storage data width) in the input data bus to the appropriate CAM cell in the block. Since the input data bits may consist of multiple storage-size data words due to a larger customized input data width than the storage data width in CAM cell, this approach enables the parallel updating of multiple CAM cells in a single operation. For search operations, the redundant bits in the input data bits are masked to ensure that only one data word serves as the search key within the CAM block. The search logic then broadcasts this key to all CAM cells for parallel comparisons. The Encoder collects the match results and generates the final output, supported by a configurable encoding scheme suited to various addressing and management strategies. When the reset signal is asserted, all stored contents in each CAM cell are cleared.

C. CAM unit microarchitecture

The CAM unit consists of multiple CAM blocks, supplemented by additional control and routing logic to facilitate multi-query operations. As illustrated in Figure 4, the overarching microarchitecture includes a Routing Compute module and a Post-Router module, alongside multiple CAM blocks. These components are encapsulated by input and output interfaces that communicate with the user kernel.

1) CAM Group: We define a CAM group as a logical abstraction that forms the foundation for executing search and update operations within the CAM unit. A CAM group consists of multiple CAM blocks and is not tied to the physical layout, enabling flexible configuration and dynamic reassignment of resources. Let N_{total} denote the total number of CAM blocks in the CAM unit and M the number of CAM groups. Each group then contains $N = \frac{N_{\text{total}}}{M}$ blocks; M needs to be divisible by N. The number of CAM groups M is configured at runtime by the user kernel, thereby enabling the CAM unit to dynamically adapt to varying workload demands, specifically catering to multi-query parallelism requirements.

2) Update Process: During the update process, the Routing Table, implemented as an array within the Routing Compute module, stores the mapping relationships between Block IDs and Group IDs. This

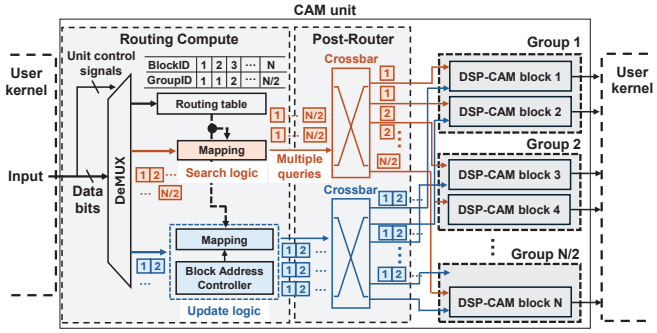


Fig. 4: Architecture of DSP-based CAM unit.

TABLE III: Configurable Parameters for CAM Unit

Granularity	Parameter	Description
CAM Cell	Cell type	The type of CAM: Binary, Ternary, Range-matching.
	Storage Data Width	Width of the stored data ($\leq$ 48 bits).
CAM Block	Block Size	Number of cells per CAM block.
	Block Bus Width	Data path width in CAM block.
	Result Encoding	Encoding scheme for search results from multiple CAM cells.
CAM Unit	Unit Size	Number of CAM blocks per CAM unit.
	Unit Bus Width	Data path width in CAM unit.

array shares the same data path as the input update data, where a control signal is used to update the mapping configuration. Based on the mapping relationship defined in the Routing Table, the input data is replicated and directed to all M CAM groups, ensuring that the update is applied globally. Subsequently, the update crossbar in the Post-Router module routes the replicated data to the designated block in each group. Within each CAM group, the Block Address Controller sequentially assigns data to each block using a round-robin policy; once the current block is full, the controller points to the next block in the group.

3) Search Process: In the search process, each incoming search key is first processed by the Routing Compute module, which employs a mapping function based on the Routing Table to allocate the key to the appropriate CAM group. Once a group is selected, the search key is replicated N times (where N is the number of blocks within that group) and broadcast to every block in the group, thereby enabling parallel comparison across all blocks. The Post-Router module then delivers the duplicated keys to the corresponding blocks, initiating parallel search operations. With each search key assigned to a distinct CAM group, the overall CAM unit is capable of supporting up to M search queries per cycle.

4) Example: Consider the configuration shown in Figure 4, where each CAM group consists of two blocks. During the update process, each incoming data packet is duplicated across all $M = N_{\text{total}}/2$ groups and written sequentially: data is first stored in the first block of each group, and upon reaching capacity, subsequent entries are directed to the second block following a round-robin schedule. In the search phase, the CAM unit supports up to M concurrent search keys, each routed to a distinct CAM group. Within each group, the key is broadcast to both blocks, enabling parallel comparison and efficient multi-query processing.

D. Parameterized architectural components

To enable seamless integration into different applications, our CAM unit is fully parameterized with different hierarchies of configurations. The configurations include cell-level, block-level, and unit-level. As shown in Table III, parameters such as cell type, data width, block size, unit size, and bus width are configured at the design stage, ensuring optimal resource utilization and hardware compatibility. We design the source file in templates where all the parameters can be defined before the CAM unit is generated.

IV. CAM Evaluations

To comprehensively evaluate the effectiveness of our proposed parallel CAM design, metrics are analyzed at three different granularities: CAM cell, CAM block, and CAM unit. We assess performance, efficiency, and scalability at each granularity to provide a detailed understanding of the architecture's characteristics. Across all granularities of evaluation, the key metrics include:

- **Latency**: Measured in clock cycles, indicates the time required for an end-to-end operation.
- **Throughput**: The number of operations (updates or searches) performed per second.
- **Resource Utilization**: Quantifies the FPGA resources that are consumed, including LUTs, FFs, DSP, and BRAMs.
- **Scalability**: Reflects the ability to adapt the design to a larger size without significant performance degradation.

The experiments were conducted on an AMD Alveo U250 accelerator card, an FPGA platform based on the UltraScale+ architecture. The U250 features four DDR4 memory channels and a PCIe Gen3 x16 interface, providing ample external memory bandwidth for data-intensive applications. The resources of the chip are collected in Table IV. The results are collected through implementation using Xilinx Vivado Design Suite v2021.2, and the CAM architecture was implemented and deployed on the U250. Resource utilization metrics include LUTs, BRAMs, and DSPs.

TABLE IV: Resource capacity of AMD Alveo U250

Resource	LUTs	Registers	BRAM	URAM	DSP
Quantity	1,728K	3,456K	2,688	1,280	12,288

A. CAM Cell Evaluation

Taking advantage of the reconfigurability of the FPGA DSP slice, our CAM cell is compact and flexible for different CAM functionalities. The configuration of the OPMODE and ALUMODE does not change the resource utilization of the memory cell. As shown in Table V, the resource consumption, update, and search latency stay the same for the configuration of Binary CAM cell, Ternary CAM cell, and Range-matching CAM cells. This provides the basis for a scalable and high-performance CAM unit design.

TABLE V: CAM Cell Evaluation

Metric	Value
Storage Capacity	1 entry $\leq$ 48 bits
Update Latency	1 cycle
Search Latency	2 cycles
Resource Utilization	1 DSP, 0 LUT, 0 BRAM

B. CAM Block Evaluation

Due to the stable characteristic of our DSP-based CAM cell design, at the granularity of the CAM block, we mainly focus on

evaluating the block configuration with scaled numbers of CAM cells for storage of data width no more than 48 bits. The results are shown in Table VI. We set the size of the block to be power-of-two values to maintain a hardware-friendly architecture, which are also the commonly used values for the size of a cache line. Notably, the Update Throughput and Search Throughput are measured in data update or search operations per second, which is different from the normally used processed packet per second. Our CAM design maintains high frequency and stable update and search latency with different size configurations. Specifically, when the size of the block reaches 256, we added an additional buffer at the Encoder output to optimize the implementation timing. It leads to an increased search latency of 4 but does not harm the search throughput. Our LUT utilization remains very low when compared to designs in Table I, this leaves more resources for the other parts of the system.

TABLE VI: CAM Block Evaluation with different size

CAM size	32	64	128	256	512
Update Latency (cycle)	1	1	1	1	1
Search Latency (cycle)	3	3	3	4	4
Update Throughput (op/s)	4800	4800	4800	4800	4800
Search Throughput (op/s)	300	300	300	300	300
# of LUTs	694	745	808	1225	1371
Utilization(%)	0.05	0.05	0.05	0.07	0.08
# of DSP	32	64	128	256	512
Utilization(%)	0.26	0.52	1.04	2.08	4.17
BRAM Utilization	0	0	0	0	0
Frequency(MHz)	300	300	300	300	300

C. CAM Unit Evaluation

a) Scalability evaluation: To evaluate the effectiveness of our proposed microarchitecture at managing the scaled number of CAM blocks, except for the basic metrics, the evaluations are extended to the performance of randomly updating and searching a single value in the CAM unit. Specifically, the size of the CAM block is set to 256, and the Input Bus Width is 512, which is to be compatible with the interface width of the external DDR memory port in our evaluation platform. The number of adopted DSP blocks increases when the CAM size increases. We maintain the data to be 48 bits so all the internal data paths are active.

The results are collected in Table VII. The required number of LUT increases linearly when the size of the CAM unit increases, this is due to the logic required by the data update and search logic. With the given 11,508 DSPs on our platform, we can easily achieve a CAM size that reaches $9K \times 48$ bits, where the 79.25% of the DSPs are adopted for this CAM unit but only 2.92% of the LUT resource is used. As a result, we achieve a 254MHz clock frequency. The update and search performance are collected in Table VIII, where the data width is configured as 32 bits for wider adoption. The update latency does not change when the CAM unit size changes, which is mainly because of the simpler datapath for updates. The search latency increases by one clock cycle when the CAM size is larger than 2K, which is mainly due to the buffering in the encoder of the CAM block to optimize the timing during implementation. The update and search throughput only relate to the clock frequency since the processes are pipelined with an initial interval of 1.

b) Comparison to the state-of-the-arts: We incorporate our design with its maximum configuration into Table I, which includes state-of-the-art designs at their respective maximum configurations. It is worth noting that 4 BRAMs are utilized by the bus interfaces for FIFOs, which we add to facilitate complete synthesis and implementation. Our CAM design exhibits superior scalability, occupying 79.25% of the on-chip DSP resources while utilizing significantly fewer LUTs. Additionally, when compared with existing DSP-based CAM designs, the update and search latency of our proposed CAM is well-balanced and more conducive to data-intensive applications.

TABLE VII: CAM Unit Configuration and Resource Utilization

CAM size	LUT Utilization	DSP Utilization	Freq. (MHz)
512 × 48 bits	2491	512	300
1024 × 48 bits	5072	1024	300
2048 × 48 bits	10167	2048	300
4096 × 48 bits	20330	4096	265
6144 × 48 bits	29385	6144	252
8192 × 48 bits	38191	8192	240
9728 × 48 bits	45244	9728	235

TABLE VIII: CAM Performance for 32-bit data with different sizes

CAM size	128	512	2048	4096	8192
Update Latency (cycle)	6	6	6	6	6
Search Latency (cycle)	7	7	8	8	8
Update Throughput (op/s)	4800	4800	4800	4064	3840
Search Throughput (op/s)	300	300	300	254	240

V. Case Study: Triangle Counting with CAM

To demonstrate the effectiveness of our proposed CAM design, we implement an accelerator that primarily relies on the CAM to perform triangle counting. Furthermore, we conduct this evaluation to highlight both the seamless integration of our CAM design into the accelerator and the enhanced functionality enabled by the CAM.

A. Preliminary of triangle counting

Triangle counting is a fundamental operation in graph analytics, widely used for calculating clustering coefficients and understanding transitivity in networks across domains such as social sciences, biology, and computer science. A typical edge-centric triangle counting algorithm involves retrieving the adjacency lists of two connected vertices and performing a set intersection to identify common neighbors, as shown in Figure 5. Due to the compact storage format of graph data, which are generally in the Compressed Sparse Row (CSR) format, this operation relies heavily on set intersection on two random length lists, a typical data-intensive task that poses significant challenges to common address-based memory systems, particularly for large-scale and irregular graph data. Two primary challenges arise from this operation. First, the inherent sequential nature of the set-intersection operation makes it difficult to perform efficient parallel processing between two input sets. Second, due to the random lengths of adjacency lists, traditional memory systems are poorly suited to the unpredictable and frequently small sizes of adjacency lists, resulting in under-utilization of on-chip memory resources. CAM is one of the effective solutions to address the above challenges by enabling highly parallel set intersection operations directly in memory. Consider two adjacency lists with length n and m ($n \leq m$) that require a set intersection. The most commonly used method is the merge-based method, which processes one comparison per cycle and updates the list indices based on the comparison results, achieving a time complexity of $O(m+n)$. When CAM is applied to this operation, it significantly reduces processing complexity by naturally enabling parallel comparisons. For instance, if we store the longer list in CAM, the total set intersection complexity becomes $O(n)$.

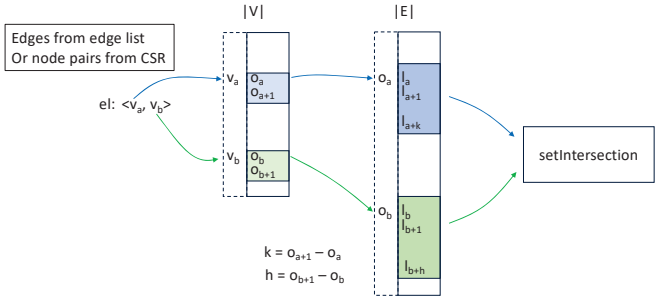


Fig. 5: Simplified process of triangle counting on graphs.

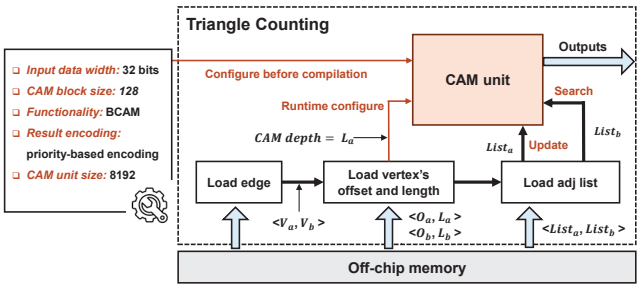


Fig. 6: System architecture of CAM-based triangle counting.

B. Integration of configurable CAM in triangle counting

We build a triangle counting accelerator mainly based on our proposed CAM design, as shown in Figure 6. Specifically, the storage data width is set to 32 bits, the CAM cell is configured to be binary, the block size is set to 128, the system input bus width is 512, and the Encoder in the CAM block is configured to a priority-based encoding scheme. There are additional modules as user kernels to perform the other related functionality for triangle counting, including Load edge, Load vertex's offset and length, and Load adjacent lists modules.

All these graph data are stored in the off-chip memory. The input graph is represented in the CSR format, where each vertex is associated with an offset and length pointing to its neighbors in a column list. The user kernels in the accelerator processes each edge (V_a, V_b) in the graph by first loading the offset index and the length of the lists associated with each vertex, then loading the adjacency lists $<List_a, List_b>$. Before performing the set intersection, the longer list $(List_a)$ is determined and loaded into the generated CAM unit, and the vertices from the shorter one $(List_b)$ are used as the search keys. The number of groups is decided by the length of the longer list. The retrieving process performs as follows: a search operation is performed for each vertex in the shorter list. If there is a match, the number of the triangle is increased by one.

C. Performance evaluation

Since our CAM design provides the grouping mechanism, the adjacent list is duplicated in all groups, enabling a parallel search of the M nodes from the other list. To note here, a list with a length less than 128 occupies the entire CAM block for easy implementation. We conduct a performance comparison against the official implementation of the triangle counting accelerator provided in the AMD Vitis library [1]. This implementation, which utilizes an optimized merge-based set intersection approach, serves as the baseline for our evaluation. To ensure a fair and consistent comparison, we replicate the baseline [1] implementation on the same platform, AMD Alveo U250, under identical operating conditions. Our CAM-based triangle counting accelerator is configured to match the constraints of the baseline design, which is limited to a single DDR channel. Accordingly, our CAM unit is set with 2K entries to remain within a single super logic region (SLR) since the baseline design is also implemented inside a single SLR.

TABLE IX: Execution time (ms) of Traditional TC and CAM-Based Approaches

Dataset	Triangles	Ours	Baseline [1]	Speedup
facebook_combined	1,612,010	5.054	18.7	**3.70x**
amazon0302	717,719	23.086	89.5	**3.88x**
amazon0601	3,986,507	71.210	230.3	**3.23x**
as20000102	6,584	0.422	7.4	**17.54x**
cit-Patents	7,515,023	415.808	800.0	**1.92x**
ca-cit-HepPh	195,758,685	1,526.05	5,361.1	**3.51x**
roadNet-CA	120,676	62.058	108.8	**1.75x**
roadNet-PA	67,150	34.559	88.7	**2.57x**
roadNet-TX	82,869	42.323	96.8	**2.29x**
soc-Slashdot0811	551,724	29.402	259.7	**8.83x**

D. Result Analysis

The performance evaluation of our triangle counting accelerator across various graph datasets [11] is summarized in Table IX. The baseline design [1], representing the current state-of-the-art FPGA-based approach for triangle counting, utilizes a fine-grained pipeline optimized to minimize pipeline bubbles. Nevertheless, its overall performance remains limited due to the merge-based set intersection method, which inherently enforces sequential processing. In contrast, our CAM-based accelerator significantly reduces this sequential bottleneck by enabling parallel execution of set intersection operations, achieving an average speedup of $4.92\times$ compared to the baseline. These results clearly demonstrate the efficacy of our parallel CAM-based approach in overcoming the limitations imposed by sequential set intersection methods.

In summary, the parameterized design of our proposed CAM architecture enables easy integration of it into the triangle counting accelerator. The performance results of the triangle counting accelerator based on our CAM design demonstrate significant improvement over the existing optimized baseline design under the same memory and platform condition.

VI. Conclusion

This paper introduces a novel CAM architecture leveraging DSP slices in FPGAs to address the limitations of traditional LUT and BRAM-based designs when facing data-intensive applications. By repurposing DSP blocks, the proposed architecture achieves high scalability, low update and search latency, efficient resource utilization, and easy system integration. The design supports configurable granularity at cell, block, and unit levels, enabling adaptability to diverse data-intensive applications. Experimental results demonstrate the superior performance and scalability of the architecture, with a triangle counting case study showcasing its practical effectiveness. This work establishes a foundation for integrating DSP-based CAMs into high-performance FPGA systems, opening new avenues for optimizing data-intensive tasks. Our implementation is open-sourced at: https://github.com/Xtra-Computing/DSP_CAM/.

Acknowledgment

This work is supported in part by the Ministry of Education AcRF Tier 2 grant, Singapore (MOE-T2EP20121-0016), and a Google South & Southeast Asia Research Award. This work is also supported in part by AMD under the Heterogeneous Accelerated Compute Clusters (HACC) program.

REFERENCES

[1] "Triangle count - vitis graph library documentation," 2022. [Online]. Available: https://github.com/Xilinx/Vitis_Libraries/tree/main/graph/L2/benchmarks/triangle_count.

[2] A. M. S. Abdelhadi and G. G. F. Lemieux, "Deep and narrow binary content-addressable memories using fpga-based brams," in *2014 International Conference on Field-Programmable Technology (FPT)*, 2014, pp. 318–321. [Online]. Available: https://ieeexplore.ieee.org/document/7082808

[3] A. M. Abdelhadi and G. G. Lemieux, "Modular sram-based binary content-addressable memories," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 207–214. [Online]. Available: https://ieeexplore.ieee.org/document/7160073

[4] A. Ahmed, K. Park, and S. Baeg, "Resource-efficient sram-based ternary content addressable memory," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 4, pp. 1583–1587, 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7797247

[5] T. Fu, C. Wei, Z. Zhu, S. Yang, Z. Yu, G. Dai, H. Yang, and Y. Wang, "Clap: Locality aware and parallel triangle counting with content addressable memory," in *2023 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2023, pp. 1–6.

[6] X. Inc., *UltraScale Architecture DSP Slice User Guide*, 2019, document UG579 (v1.9.1), Last updated: October 2, 2019. [Online]. Available: https://0x04.net/~mwk/xidocs/ug/ug579-ultrascale-dsp.pdf

[7] M. Irfan and Z. Ullah, "G-aetcam: Gate-based area-efficient ternary content-addressable memory on fpga," *IEEE Access*, vol. 5, pp. 20 785–20 790, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8049445

[8] M. Irfan, Z. Ullah, and R. C. Cheung, "D-tcam: A high-performance distributed ram based tcam architecture on fpgas," *IEEE Access*, vol. 7, pp. 96 060–96 069, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8755972

[9] M. Irfan, H. E. Yantır, Z. Ullah, and R. C. Cheung, "Comp-tcam: An adaptable composite ternary content-addressable memory on fpgas," *IEEE Embedded Systems Letters*, vol. 14, no. 2, pp. 63–66, 2021. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9613821

[10] W. Jiang, "Scalable ternary content addressable memory implementation using fpgas," in *Architectures for Networking and Communications Systems*, 2013, pp. 71–82. [Online]. Available: https://ieeexplore.ieee.org/document/6665177

[11] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection," 2014. [Online]. Available: http://snap.stanford.edu/data

[12] Z. U. Muhammad Irfan and R. C. C. Cheung, "Zi-cam: A power and resource efficient binary content-addressable memory on fpgas," *Electronics*, vol. 8, no. 5, p. 584, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/5/584

[13] X.-T. Nguyen, T.-T. Hoang, H.-T. Nguyen, K. Inoue, and C.-K. Pham, "An efficient i/o architecture for ram-based content-addressable memory on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 3, pp. 472–476, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8395019

[14] T. B. Preußer, M. Chiosa, A. Weiss, and G. Alonso, "Using dsp slices as content-addressable update queues," in *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020, pp. 121–126. [Online]. Available: https://ieeexplore.ieee.org/document/9221593

[15] A. Ullah, A. Zahir, N. A. Khan, W. Ahmad, A. Ramos, and P. Reviriego, "Bpr-tcam—block and partial reconfiguration based tcam on xilinx fpgas," *Electronics*, vol. 9, no. 2, p. 353, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/2/353

[16] I. Ullah, Z. Ullah, U. Afzaal, and J.-A. Lee, "Dure: An energy- and resource-efficient tcam architecture for fpgas with dynamic updates," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 6, pp. 1298–1307, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8674772

[17] I. Ullah, Z. Ullah, and J.-A. Lee, "Efficient tcam design based on multipumping-enabled multiported sram on fpga," *IEEE Access*, vol. 6, pp. 19 940–19 947, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8329957

[18] Z. Ullah, "Lh-cam: Logic-based higher performance binary cam architecture on fpga," *IEEE Embedded Systems Letters*, vol. 9, no. 2, pp. 29–32, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/7842534

[19] Z. Ullah, K. Ilgon, and S. Baeg, "Hybrid partitioned sram-based ternary content addressable memory," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 12, pp. 2969–2979, 2012. [Online]. Available: https://ieeexplore.ieee.org/document/6270666/citations#citations

[20] A. Zahir, S. K. Khattak, A. Ullah, P. Reviriego, F. B. Muslim, and W. Ahmad, "Fractcam: fracturable lutram-based tcam emulation on xilinx fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2726–2730, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9217507

Swift or Exact? Boosting Efficient Microarchitecture DSE via Multi-fidelity Partial Order Prediction

Hang Liu[1], Hao Geng[2], Zhuolun He[3], Qi Sun[1,*], Cheng Zhuo[1,*]
[1]Zhejiang University, Hangzhou, China; [2]ShanghaiTech University, Shanghai, China;
[3]The Chinese University of Hong Kong, Hong Kong, China
[*]Corresponding authors, email: {qisunchn, czhuo}@zju.edu.cn

***Abstract*—A significant challenge in microarchitecture design space exploration (DSE) lies in the time-intensive synthesis and simulation process, making rapid design exploration infeasible. While the simulation tools offer reports on performance, power, and area (PPA) in the different stages, the PPA reports at early stages may fail to reflect the true relative qualities for various designs, *i.e.*, with low fidelities. To address these limitations, we propose a novel multi-fidelity optimization algorithm tailored for multi-stage optimization problems. The proposed method employs a non-linear Gaussian process model to effectively fuse data from different stages with different fidelities, minimizing the need for expensive high-fidelity data while maximizing accuracy. Furthermore, a logical regression function and a multi-objective partial order relation are introduced to evaluate the reliability of low-fidelity data, mitigating their potential inaccuracies. Experiments demonstrate that our proposed multi-fidelity optimization algorithm can approximate the Pareto front of the direct design space in a shorter time with better performance.**

I. Introduction

The rapid advancement of artificial intelligence algorithms places increasingly stringent demands on hardware design [1]. Developing a high-quality microarchitecture typically requires specialized expertise and extensive engineering efforts, often resulting in prolonged development cycles. Recently, academia and industry have proposed and adopted systematic, efficient, and universal development processes to mitigate design overhead and expedite design cycles [2]–[4]. For instance, platforms like Chipyard [5] and Gem5 [6] use configurable modules and design parameters to transform intricate microarchitecture design optimization challenges into essential design space exploration problems.

However, the intricate nature of the very large-scale integration (VLSI) flow continues to hinder efficient microarchitecture design. The process encompasses architecture design, register-transfer level (RTL) design, netlist design stages, *etc.*, resulting in a time-consuming workflow. While early-stage performance metrics enable cross-layer optimization [7], [8], the low accuracy of these reports poses a significant risk of missing optimal solutions, as they do not correspond to the final, more accurate results.

Instead of focusing on accurately predicting the values of PPA [9], [10], our primary concern is whether the predictions at different stages can reliably reflect the true relative performance ranking among designs, which we define as fidelity. In the microarchitecture development flow, data from different stages with varying fidelity levels are available. Therefore, we introduce multi-fidelity optimization into microarchitecture DSE to further enhance the efficiency of the exploration process, as shown in Fig. 1. The core challenges of multi-fidelity optimization lie in effectively utilizing multi-fidelity data and determining the appropriate fidelity level for the next evaluation.

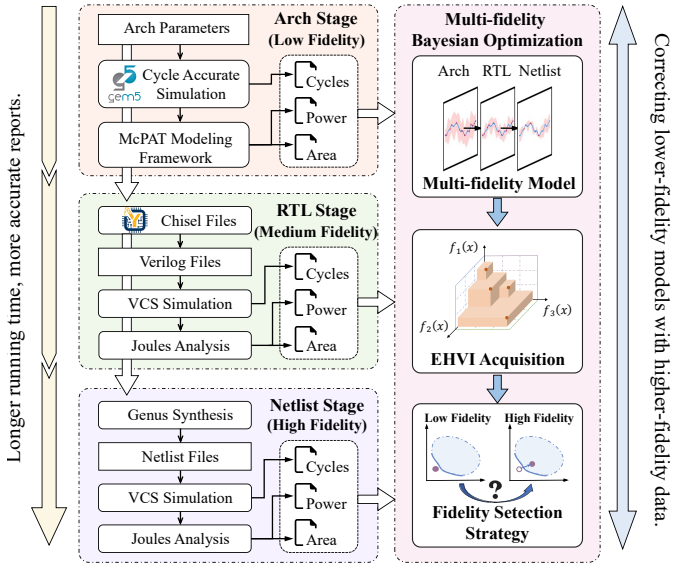


Fig. 1 Flow of agile microarchitecture development with multi-fidelity analytical reporting. Higher-fidelity data are used to correct lower-fidelity models.

Early methods often assumed that the relationships between different fidelity levels were independent or linear [11], [12]. However, in microarchitecture design, reports across different fidelity levels often exhibit non-linear relationships. To address this challenge, Zhang *et al.* [13] use Gaussian models to capture non-linear mappings between low-fidelity and high-fidelity data and optimize on the fused model. Sun *et al.* [14] extend this to multi-objective problems with a correlated multi-objective non-linear optimization algorithm for exploring Pareto solutions using multi-fidelity data. Daulton *et al.* [15] incorporate fidelity level as a new parameter feature, using a knowledge gradient-based general one-step lookahead function for simultaneous optimization. Irshad *et al.* [16] enhance this with a multi-objective policy that includes trust gain per evaluation cost, enabling efficient multi-objective, multi-fidelity optimization.

Unfortunately, these methods fail to leverage the interpretability of data from different fidelities. If low-fidelity data accurately ranks candidate solutions, additional high-fidelity evaluations become unnecessary. High-fidelity evaluations should only be conducted when low-fidelity data cannot reliably rank exploration points. Additionally, defining an effective partial order for different points is essential, especially when ranking complex multi-objective problems across fidelity levels.

In this work, to efficiently utilize multi-fidelity data, we propose an efficient multi-fidelity Bayesian optimization framework based on a non-linear multi-fidelity model and partial order prediction, as shown in the Fig. 1. Our contributions are summarized as follows:

- We utilize non-linear Gaussian modeling to integrate data of varying fidelities and employ the expected hypervolume improvement (EHVI) acquisition function to balance exploitation and exploration.
- We employ logistic regression to predict the potential inversion of PPA values between target points and current Pareto points across different fidelity levels.
- A decision-making framework has been proposed to evaluate the partial order relationship in multi-objective problems, facilitating the efficient discovery of new Pareto points.
- We conduct comprehensive experiments on parameterized RISC-V CPUs, benchmarked on a widely recognized RISC-V benchmark suite. Results demonstrate that our method increases hypervolume by 12.9%, enhances Pareto front quality by 57.7%, and achieves a 48.0% faster convergence speed compared to the previous method.

II. Preliminaries

A. RISC-V CPU

The RISC-V architecture, a reduced instruction set computer architecture, has gained widespread adoption in recent years due to its efficient hardware design and implementation, coupled with its remarkable power efficiency [17]. Frameworks such as Chipyard [5] further accelerate the development of RISC-V CPUs by providing an open-source platform for integrating diverse components. Chipyard integrates various RISC-V CPUs, including the high-performance Berkeley Out-of-Order Machine (BOOM) [18] and the streamlined, efficient Rocket [19] in-order CPU.

B. Multi-objective Optimization

Bayesian optimization [20] is a powerful global optimization method for black-box functions with high evaluation costs, such as microarchitecture design. It approximates the objective function with a surrogate model, often a Gaussian process, and uses an acquisition function to balance exploration and exploitation.

Optimizing microarchitecture designs often requires balancing multiple, conflicting objectives. Multi-objective Bayesian optimization (MOBO) extends standard Bayesian optimization, allowing efficient exploration of multi-objective design space. Let functions $\{f_i(\boldsymbol{x})\}_{i=1}^{m}$, *i.e.*, f_1 = power, f_2 = area, f_3 = cycles, denote the m-dimension metrics to be minimized and $\mathbb{X}$ denotes the parameter space. A parameter vector $\boldsymbol{x}^* \in \mathbb{X}$ is said to dominate $\boldsymbol{x}' \in \mathbb{X}$, denoted as $\boldsymbol{x}^* \succeq \boldsymbol{x}'$, if the following two conditions are met in this minimization problem:

1) $f_i(\boldsymbol{x}') \geq f_i(\boldsymbol{x}^*), \forall i \in \{1, ..., m\}$,
2) There exists at least one j such that $f_j(\boldsymbol{x}') > f_j(\boldsymbol{x}^*)$.

The set of parameter vectors that are not dominated by other vectors is called the Pareto-optimal set, denoted as $\mathbb{X}_{Pareto}$, which is the target of MOBO:

$$\mathbb{X}_{Pareto} = \{\boldsymbol{x}^* \in \mathbb{X} | \nexists \boldsymbol{x}' \in \mathbb{X}, \boldsymbol{x}' \succeq \boldsymbol{x}^*\}. \tag{1}$$

For the problem with multiple optimization objectives, Bayesian optimization constructs surrogate models for these unknown black-box objectives. A multi-objective acquisition function is then constructed based on these surrogate models to evaluate the quality of a parameter configuration $\boldsymbol{x}$ for identifying the Pareto set, eliminating the need for extensive and time-consuming EDA flow evaluations.

C. Multi-fidelity Optimization

Fidelity refers to the degree to which a model accurately represents the state of a real-world project or application. It essentially quantifies the realism of the model. Low-fidelity models are characterized by lower precision, while high-fidelity models exhibit higher precision.

For a multi-fidelity problem, each fidelity i has a regression model $f_i(\boldsymbol{x})$. The multi-fidelity model can be defined as:

$$f_{i+1}(\boldsymbol{x}) = z(f_i(\boldsymbol{x}), \boldsymbol{x}) \tag{2}$$

where $z(\cdot)$ is a machine learning model or function, f_i is the model of the lower fidelity and f_{i+1} is the higher one. Higher fidelity levels offer greater accuracy but come at the cost of significantly longer running times compared to lower fidelity levels. If low-fidelity results are satisfactory, we can skip higher-fidelity tests, to save time. Therefore, we need to measure the quality of reports at each fidelity level to determine if subsequent design stages are necessary. In the rest of this paper, the three fidelities in microarchitecture design shown in Fig. 1 are shorted as $arch$, rtl, and $netlist$.

D. Problem Formulation

Definition 1 (Hypervolume)**.** In multi-objective problems, solutions are defined in terms of Pareto fronts. The quality of such fronts can be measured by hypervolume [21]:

$$\mathrm{HV}\left(f^{\mathrm{ref}}, \mathbb{X}\right) = \Lambda\left(\bigcup_{\mathbb{X}_n \in \mathbb{X}} \left[f_1(\mathbb{X}_n), f_1^{\mathrm{ref}}\right] \times \cdots \times \left[f_m(\mathbb{X}_n), f_m^{\mathrm{ref}}\right]\right), \tag{3}$$

where f^{ref} denotes a chosen reference point, typically set to represent a very poor performance value. f_m means the m-th metric, *i.e.*, cycles, power, and area. Λ refers to the Lebesgue measure.

Definition 2 (ADRS)**.** Another metric to measure the quality of Pareto fronts is the average distance to reference set (ADRS) [22]:

$$\mathrm{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|} \sum_{\gamma \in \Gamma} \min_{\omega \in \Omega} D(\gamma, \omega), \tag{4}$$

where D is the Euclidean distance function. Γ is the real Pareto-optimal set and Ω is the learned Pareto-optimal set. ADRS is often exploited to measure how close a learned Pareto-optimal set is to the real Pareto-optimal set of the design space.

Problem 1 (Multi-fidelity microarchitecture DSE)**.** Given a search space $\mathbb{X}$, the metric space including three different fidelity levels: $\{\mathbb{Y}_{arch}, \mathbb{Y}_{rtl}, \mathbb{Y}_{netlist}\}$, we define the multi-fidelity microarchitecture DSE as finding the subset $\mathbb{X}^* \in \mathbb{X}$ with corresponding metrics $\mathbb{Y}^*_{netlist}$ forming the Pareto optimal set, by utilizing data from different fidelity levels.

III. Multi-fidelity Bayesian Optimization

In this section, we present a multi-fidelity Bayesian optimization method designed to efficiently explore the microarchitecture design space by utilizing data from various fidelity levels with partial order prediction.

A. Overall Optimization Flow

Fig. 2 shows an overview of the proposed multi-fidelity Bayesian optimization. Firstly, we determine the design space to be explored based on the architecture constraints and engineers' expertise. Since the three fidelity stages are performed sequentially, the points run at a higher-fidelity stage are subsets of the points run at a lower-fidelity stage, *i.e*, $\mathrm{X}_{netlist} \subseteq \mathrm{X}_{rtl} \subseteq \mathrm{X}_{arch}$. These points are then fed into the VLSI flow to get corresponding fidelity reports. To fully leverage data from different fidelity stages, we construct and initialize a non-linear multi-fidelity Gaussian Process (GP) model. The expected hypervolume improvement acquisition function guides the selection of the next sampling point x^*, balancing the trade-off between exploration and exploitation. In each optimization time step, we employ a sequential fidelity strategy. After obtaining a fidelity report for x^* from the VLSI flow, the report is evaluated to determine if further evaluation at the next fidelity stage is necessary. The learned Pareto optimal set is acquired when a pre-determined time budget is met.

B. Non-linear Multi-fidelity Model

Generally, in multi-fidelity optimization, the relationships between multiple fidelities are assumed to be linear, *e.g.*, [12]. However, this assumption does not hold for the three stages of the VLSI flow, as they exhibit strong non-linear correlations. Therefore, we propose a non-linear multi-fidelity model [23] to further exploit the corresponding non-linear correlations between low-fidelity and high-fidelity data. The first level of the multi-fidelity model is based on the traditional GP model. Then the non-linear model can be formulated as Equation (5).

$$f_h(x) = z(f_l(x)) + \delta(x), \tag{5}$$

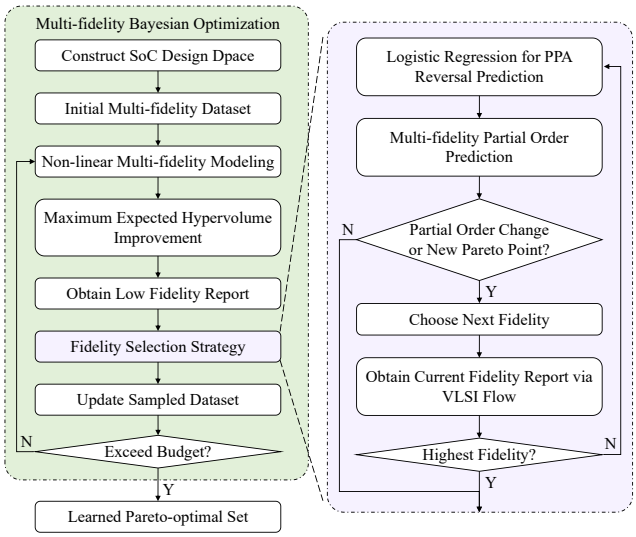


Fig. 2 Overview of the proposed multi-fidelity Bayesian optimization.

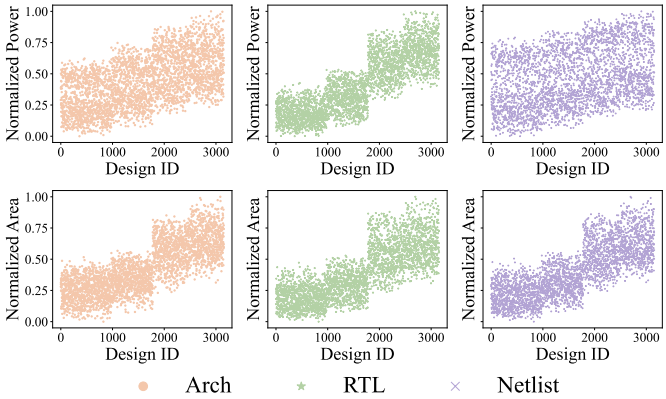


Fig. 3 Normalized power and area values of the BOOM design across three fidelity stages.

where $z(\cdot)$ represents the non-linear correlation between the low-fidelity and high-fidelity data, while $\delta(\cdot)$ is a Gaussian process model for the high-fidelity data, independent of the low-fidelity model. By concatenating the outputs of the low-fidelity model with the features of the architectural parameters as input features for the high-fidelity GP model, we can effectively fuse low-fidelity and high-fidelity information by constructing $GP(f_l(x), x)$.

The power and area values obtained at different fidelity stages are shown in Fig. 3 as examples to illustrate the complex non-linear relationships between the three fidelities. It can be seen that the reports from the three fidelities exhibit high divergence, making it difficult to regress their relationships using traditional linear models. Therefore, employing a non-linear model is a sensible choice for handling multi-fidelity exploration of microarchitecture design space.

C. Multi-fidelity Acquisition Function

After constructing a non-linear multi-fidelity model that integrates information from all fidelities, we introduce Expected

Hypervolume Improvement (EHVI) [24] as the acquisition function for efficient learning and measurement of the Pareto set in multi-objective optimization problems.

Assuming at the current optimization step $t+1$, we have a set of parameter configurations $\mathbb{X}_h = \{\boldsymbol{x}_h\}^t$ and their corresponding values $\mathbb{Y}_h = \{\boldsymbol{y}_h\}^t$ at the highest fidelity level. A virtual configuration point f^{ref} is a reference point, set to extremely large values for all design objectives in a multi-objective minimization problem. At each iteration, we aim to sample a parameter configuration that leads to the highest expected hypervolume improvement of the sampling configuration set. To achieve this, we will traverse the entire unsampled configuration space and estimate the expected hypervolume improvement for each configuration at the highest fidelity. The expected hypervolume improvement is defined as Equation (6).

$$\begin{aligned}&\text{EHVI}(\boldsymbol{x}_{t+1}|\mathbb{X},\mathbb{Y}) = \\ &\mathbb{E}_{p(\boldsymbol{y}(\boldsymbol{x}_{t+1})|\mathbb{X},\mathbb{Y})}\left[\text{HV}(f^{\text{ref}},\mathbb{Y}\cup\boldsymbol{y}(\boldsymbol{x}_{t+1})) - \text{HV}(f^{\text{ref}},\mathbb{Y})\right].\end{aligned} \tag{6}$$

By finding a parameter configuration that maximizes the EHVI, we can effectively balance exploration and exploitation in complex multi-objective optimization problems, facilitating the discovery of an improved Pareto set.

D. Fidelity Selection Strategy

In multi-fidelity optimization, in addition to employing an acquisition function to select the next sampling point, it is also crucial to determine the evaluation fidelity for the multi-fidelity Bayesian optimization process. Due to the significantly higher computational cost of high-fidelity simulations compared to low-fidelity ones, we should only sample high-fidelity data when necessary. The search process remains unaffected by varying fidelity data as long as all comparisons are accurately ranked. Given two parameter configurations with a significant difference in their low-fidelity metrics, the ranking based on high-fidelity data is unlikely to deviate from the ranking based on low-fidelity data. Therefore, we can skip the time-consuming high-fidelity simulation process. High-fidelity data is only necessary for more accurate assessments when low-fidelity data cannot confidently rank the target points. In multi-objective optimization, we aim to sample a new Pareto point in each optimization step, meaning the point that is not dominated by current Pareto points.

Therefore, we propose an adaptive fidelity selection strategy based on partial order prediction. This strategy utilizes a logistic regression function to predict the potential reversal of the PPA between the target point and the current Pareto set across different fidelity levels. In multi-objective optimization, we define the dominance of a newly sampled point by the current Pareto set as a partial order relationship. By combining the prediction results of the logistic regression function, we can predict the change in the partial order relationship between two points across low- and high-fidelity levels. This prediction informs the decision of whether or not to employ high-fidelity simulations.

Algorithm 1 Fidelity Selection Strategy

Input: The next sampling point x^*, current pareto metrics set $\mathbb{Y}_p$, Logistic regression function LR_i^m, where $i \in \{arch, rtl\}, m \in \{power, area, cycles\}$.
Output: The set of simulation results for x^* at different fidelity levels $\mathbb{Y}^*$.

1: Initialize $i \leftarrow arch$, $\boldsymbol{y}_i^* \leftarrow \text{VLSI}(x^*, i)$, and $\mathbb{Y}^* \leftarrow \boldsymbol{y}_i^*$;
2: **for all** $\boldsymbol{y}_p \in \mathbb{Y}_p$ **do**
3: Assess the partial ordering of the two points at the current fidelity: $PO_i \leftarrow (\boldsymbol{y}_p, \boldsymbol{y}_i^*)$;
4: Predict potential PPA reversals between current and highest fidelity using a logistic regression model: $REV_i = LR_i^m(\boldsymbol{y}_p, \boldsymbol{y}_i^*)$;
5: Predict the partial ordering of the two points at the highest fidelity: $PO_h \leftarrow REV_i$;
6: **if** PO_h is not equal to PO_i **then**
7: $i = i+1$, $\boldsymbol{y}_i^* \leftarrow \text{VLSI}(x^*, i)$;
8: $\mathbb{Y}^* \leftarrow \mathbb{Y}^* \cup \boldsymbol{y}_i^*$, go back to line 2;
9: **end if**
10: **end for**
11: **if** $\boldsymbol{y}_i^*$ is a new Pareto metrics at fidelity i **then**
12: $i = i+1$, $\boldsymbol{y}_i^* \leftarrow \text{VLSI}(x^*, i)$;
13: $\mathbb{Y}^* \leftarrow \mathbb{Y}^* \cup \boldsymbol{y}_i^*$, go back to line 2;
14: **end if**
15: **return** $\mathbb{Y}^*$.

Algorithm 1 presents a detailed description of the fidelity selection strategy applied at each step of the optimization process. In line 1, we start by using the lowest fidelity level to obtain data for the newly sampled point within the VLSI process. Next, we sequentially compare the new point with all current Pareto points to evaluate the partial order at the current fidelity (line 3).

To infer the partial order relationship between the two points at high fidelity, we employ a logistic regression function to predict the potential reversal of their PPA values between low and high fidelity levels (line 4). After training on the previously sampled dataset, the relationship between the difference in the PPA values of the two points at $i-th$ fidelity $\Delta = |y_i^m - y_p^m|$ and the probability of a wrong ranking for the highest fidelity level can be described by a logistic regression model as Equation (7):

$$\text{LR}_i^m(y_i^m, y_p^m) = \frac{1}{1+e^{-(\beta_0+\beta_1\Delta)}}, \tag{7}$$

where the hyperparameters β_0 and β_1 are trained on the currently sampled high-fidelity data and its corresponding low-fidelity data. Specifically, we use a balanced class weighting approach to address the issue of class imbalance in the sampled data.

Then, based on the logistic regression prediction, we will determine whether the partial order relationship between the new sample point and the Pareto point changes (line 5).

TABLE I Configurable Parameters in the BOOM.

Module	Parameters	Candidates
FrontEnd	FetchWidth	4, 8
	FetchBufferEntry	8, 16, 24, 32
	RasEntry	16, 32, 36, 48
IDU	DecodeWidth	1, 2, 3, 4
	RobEntry	32, 64, 96, 128
	IntPhyRegister	52, 80, 100, 128
	FpPhyRegister	48, 64, 96, 128
ISU	Mem/Int/FpIssueWidth	1, 2, 3, 4
	Mem/Int/FpIssueEntry	8, 12, 16, 20, 24, 32, 40
LSU	LDQEntry	8, 16, 24
	STQEntry	8, 16, 24
ICache	ICacheWay	4, 8
	ICacheSet	32, 64
DCache	DCacheWay	4, 8
	DCacheSet	64, 128
	DCacheMSHR	2, 4, 8

If a change in the partial order relationship with a current Pareto point is detected (line 6), or if the new sample point is predicted to be a new Pareto point at high fidelity (line 11), we will proceed to the next fidelity level and obtain a more accurate evaluation report through the corresponding VLSI simulation process. Finally, through partial order prediction between the newly sampled point and the current Pareto points, we can effectively select the appropriate fidelity level and acquire multi-fidelity data.

IV. Experiment

A. Experimental Setup and Benchmarks

To evaluate the effectiveness of our proposed method, we conducted tests on two RISC-V CPU designs: BOOM with 20 design parameters and Rocket with 8 design parameters. Early architectural-level simulations are conducted using Gem5 [6], a cycle-accurate simulator, and McPAT [25], an integrated power and area modeling framework for architectures. Chipyard is used to generate RTL code from parameterized Chisel files. Subsequent PPA values for RTL and netlists are obtained from Synopsys VCS [26], Cadence Genus [27], and Cadence Joules [28] with the ASAP7 7nm PDK [29]. We use official RISC-V benckmarks [30] as workloads in the DSE. Multi-fidelity data encompasses simulation data generated at the architectural, RTL, and netlist levels. The multiple optimization objectives include power, area, and cycles. The cycles here are the average cycles of the benchmarks' inference performance on the CPU.

TABLE I presents the design space of the BOOM microarchitecture explored in our study. Hypervolume (HV) and average distance to reference set (ADRS) are used to compare the effectiveness of each multi-fidelity design space exploration method. Both metrics here are evaluated on the highest fidelity data learned by each method. To provide a comprehensive comparison of our method, we select several previous multi-fidelity algorithms and a classic single-fidelity algorithm:

- FPL'18 [12]: Method based on linear multi-fidelity and independent multi-objective models.

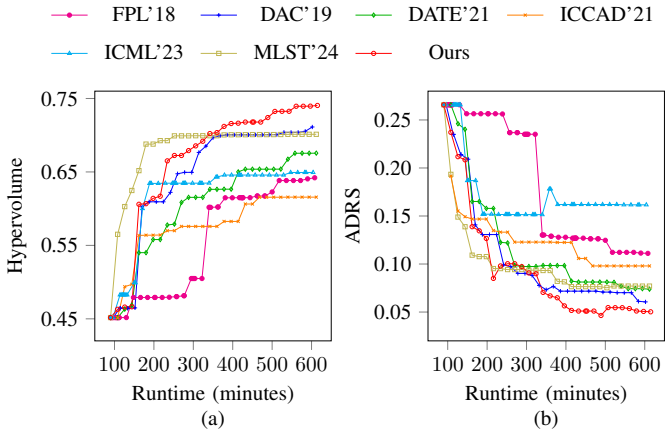


Fig. 4 Comparisons on the BOOM, (a) Hypervolume changes with EDA tool runtime. (b) ADRS changes with runtime.

- DAC'19 [13]: Method based on a nonlinear Gaussian model and a sequential fidelity strategy.
- DATE'21 [14]: Method with a correlated multiobjective non-linear optimization algorithm.
- ICCAD'21 [31]: Single-fidelity method with active learning-based initial dataset sampling and Gaussian process with deep kernel learning.
- ICML'23 [15]: Method utilizing one-step lookahead acquisition function based on the Knowledge Gradient.
- MLST'24 [16]: Method with trust-based multi-objective optimization.

Since verifying every sampled microarchitecture design online is time-consuming, we construct two offline datasets. Each design is simulated at the architecture, RTL, and netlist levels. The BOOM dataset contains 3153 parameter configurations, while the Rocket dataset contains 640 configurations. These two datasets consume 4,423 CPU hours to run the VLSI flow and obtain PPA values for each configuration at different fidelity levels. Our method and the baselines share the same randomly initialized parameter configurations. All experiments are conducted on a Linux server with Intel Xeon CPU (Platinum 8475B) and 512 GB RAM.

B. Comparisons against Previous Works

Considering the varying time costs of simulations at different fidelity levels, we initialize the models for all multi-fidelity methods using 30 architectural-level data, 15 RTL-level data, and 2 netlist-level data. Specifically, for the single-fidelity method [31], we initialize it with 6 high-fidelity data. Additionally, we replace the expected improvement acquisition function used in DAC'19 [13] with the EHVI acquisition function employed in this work to adapt it to our proposed multi-objective optimization problem. To compare the performance of different methods, we set the EDA tool runtime budget to 600 minutes.

TABLE II shows the comparison of our method with previous methods on two different RISC-V CPUs. The results show that our proposed multi-objective optimization method

TABLE II Comparisons against previous works.

Metric	Method	FPL'18 [12]	DAC'19 [13]	DATE'21 [14]	ICCAD'21 [31]	ICML'23 [15]	MLST'24 [16]	**Ours**
BOOM	HV	0.6420 (100%)	0.7112 (↑10.8%)	0.6756 (↑ 5.2%)	0.6157 (↓4.1%)	0.6493 (↑ 1.1%)	0.7013 (↑ 9.2%)	**0.7405 (↑15.3%)**
	ADRS	0.1111 (100%)	0.0606 (↓45.5%)	0.0738 (↓33.6%)	0.0980 (↓11.8%)	0.1616 (↑45.4%)	0.0770 (↓30.7%)	**0.0502 (↓54.8%)**
Rocket	HV	0.6799 (100%)	0.7214 (↑ 6.1%)	0.7367 (↑ 8.4%)	0.6675 (↓ 1.8%)	0.6862 (↑ 0.9%)	0.7337 (↑ 7.9%)	**0.7513 (↑10.5%)**
	ADRS	0.2185 (100%)	0.0963 (↓55.9%)	0.1408 (↓35.6%)	0.1745 (↓20.2%)	0.1011 (↓53.7%)	0.1547 (↓29.2%)	**0.0894 (↓59.1%)**
Average	HV	0.6610 (100%)	0.7163 (↑ 8.4%)	0.7062 (↑ 6.8%)	0.6416 (↓ 2.9%)	0.6678 (↑ 1.0%)	0.7175 (↑ 8.5%)	**0.7459 (↑12.9%)**
	ADRS	0.1648 (100%)	0.0785 (↓52.4%)	0.1073 (↓34.9%)	0.1363 (↓17.3%)	0.1313 (↓20.3%)	0.1159 (↓29.7%)	**0.0698 (↓57.7%)**

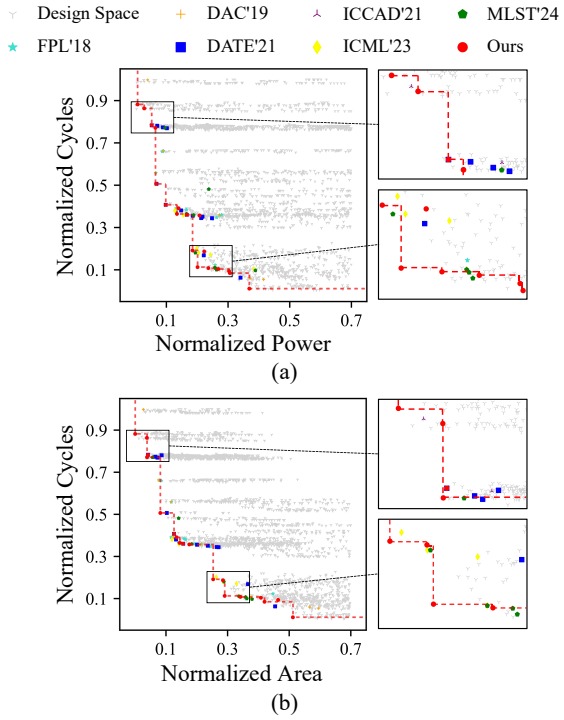


Fig. 5 Learned Pareto-optimal set of BOOM in cycles-power & cycles-area metric space. The red dashed line represents the Pareto frontier learned by our proposed method.

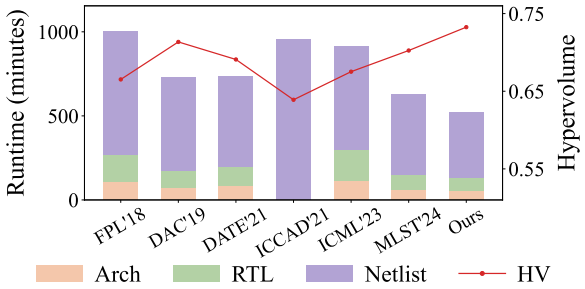


Fig. 6 Comparison of runtime distribution and hypervolume on the BOOM across different multi-fidelity optimization methods.

can better explore the search space under the same runtime allocation. Our method achieves the highest hypervolume and the lowest ADRS in the normalized space of the three metrics (cycles, power, and area). On average of hypervolume metrics, our method is 12.9% higher than FPL'18 [12], 4.5% higher than DAC'19 [13], 6.1% higher than DATE'21 [14], 10.0% higher than ICCAD'21 [31], 11.9% higher than ICML'23 [15], 4.4% higher than MLST'24 [16].

Fig. 4 shows the increase of hypervolume and the decrease of ADRS of each method on the BOOM dataset. The red curve rises faster than others in hypervolume. It falls faster than others in ADRS, which means that our method enables a more efficient and comprehensive exploration of the entire design space compared to other approaches. By leveraging low-fidelity data and partial order prediction, our method avoids spending excessive time on unexplored regions with suboptimal performance. Fig. 5 visually illustrates the distribution of the Pareto frontiers learned by various methods in the cycles-power and cycles-area spaces. It is evident that our method not only more effectively explores the Pareto frontier but also achieves a more comprehensive exploration of the entire design space.

Fig. 6 shows the distribution of simulation time across different fidelity levels consumed by our method and others during the exploration of the BOOM design space, along with the final hypervolume values achieved. As shown in the figure, compared to single-fidelity [31] and linear multi-fidelity methods [12], the non-linear multi-fidelity approach significantly enhances the efficiency of design space exploration. The figure also shows that on discrete fidelity data, our method, by constructing a corresponding GP model for each fidelity level, learns multi-fidelity data more effectively than using a single GP model to handle all fidelity levels [15], [16]. Guided by partial order prediction, our method significantly reduces the overall time cost of design space exploration.

V. Conclusion

In this paper, we have proposed an efficient multi-fidelity Bayesian optimization method for multi-objective optimization problems. We effectively fuse data from different fidelities by employing a non-linear multi-fidelity model to learn the complex correlation between low-fidelity and high-fidelity data. Under the inference of logistic regression function, we determine whether a more time-consuming but accurate evaluation is necessary by assessing whether the partial order relationship between two points changes across different fidelity levels. Experiments in exploring different RISC-V design spaces demonstrate the effectiveness and superior performance of the proposed framework.

Acknowledgment

This work was partially supported by Yangtze River Delta Science and Technology Innovation Community Joint Fundamental Research Project (Grant No. 2024CSJZN0500).

REFERENCES

[1] J. Deng, Z. Shi, and C. Zhuo, "Energy-efficient real-time uav object detection on embedded platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 3123–3127, 2019.

[2] L. Chen, Y. Wu, C. Wen, S. Wang, L. Zhang, B. Yu, Q. Sun, and C. Zhuo, "An agile framework for efficient llm accelerator development and model inference," 2024.

[3] D. Luo, Q. Sun, X. Li, C. Bai, B. Yu, and H. Geng, "Knowing the spec to explore the design via transformed bayesian optimization," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024.

[4] Y. Gao, D. Luo, C. Bai, B. Yu, H. Geng, Q. Sun, and C. Zhuo, "Is vanilla bayesian optimization enough for high-dimensional architecture design optimization?" 2024.

[5] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge *et al.*, "Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs," *IEEE Micro*, 2020.

[6] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[7] C. Zhuo, K. Unda, Y. Shi, and W.-K. Shih, "A novel cross-layer framework for early-stage power delivery and architecture co-exploration," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016.

[8] ——, "From layout to system: Early stage power delivery and architecture co-exploration," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1291–1304, 2018.

[9] D. Luo, Q. Sun, Q. Xu, T. Chen, and H. Geng, "Attention-based eda tool parameter explorer: From hybrid parameters to multi-qor metrics," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024.

[10] L. Feng, W. Liu, C. Guo, K. Tang, C. Zhuo, and Z. Wang, "Gandse: Generative adversarial network-based design space exploration for neural network accelerator design," *ACM Transactions on Design Automation of Electronic Systems*, vol. 28, no. 3, pp. 1–20, 2023.

[11] K. Kandasamy, G. Dasarathy, J. Oliva, J. Schneider, and B. Poczos, "Multi-fidelity gaussian process bandit optimisation," *Journal of Artificial Intelligence Research*, vol. 66, pp. 151–196, 2019.

[12] C. Lo and P. Chow, "Multi-fidelity optimization for high-level synthesis directives," in *2018 28th International conference on field programmable logic and applications (FPL)*, 2018.

[13] S. Zhang, W. Lyu, F. Yang, C. Yan, D. Zhou, X. Zeng, and X. Hu, "An efficient multi-fidelity bayesian optimization approach for analog circuit synthesis," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019.

[14] Q. Sun, T. Chen, S. Liu, J. Miao, J. Chen, H. Yu, and B. Yu, "Correlated multi-objective multi-fidelity optimization for hls directives design," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021.

[15] S. Daulton, M. Balandat, and E. Bakshy, "Hypervolume knowledge gradient: a lookahead approach for multi-objective bayesian optimization with partial information," in *International Conference on Machine Learning*, 2023.

[16] F. Irshad, S. Karsch, and A. Döpp, "Leveraging trust for joint multi-objective and multi-fidelity optimization," *Machine Learning: Science and Technology*, vol. 5, no. 1, p. 015056, 2024.

[17] E. Blem *et al.*, "Power Struggles: Revisiting The RISC vs. CISC Debate on Contemporary ARM and x86 Architectures," in *Proc. HPCA*, 2013.

[18] C. Celio, D. A. Patterson, and K. Asanović, "The Berkeley Out-of-order Machine (boom): An Industry-competitive, Synthesizable, Parameterized RISC-V Processor," *University of California at Berkeley Berkeley United States, Tech. Rep*, 2015.

[19] K. Asanovic *et al.*, "The Rocket Chip Generator," *EECS Department, University of California, Berkeley, Tech. Rep.*, 2016.

[20] J. Mockus, "The Application of Bayesian Methods for Seeking the Extremum," *Towards global optimization*, 1998.

[21] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: Methods and applications*. Shaker Ithaca, 1999, vol. 63.

[22] D. A. Van Veldhuizen, *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. Air Force Institute of Technology, 1999.

[23] P. Perdikaris, M. Raissi, A. Damianou, N. D. Lawrence, and G. E. Karniadakis, "Nonlinear information fusion algorithms for data-efficient multi-fidelity modelling," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 473, no. 2198, p. 20160751, 2017.

[24] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9851–9864, 2020.

[25] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

[26] "Synopsys vcs," https://www.synopsys.com/verification/simulation/vcs.html, 2024.

[27] "Genus synthesis solution," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/synthesis/genus-synthesis-solution.html, 2024.

[28] "Joules rtl power solution," https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html, 2024.

[29] V. Vashishtha, M. Vangala, and L. T. Clark, "Asap7 predictive design kit development and cell design technology co-optimization," in *Proc. ICCAD*, 2017.

[30] "Risc-v tests," https://github.com/riscv-software-src/riscv-tests, 2023.

[31] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework," in *Proc. ICCAD*, 2021.

FT-MUX: A Fault-Tolerant Microfluidic Multiplexer Design

Mengchu Li
TU Munich
mengchu.li@tum.de

Jiahui Peng
TU Munich
jiahui.peng@tum.de

Tsun-Ming Tseng
TU Munich
tsun-ming.tseng@tum.de

Ulf Schlichtmann
TU Munich
ulf.schlichtmann@tum.de

***Abstract*—Continuous-flow microfluidic chips are multilayered miniaturized platforms to manipulate small volumes of fluids with valves. There are two types of channels on a chip: flow channels for the reaction of fluids, and control channels for the actuation of valves. Multiplexers (MUXes) are essential microfluidic components for individually addressing many flow channels with few control channels. As the integration scale of microfluidic chips increases, the reliability of MUXes becomes a critical concern, as a single defective control channel in a MUX will affect a large part of the flow channels addressed by the MUX. This paper formally analyzes and identifies the design rules for a MUX to tolerate n defective control channels, and model the fault-tolerant MUX (FT-MUX) design problem as a binary constant weight code problem to minimize resource overheads. We demonstrate that FT-MUX improves resource efficiency by up to hundreds of times compared to the conventional fault-tolerant design method. Besides, given no less than 10 control channels, FT-MUX tolerates at least one defective control channel and addresses even more flow channels with equal or fewer resources than a standard MUX. The advantages become more significant as the integration scale increases.**

I. Introduction

Microfluidic technology enables the manufacturing of miniaturized devices, also known as microfluidic chips, that manipulate fluids at the nanoliter scale or below, exhibiting advantages in automation, rapid execution, better sensitivity, less waste, etc. In particular, the fabrication of valves using multilayer soft lithography [1] allows for the creation of microfluidic large-scale integration (mLSI) chips [2]. Such a chip consists of a flow layer (integrated) with flow channels for the storage and transportation of fluid samples and reagents, and a control layer (integrated) with control channels for the actuation of valves to manipulate the fluids. Nowadays, it is common to have hundreds to thousands of flow channels on a single small chip for high-throughput parallel applications such as cell-culture [3], cell-analysis [4], drug screening [5], serology test [6], etc. To independently address many flow channels with an affordable number of control inputs, microfluidic multiplexers (MUXes) have become indispensable for large-scale microfluidic applications [7].

A standard microfluidic multiplexer (MUX) [8] can independently address up to n flow channels with $2\lceil\log_2(n)\rceil$ control channels, as shown in Figure 1. To that end, each control channel is responsible for the pressure supply to valves on at least $\lfloor\frac{n}{2}\rfloor$ flow channels. Compared to flow channels, control channels have smaller feature sizes and are thus more prone to defects [9]. If a control channel in a standard MUX is defective, at least half of the flow channels addressed by the MUX will be affected, leading to waste of samples and reagents, contamination of products, or even disposal of the whole chip. As the integration scale of microfluidic chips increases, the reliability of MUXes becomes a critical concern [10].

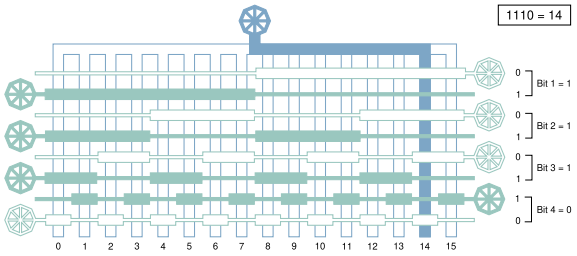


Fig. 1. A standard MUX to address 16 flow channels (colored in blue) with 8 control channels (colored in green). The pressure states in pressurized/depressurized control channels are denoted as 1 and 0, respectively. Valves are widened control channel segments. A pressurized valve blocks the flow channel underneath. Every two control channels form a pair representing one bit of a flow channel index: pressure states $(0, 1)$ and $(1, 0)$ represent bit '1' and '0', respectively. E.g., configuring the control channels to pressure states $(0, 1)$ $(0, 1)$ $(0, 1)$ $(1, 0)$ constructs a unique flow path from the fluid inlet to flow channel 14 (1110). Figure adapted from [2].

Efforts have been made in the past to improve the reliability of MUXes from two aspects:

1) Reduce the likelihood of defects. Q. Wang et al. proposed to optimize the switching order of valves to reduce the switching frequency of a MUX [11], and S. Liang et al. proposed a novel MUX design with certain valves and channels merged together [12]. However, as the number of control channels in the MUX increases, defects are almost inevitable, as reported in [12]. Since the proposed methods do not tolerate faults, once a defect happens, many flow channels will be affected, leading to significant waste.
2) Tolerate defects by introducing backups. W. Huang et al. proposed an approach to introduce redundant components considering a given set of fault scenarios [13], and Y. Zhu et al. proposed an approach to synthesize application-specific MUXes with duplicated valves and backup paths to actuate the valves [14], [15]. However, the performances of both approaches heavily depend on the given applications and are thus, in general, not predictable.

Besides, state-of-the-art methods mostly assume that defects happen at individual valves without influencing other valves

along the same control channel, which is not the case. Specifically, microfluidic chips suffer two major types of defects: blockage and leakage [16]. Blockage means a channel is disconnected such that pressure cannot pass through, and the risk of blockage inversely correlates with the feature size. Since valves are widened segments of control channels, blockage defects are more likely at the thinner parts of a control channel instead of valves. When a control channel is blocked, all valves along the channel cannot be properly pressurized. On the other hand, leakage means two control channels are unintentionally connected such that both channels share the same pressure states. When a control channel leaks to another, pressurizing/depressurizing one channel will also inevitably close/open all valves in the other channel. As conclusion, a defective valve in the MUX implies the malfunctions of all valves along a defective control channel. Thus, a fault-tolerant MUX design should target defective control channels rather than defective valves.

In this paper, we propose a rule for designing a fault-tolerant microfluidic multiplexer (FT-MUX) to tolerate n defective control channels with minimal resource overheads. We further model the FT-MUX design problem as a binary constant weight code problem, and proposes a graph model to solve it as an independent vertex set problem. We demonstrate that FT-MUX improves resource efficiency by up to hundreds of times compared to the conventional fault-tolerant design method. Besides, given no less than 10 control channels, FT-MUX tolerates at least one defective control channel and addresses even more flow channels with equal or fewer resources than a standard MUX. The performance advantages become more significant as the integration scale increases, showing that FT-MUX provides an efficient and reliable solution for fluid multiplexing in large-scale microfluidic applications.

II. Background

A. State-of-the-art MUX designs

A standard MUX consisting of n control channels can independently address up to $2^{\frac{n}{2}}$ flow channels, as shown in Figure 1, and has been widely applied in various state-of-the-art high-throughput microfluidic applications.

Recently, S. Liang et al. proposed a novel MUX design named CoMUX with a combinatorial coding strategy. A CoMUX consisting of n control channels can independently address up to $C^n_{\lceil \frac{n}{2} \rceil}$ flow channels, which is theoretically proven to be the maximum number of flow channels that a MUX can independently address [12].

B. General design rules for a microfluidic MUX

In the following, we use the CoMUX shown in Figure 2 as an example to introduce the general design rules of a MUX.

A MUX is a matrix of binary valve patterns on overlapping control and flow channels. We consider every control channel as a row of the matrix, with the topmost control channel as the first row, and every flow channel as a column of the matrix, with the leftmost flow channel as the first column. We denote the entry in the ith row and j^{th} column of the matrix as '0' or '1', depending on whether there is no or one valve at the intersection of the ith control channel and the jth flow channel, respectively. We refer to the entries in the jth column of the matrix as the **code** of the jth flow channel.

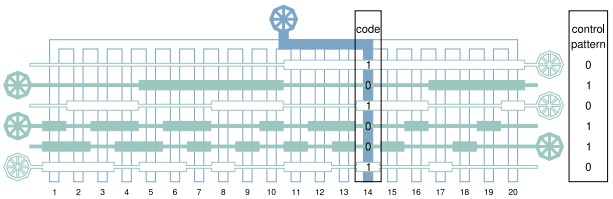


Fig. 2. A CoMUX to address 20 flow channels with 6 control channels.

Example: The code of flow channel 14 *in Figure 2 can be read as* 101001*, representing that the* 1st*,* 3rd *and* 6th *control channels have valves on flow channel* 14*, while the* 2rd*,* 4th *and* 5th *control channels do not have valves on flow channel* 14*.*

We denote the pressure state of a pressurized or depressurized control channel as '1' or '0', respectively, and we refer to the sequence of the pressure states of all control channels to address a flow channel f as the **control pattern** of f.

Example: The code of flow channel 14 in Figure 2 is 101001, and thus the control pattern of flow channel 14 is $\bar{1}\bar{0}\bar{1}\bar{0}\bar{0}\bar{1}$, i.e. 010110, such that all control channels with valves on flow channel 14 are depressurized.

To ensure that fluids can flow from a universal fluid inlet of a MUX to any flow channel without contaminating other flow channels, a MUX design must obey the following rule:

Rule 1*: The control pattern of a flow channel* f *depressurizes all valves on* f *and pressurizes at least one valve on every flow channel other than* f*.*

By negating the code of a flow channel f, the control pattern of f naturally depressurizes all valves on f. Furthermore, it is clear to see that the control pattern of a flow channel f will pressurize a valve on another flow channel f', if there is a control channel that has no valve on f, but one valve on f', i.e. there is an $i \in \mathbb{N}$, such that the ith bit of the code of f is '0' and the ith bit of the code of f' is '1'. S. Liang et al. further proposed and proved the following rule [12]:

Rule 2*: The maximal number of flow channels that a MUX consisting of* n *control channels can independently address is* $C^n_{\lfloor \frac{n}{2} \rfloor}$*. The codes of the flow channels can be achieved by enumerating all* n*-bit binary strings with* $\lfloor \frac{n}{2} \rfloor$ *'*1*'-bits.*

C. Fault model: control-channel defects in a MUX

Microfluidic chips suffer two major types of defects: *blockage* and *leakage*.

- Blockage defects happen at individual control channels. Valves on a blocked control channel cannot be pressurized and thus cannot block the fluids in the corresponding flow channels.
- Leakage defects happen between parallel control channels that are close to each other [17], [18]. Valves on a

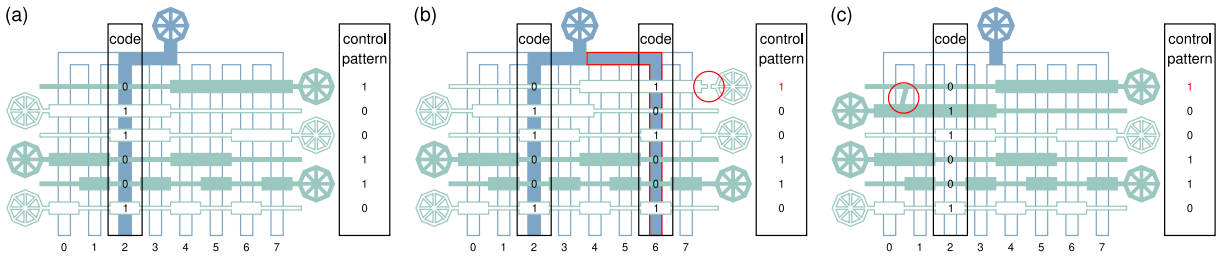


Fig. 3. (a) A standard MUX without defect. The control pattern of flow channel 2 constructs a flow path from the universal fluid inlet to flow channel 2 without contaminating other flow channels. (b) A standard MUX with a blockage defect in the 1st control channel. The control pattern of flow channel 2 contaminates flow channel 6. (c) A standard MUX with a leakage defect between the 1st and the 2nd control channels. The control pattern of flow channel 2 pressurizes the 2nd control channel unintentionally, obstructing the target flow path.

pair of leaky control channels cannot be independently pressurized but have to share the same pressure states.

For example, Figure 3 illustrates the consequences of a single blockage or leakage defect on a standard MUX.

- Suppose that the ith control channel is blocked, a control pattern with a '1' at the ith bit can no longer pressurize the flow channels with a '1' at the ith bit of their codes. For example, the 1st control channel in the MUX in Figure 3(b) is blocked. As a result, the control pattern (100110) of flow channel 2 cannot pressurize flow channel 6 (with code 101001), leading to contamination.
- Suppose that there is a leakage defect between the ith and i'^{th} control channels, a control pattern with a '1' at the ith or i'^{th} bit will inevitably pressurize all flow channels with a '1' at the ith or i'^{th} bit of their codes. For example, the 1st and the 2nd control channels in Figure 3(c) leak with each other. As a result, the control pattern (100110) of flow channel 2 pressurizes the valve at the intersection between flow channel 2 and the 2nd control channel, obstructing the flow path from the universal fluid inlet to flow channel 2.

In general, by studying the working mechanism of state-of-the-art MUXes, we conclude that even under the single-fault assumption, depending on the type and position of the defect, a defective control channel in a standard MUX will affect $50\%-100\%$ of the flow channels addressed by the MUX, and a defective control channel in a CoMUX will affect $33\%-50\%$ of the flow channels addressed by the MUX.

Flow channels affected by the defects become unavailable or at risk of contamination, which leads to the waste of reagents, samples, and chip area. Moreover, the single-fault assumption does not always hold in practice. The risk of defects increases with the integration scale of the chip, and more damage is expected if there are more defective control channels in the MUX. Thus, the design of a fault-tolerant MUX with minimal resource overhead is essential.

III. Problem Formulation

Fault scenario:

- Blockage can happen to any individual control channel.
- Leakage can happen between any two control channels.

Input:

- The number of control channels in a MUX, denoted as n.
- The number of control channel defects that should be tolerated by a MUX, denoted as k.

Output:

- A fault-tolerant MUX design consisting of n control channels to independently address every flow channel in the MUX regardless of k control channel defects.

Objective:

- Maximize the number of flow channels addressed by the fault-tolerant MUX.

IV. Method

A. Designs rules for a fault-tolerant MUX

For convenience, we denote the code of a flow channel f as c_f. As mentioned in Section II-B, a MUX design must obey Rule 1, i.e. the control pattern of a flow channel f depressurizes all valves on f and pressurizes at least one valve on every flow channel other than f. To develop the design rule for a fault-tolerant MUX, we first transform Rule 1 to the following equivalent statement:

Rule 1^**: For any two flow channels* f *and* f'*,* c_f *and* c'_f *must differ by at least two bits, among which there is at least one bit at which* c_f *takes* '1' *and* c'_f *takes* '0' *and one bit at which* c_f *takes* '0' *and* c'_f *takes* '1'*.*

Next, we propose the following rule for designing a MUX that tolerates one blockage or leakage defect:

Rule 3*: For any two flow channels* f *and* f'*,* c_f *and* c'_f *must differ by at least four bits, among which there are at least two bits at which* c_f *takes* '1' *and* c'_f *takes* '0' *and two bits at which* c_f *takes* '0' *and* c'_f *takes* '1'*.*

In the following, we prove with illustrations that if a MUX satisfies Rule 3, as shown in Figure 4(a), it satisfies Rule 1^* even when there is a blockage or leakage defect.

- As shown in Figure 4(b), if a blockage defect happens in a control channel, valves on that channel are depressurized, and the corresponding bits in c_f and $c_{f'}$ should be regarded as '0'. In that case, at least three bits of c_f and $c_{f'}$ remain different, and Rule 1^* is satisfied.
- As shown in Figure 4(c), if a leakage defect happens between two control channels, one of which has a valve

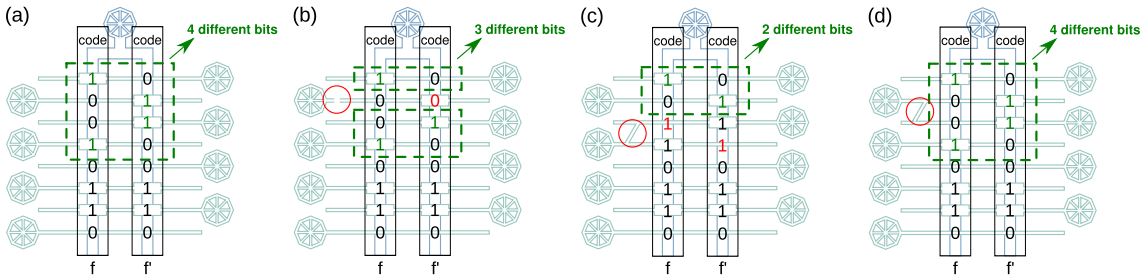


Fig. 4. (a) A design that satisfies Rule 3. (b) Suppose that the 2nd control channel is blocked, valves along the channel cannot be pressurized and should be ignored, i.e., the 2nd bits of c_f and $c_{f'}$ should be regarded as 0. Still, three bits of c_f and c'_f remain different, among which at least one bit of both c_f and c'_f takes '1'. (c) Suppose that the 3rd and the 4th control channels leak with each other, pressurizing one channel will inevitably pressurize the other. Since the control pattern of f requires the 4th control channel to be depressurized, the 3rd control channel must also be depressurized to ensure that f is not obstructed. Thus, the 3rd bit of c_f should be regarded as '1'. Similarly, the 4th bit of $c_{f'}$ should be regarded as '1'. Still, the first two bits of c_f and c'_f remain different, among which one bit of both c_f and c'_f takes '1'. (d) Suppose that the 2nd and the 3rd control channels leak with each other. Since the 2nd and the 3rd bits of c_f are both '0', i.e., the pressure states in the 2nd and the 3rd control channels are the same in the control pattern of f, the leakage does not affect c_f. Similarly, since the 2nd and the 3rd bits of $c_{f'}$ are both '1', the leakage does not affect $c_{f'}$.

on f (f') while the other has no valve on f (f'), both control channels must be depressurized in the control pattern of f (f'), and the corresponding bits in c_f ($c_{f'}$) should be regarded as '1'. In that case, at least two bits of c_f and $c_{f'}$ remain different, and Rule 1^* is satisfied.

- As shown in Figure 4(d), if a leakage defect happens between two control channels, both or neither of which have a valve on f (f'), c_f ($c_{f'}$) does not change and Rule 1^* is satisfied. □

We further generalize Rule 3 into Rule 4 for designing a MUX that tolerates $k \in \mathbb{N}$ blockage or leakage defects:

Rule 4: For any two flow channels f and f', c_f and c'_f must differ by at least $2(k+1)$ bits, among which there are at least $k+1$ bits at which c_f takes '1' and c'_f takes '0' and $k+1$ bits at which c_f takes '0' and c'_f takes '1'.

A proof can be derived using the same strategy as above.

B. Binary constant weight code model

To maximize the number of flow channels addressed by the fault-tolerant MUX, we model the fault-tolerant MUX design problem as a binary constant weight code problem. Specifically, a binary constant weight code with parameters n, w, d is a set $\mathcal{C}$ of binary words of length n and Hamming weight w such that the Hamming distance between any two binary words in $\mathcal{C}$ is at least d [19]. Specifically, a binary word of length n is a string of n bits; the Hamming weight of a binary word is the number of '1'-bits in the binary word; and the Hamming distance between two binary words of equal length is the number of bits at which the two words take different values.

According to Rule 2, the largest set of codes of flow channels in a MUX with n control channels, denoted as $\mathcal{M}_n$, can be achieved by enumerating all n-bit binary strings with $\lfloor \frac{n}{2} \rfloor$ '1'-bits. Since a fault-tolerant MUX is a MUX with extra constraints on the available codes, the codes of the flow channels in a fault-tolerant MUX with n control channels should be a subset of $\mathcal{M}_n$. Thus, we derive the following statement:

To address most flow channels with a fault-tolerant MUX consisting of n control channels, the code of every flow channel should be a binary word of length n and Hamming weight $\lfloor \frac{n}{2} \rfloor$.

We observed that for two binary words w_1 and w_2 of the same length and weight, if their Hamming distance is $2k$, there must be exactly k bits at which w_1 takes '1' and w_2 takes '0' and k bits at which w_1 takes '0' and w_2 takes '1'. Thus, given the same length and Hamming weight of the codes of all flow channels in a MUX, Rule 4 can be simplified as:

Rule 4^*: For any two flow channels f and f', the Hamming distance between c_f and c'_f must be larger than or equal to $2(k+1)$.

Therefore, the problem described in Section III can be modeled as the following binary constant weight code problem:

Finding a largest possible binary constant weight code with length n, weight $\lfloor \frac{n}{2} \rfloor$, and Hamming distance $\geq 2(k+1)$.

The binary constant weight code problem has been extensively studied in the field of information theory [20], [21]. Some upper bounds and lower bounds of the maximum size of a binary constant weight code can be computed [22], but it is generally difficult to find the largest possible code for most parameters. To approximate the theoretical upper bounds, we propose a graph model and transform the binary constant weight code problem into an independent vertex set problem.

Given n control channels and k to-be-tolerated defects, we build a graph $G = (V, E)$, where each vertex $v \in V$ represents a binary word of length n and Hamming weight $\lfloor \frac{n}{2} \rfloor$. If the Hamming distance between the binary words represented by any two vertices v_1 and v_2 is smaller than $2(k+1)$, we add an edge $(v_1, v_2) \in E$. Therefore, any independent vertex set of the graph represents a binary constant weight code with length n, weight $\lfloor \frac{n}{2} \rfloor$, and Hamming distance $\geq 2(k+1)$. The problem of finding a largest possible binary constant weight code is thus equivalent to finding a maximum independent vertex set in G.

The maximum independent vertex set problem is a classic

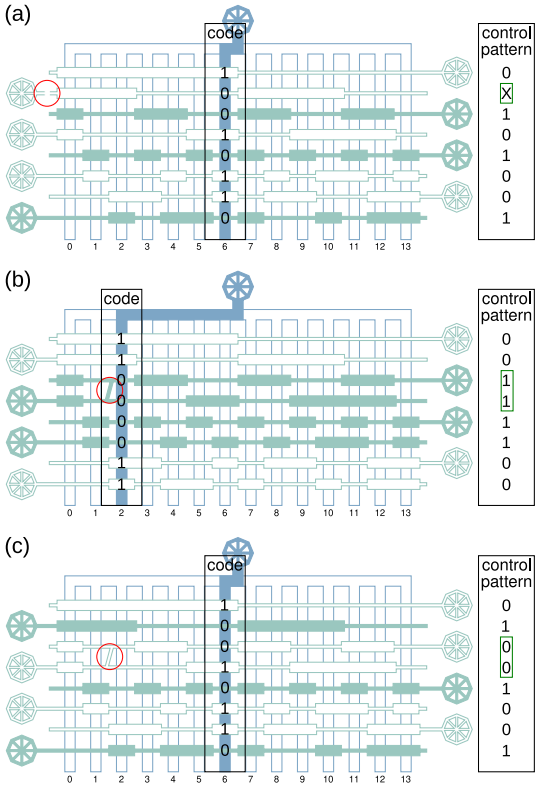


Fig. 5. (a) Control pattern in case of a blockage defect. (b)(c) Control patterns in case of a leakage defect.

NP-hard problem, for which there are numerous studies and algorithms. In this work, we solve the problem with a simple mixed integer linear programming model:

For every vertex $v \in V$, we introduce a binary variable b_v to represent whether v is in the independent vertex set. We then introduce the following constraint for each edge $(v_1, v_2) \in E$:

$$b_{v_1} + b_{v_2} \leq 1,$$

and set the objective function as:

$$\textit{Maximize:} \sum_{v \in V} b_v.$$

C. Control patterns of a fault-tolerant MUX

Based on a binary constant weight code $\mathcal{C}$ with length n, weight $\lfloor \frac{n}{2} \rfloor$, and Hamming distance $2(k+1)$, we can design a MUX that addresses $|\mathcal{C}|$ flow channels with n control channels tolerating k defects, where $|\mathcal{C}|$ denotes the cardinality of set $\mathcal{C}$.

For example, Figure 5 shows a MUX designed based on the largest binary constant weight code with length 8, weight 4, and Hamming distance 4, which addresses 14 flow channels with 8 control channels tolerating one defect. The control pattern to address an arbitrary flow channel with the fault-tolerant MUX depends on the code of the flow channel and the type of defect.

- If there is a blockage defect on any control channel, the control pattern to address a flow channel remains the negation of the code of that flow channel. E.g. the 2nd control channel of the fault-tolerant MUX in Figure 5(a) suffers a blockage defect. Nevertheless, the control pattern to address flow channel 6 with code 10010110 remains $\bar{1}\bar{0}\bar{0}\bar{1}\bar{0}\bar{1}\bar{1}\bar{0}$, i.e. 01101001. In fact, since the second control channel cannot be properly pressurized, we can also denote the second bit of the control pattern as 'X', representing a don't care term.
- If there is a leakage defect between the ith and i'^{th} control channels, e.g. between the 3rd and the 4th control channels, as shown in Figure 5(b) and (c), there are two cases:
  - If the ith and i'^{th} bits of the code of the to-be-addressed flow channel take the same value, i.e. 00 or 11, the control pattern remains the negation of the code of that channel. E.g. as shown in Figure 5(b), the control pattern to address flow channel 2 with code 11000011 remains $\bar{1}\bar{1}\bar{0}\bar{0}\bar{0}\bar{0}\bar{1}\bar{1}$, i.e. 00111100, since the 3rd and the 4th bits of the code of channel 2 are both '0'.
  - If the ith and i'^{th} bits of the code of the to-be-addressed flow channel take different values, i.e. 01 or 10, we change the ith and i'^{th} bits of the control pattern to 00, and keep other bits of the control pattern the negation of the code of that channel. E.g. as shown in Figure 5(c), the control pattern to address flow channel 6 with code 10010110 becomes $\bar{1}\bar{0}0\bar{0}\bar{1}\bar{1}\bar{0}$, i.e. 01001001, since the 3rd and the 4th bits of the code of channel 6 are '0' and '1', respectively.

V. Results

We compare our fault-tolerant MUX design, abbreviated as FT-MUX, with two MUX designs: the mainstream design, referred to as the standard MUX [8] and a novel design, referred to as CoMUX [12]. Since the standard MUX and CoMUX do not tolerate defects, we apply the conventional fault-tolerant design method [13] to duplicate the control channels and valves in these MUXes. Specifically, since it is in general not possible to predict the position of defects, to tolerate one or two defects, all control channels in a standard MUX/CoMUX need to be doubled or tripled, respectively. We denote MUXes that tolerate one or two defects by adding (I) or (II) next to their names, respectively.

To achieve the optimal code for our FT-MUX, we run the mixed integer linear programming model on a computer with two Xeon Gold 6126 2.6 GHz processors. We achieved the optimal solutions, i.e. the largest possible codes, for FT-MUX(I) and FT-MUX(II) with up to 12 control channels within a few seconds. Given more than 12 control channels, the model cannot guarantee the optimality of the solution. In those cases, we adopt the constant weight code table published in [23]. The results of the comparison are shown in Figure 6. Based on the comparison, we make the following conclusions:

- FT-MUX greatly improved the resource efficiency compared to the conventional fault-tolerant design method.

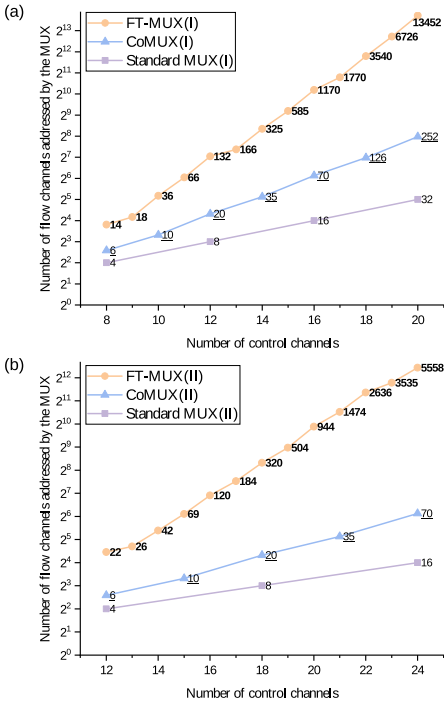


Fig. 6. Comparison among (a) MUXes that tolerate one defect; (b) MUXes that tolerate two defects.

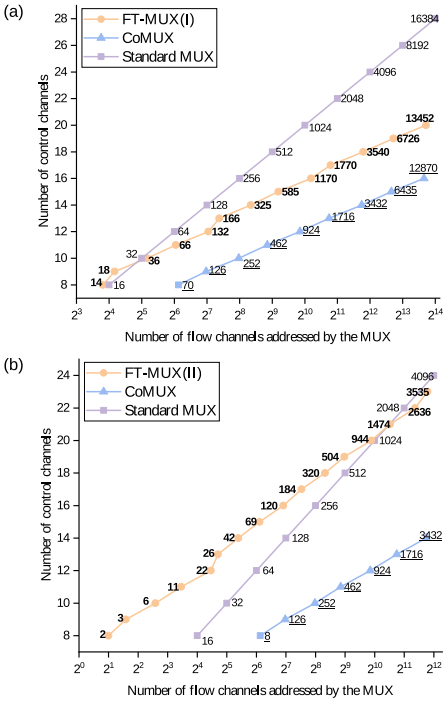


Fig. 7. Comparison among state-of-the-art MUXes that tolerate no defect and (a) FT-MUXes that tolerate one defect; (b) FT-MUXes that tolerate two defects.

In particular, the efficiency advantages increase with the integration scale. Given 12 control channels, FT-MUX(I) addresses 6 and 16 times more flow channels than CoMUX(I) and standard MUX(I), respectively; and given 20 control channels, FT-MUX(I) addresses 53 and 420 times more flow channels than CoMUX(I) and standard MUX(I), respectively.

- FT-MUX is the only reasonable MUX design to tolerate more than one defect. In case two defects need to be tolerated, with up to 18 control channels, CoMUX(II) and standard MUX(II) require a similar or even larger number of control channels than the to-be-addressed flow channels, making the multiplexing meaningless. In the meanwhile, FT-MUX(II) can still reliably address many flow channels with only a few control channels.

Besides, given 24 control channels, FT-MUX(III) can tolerate three defects while addressing 2576 flow channels; and given 32 control channels, FT-MUX(IV) can tolerate four defects while addressing 3038 flow channels.

To evaluate the resource usage for implementing the fault-tolerant features, we also compare FT-MUXes that tolerate one or two defects to the standard MUXes and CoMUXes without fault-tolerant features. Figure 7 shows the results of the comparison.

- Comparison to standard MUXes. Given no less than 10 control channels, FT-MUX(I) tolerates one defect while addressing more flow channels than a standard MUX that tolerates no defect; and given no less than 22 control channels, FT-MUX(II) tolerates two defects while addressing more flow channels than a standard MUX that tolerates no defect. In other words, FT-MUX significantly outperforms the standard MUX, not only in reliability but also in resource efficiency.
- Comparison to CoMUXes. In general, to address a similar number of flow channels, FT-MUXes that tolerate one defect use 3 to 4 more control channels than CoMUXes that tolerate no defect; and FT-MUXes that tolerate two defects use 7 to 9 more flow channels than CoMUXes that tolerate no defect. It is worth noticing that the resource for implementing the fault-tolerant features does not increase significantly with the number of to-be-addressed flow channels, demonstrating good scalability of FT-MUXes.

VI. Conclusion

Fluid multiplexing is one of the most important operations on microfluidic chips. The reliability of fluid multiplexers, or MUXes, has thus been drawing more and more attention in recent years. This paper proposes the design rules for a fault-tolerant MUX (FT-MUX) for the first time and models the FT-MUX design problem as a constant weight code problem to minimize resource overheads. We demonstrated that FT-MUX significantly outperforms the state-of-the-art MUX designs and fault-tolerant design methods. As the integration scale of microfluidic chips increases, FT-MUX enables an efficient and reliable solution to execute microfluidic applications.

Acknowledgment

This work is supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation, No. 515003344).

REFERENCES

[1] Marc A Unger, Hou-Pu Chou, Todd Thorsen, Axel Scherer, and Stephen R Quake. Monolithic microfabricated valves and pumps by multilayer soft lithography. *science*, 288(5463):113–116, 2000.

[2] Jessica Melin and Stephen R Quake. Microfluidic large-scale integration: the evolution of design rules for biological automation. *Annu. Rev. Biophys. Biomol. Struct.*, 36:213–231, 2007.

[3] Nina Compera, Scott Atwell, Johannes Wirth, Christine von Törne, Stefanie M Hauck, and Matthias Meier. Adipose microtissue-on-chip: a 3d cell culture platform for differentiation, stimulation, and proteomic analysis of human adipocytes. *Lab on a Chip*, 22(17):3172–3186, 2022.

[4] Mohamed Ibrahim, Krishnendu Chakrabarty, and Ulf Schlichtmann. Cosyn: Efficient single-cell analysis using a hybrid microfluidic platform. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1673–1678. IEEE, 2017.

[5] Brooke Schuster, Michael Junkin, Sara Saheb Kashaf, Isabel Romero-Calvo, Kori Kirby, Jonathan Matthews, Christopher R Weber, Andrey Rzhetsky, Kevin P White, and Savaş Tay. Automated microfluidic platform for dynamic and combinatorial drug screening of tumor organoids. *Nature communications*, 11(1):5271, 2020.

[6] Roberto Rodriguez-Moncayo, Diana F Cedillo-Alcantar, Pablo E Guevara-Pantoja, Oriana G Chavez-Pineda, Jose A Hernandez-Ortiz, Josue U Amador-Hernandez, Gustavo Rojas-Velasco, Fausto Sanchez-Muñoz, Daniel Manzur-Sandoval, Luis D Patino-Lopez, et al. A high-throughput multiplexed microfluidic device for covid-19 serology assays. *Lab on a Chip*, 21(1):93–104, 2021.

[7] Tsun-Ming Tseng, Mengchu Li, Daniel Nestor Freitas, Amy Mongersun, Ismail Emre Araci, Tsung-Yi Ho, and Ulf Schlichtmann. Columba s: A scalable co-layout design automation tool for microfluidic large-scale integration. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018.

[8] Todd Thorsen, Sebastian J Maerkl, and Stephen R Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.

[9] Mengchu Li, Yushen Zhang, Ju Young Lee, Hudson Gasvoda, Ismail Emre Araci, Tsun-Ming Tseng, and Ulf Schlichtmann. Integrated test module design for microfluidic large-scale integration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[10] Siyuan Liang, Meng Lian, Mengchu Li, Tsun-Ming Tseng, Ulf Schlichtmann, and Tsung-Yi Ho. Armm: Adaptive reliability quantification model of microfluidic designs and its graph-transformer-based implementation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2023.

[11] Qin Wang, Shiliang Zuo, Hailong Yao, Tsung-Yi Ho, Bing Li, Ulf Schlichtmann, and Yici Cai. Hamming-distance-based valve-switching optimization for control-layer multiplexing in flow-based microfluidic biochips. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 524–529. IEEE, 2017.

[12] Siyuan Liang, Mengchu Li, Tsun-Ming Tseng, Ulf Schlichtmann, and Tsung-Yi Ho. Comux: Combinatorial-coding-based high-performance microfluidic control multiplexer design. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2022.

[13] Wei-Lun Huang, Ankur Gupta, Sudip Roy, Tsung-Yi Ho, and Paul Pop. Fast architecture-level synthesis of fault-tolerant flow-based microfluidic biochips. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 1667–1672. IEEE, 2017.

[14] Ying Zhu, Bing Li, Tsung-Yi Ho, Qin Wang, Hailong Yao, Robert Wille, and UIf Schlichtmann. Multi-channel and fault-tolerant control multiplexing for flow-based microfluidic biochips. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8. IEEE, 2018.

[15] Ying Zhu, Xing Huang, Bing Li, Tsung-Yi Ho, Qin Wang, Hailong Yao, Robert Wille, and Ulf Schlichtmann. Multicontrol: Advanced control-logic synthesis for flow-based microfluidic biochips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(10):2489–2502, 2019.

[16] Kai Hu, Feiqiao Yu, Tsung-Yi Ho, and Krishnendu Chakrabarty. Testing of flow-based microfluidic biochips: Fault modeling, test generation, and experimental demonstration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1463–1475, 2014.

[17] Jiaxing Wang, Mingxin Zheng, Wei Wang, and Zhihong Li. Optimal protocol for moulding pdms with a pdms master. *Chips & Tips (Lab on a Chip)*, 6, 2010.

[18] Mengchu Li, Hanchen Gu, Yushen Zhang, Siyuan Liang, Hudson Gasvoda, Rana Altay, Ismail Emre Araci, Tsun-Ming Tsneg, Tsung-Yi Ho, and Ulf Schlichtmann. Late breaking results: Efficient built-in self-test for microfluidic large-scale integration (mlsi). In *Proceedings of the 62th Annual Design Automation Conference*, pages 1–6, 2024.

[19] Kenneth S Suslick. Encyclopedia of physical science and technology. *Sonoluminescence and sonochemistry, 3rd edn. Elsevier Science Ltd, Massachusetts*, pages 1–20, 2001.

[20] A. E. Brouwer, J. B. Shearer, N. J. A. Sloane, and W. D. Smith. A new table of constant weight codes. *IEEE Trans. Inf. Theory*, 36:1334–1380, 1990.

[21] Derek H Smith, Lesley A Hughes, and Stephanie Perkins. A new table of constant weight codes of length greater than 28. *the electronic journal of combinatorics*, 13(1):A2, 2006.

[22] Selmer Johnson. A new upper bound for error-correcting codes. *IRE Transactions on Information Theory*, 8(3):203–207, 1962.

[23] Bounds for binary constant weight codes. https://www.win.tue.nl/ aeb/codes/Andw.html. Accessed: 2024-07-05.

PIMPAL: Accelerating LLM Inference on Edge Devices via In-DRAM Arithmetic Lookup

Yoonho Jang*, Hyeongjun Cho†, Yesin Ryu‡§, Jungrae Kim*, Seokin Hong*
*Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Republic of Korea
†Department of Semiconductor Convergence Engineering, Sungkyunkwan University, Suwon, Republic of Korea
‡Department of Semiconductor and Display Engineering, Sungkyunkwan University, Suwon, Republic of Korea
§Samsung Electronics, Hwaseong, Republic of Korea
{jyh8807, cho0624ck}@skku.edu {seleneyou}@g.skku.edu {dale40, seokin}@skku.edu

***Abstract*—Deploying Large Language Models (LLMs) on edge devices poses significant challenges due to their high computational and memory demands. In particular, General Matrix-Vector Multiplication (GEMV), a key operation in LLM inference, is highly memory-intensive, making it difficult to accelerate using conventional edge computing systems. While Processing-in-memory (PIM) architectures have emerged as a promising solution to this challenge, they often suffer from high area overhead or restricted computational precision.**

This paper proposes *PIMPAL (Processing-In-Memory architecture with Parallel Arithmetic Lookup)*, a cost-effective PIM architecture leveraging LookUp Table (LUT)-based computation for GEMV acceleration in sLLMs (small LLMs). By replacing traditional arithmetic operations with parallel in-DRAM LUT lookups, PIMPAL significantly reduces area overhead while maintaining high performance. PIMPAL introduces three key innovations: (1) it divides DRAM bank subarrays into compute blocks for parallel LUT processing; (2) it employs *Locality-aware Compute Mapping (LCM)* to reduce row activations by maximizing LUT access locality; and (3) it enables multi-precision computations through a *LUT Aggregation (LAG)* mechanism that combines results from multiple small LUTs. Experimental results show that PIMPAL achieves up to 17.8x higher performance than previous LUT-based PIM designs and reduces area overhead by 40% compared to conventional processing unit-based PIM designs.

I. Introduction

Transformer-based Large Language Models (LLMs) are driving significant advancements in Natural Language Processing (NLP) and transforming diverse applications. However, their massive computational and memory requirements restrict deployment to high-performance data centers equipped with specialized hardware like GPUs and TPUs. To address these challenges, Small LLMs (sLLMs) have emerged as compact alternatives [22, 25, 26, 29]. These lightweight models significantly reduce parameter counts compared to full-scale LLMs, making LLM inference feasible on resource-constrained devices such as smartphones and IoT systems.

Although sLLMs are compact and efficient, they still face significant challenges in accelerating General Matrix-Vector Multiplication (GEMV), a core operation in both LLMs and sLLMs. GEMV is memory-intensive, making it difficult to optimize on conventional computing systems with limited memory bandwidth [4, 27, 28]. In data centers, LLMs mitigate this bottleneck by replacing GEMV with General Matrix-Matrix Multiplication (GEMM), which is a compute-intensive alternative, through input batching. However, this optimization is infeasible on edge devices, where only a single input is typically available during inference.

Recently, many prior works have proposed Processing-In-Memory (PIM) architectures to accelerate memory-intensive workloads, such as GEMV, by performing computations within memory to exploit high internal bandwidth [1, 5, 9, 15, 20, 21]. One notable example is the Processing Unit (PU)-based approach [9, 15], where numerous PUs are integrated within or near memory banks. Prototype devices developed on commodity DRAMs like HBM2 [15] and GDDR6 [9] demonstrate the significant acceleration of GEMV for deep neural network workloads. However, despite these performance gains, PU-based PIM incurs substantial area overhead, reducing memory capacity by 50% in the prototypes. This overhead mainly arises from the inherent differences between memory and logic fabrication processes, such as larger feature sizes and fewer metal layers in memory-focused processes [5].

Another PIM architecture, LookUp Table (LUT)-based PIM [6, 23, 30], performs computations by leveraging precomputed results stored in a LUT within memory. Unlike PU-based approaches, LUT-based PIM does not require additional processing units; instead, it directly looks up values stored in memory cells, enabling the implementation of complex functions with minimal area overhead. Furthermore, by modifying the values stored in the LUT, this architecture can flexibly support a wide range of functions without requiring changes to the hardware design [6, 30]. These advantages make LUT-based PIM a cost-effective and versatile solution.

However, LUT-based PIM faces two major challenges. First, storing LUTs in DRAM cells often leads to frequent row activations, which are time-consuming and power-intensive [8, 17], especially when large LUTs span multiple DRAM rows. This problem is exacerbated when the inputs to the arithmetic function are randomly distributed, resulting in poor row buffer locality. Second, supporting high-precision arithmetic requires large LUTs. For instance, a LUT for multiplying two 8-bit integers requires 128KB ($2^8 \times 2^8 \times 16$ bits), which far exceeds the typical DRAM row size (i.e., 1 $\sim$2KB). As a result, such large LUTs must be distributed across multiple rows, further increasing row activation frequency.

This paper presents ***PIMPAL***, a novel LUT-based PIM

architecture specifically designed to accelerate the GEMV operations of sLLMs. PIMPAL employs a minimal number of PUs and replaces complex operations with highly parallel LUT lookups. By replacing substantial portions of the PUs with LUTs, it achieves high performance with low area overhead. PIMPAL addresses the key challenges of prior LUT-based PIM architectures via its parallel organization and three novel mechanisms as follows:

Organization for Subarray-level Parallel Lookup: PIMPAL divides a DRAM bank's subarray into compute blocks, each consisting of multiple subarrays. Within each compute block, one subarray is responsible for supplying data, while another subarray is dedicated to storing the LUT. This organization simplifies LUT access and ensures efficient data retrieval. By enabling parallel processing across multiple compute blocks, PIMPAL provides a scalable framework that supports high-throughput computation.

Locality-aware Compute Mapping: To minimize DRAM row activations during GEMV computations, PIMPAL employs *Locality-aware Compute Mapping (LCM)*. This mechanism uses a column-major GEMV approach, where each vector element is multiplied by all elements in its corresponding matrix column. LCM distributes matrix columns across compute blocks, assigning each block to calculate individual partial products. This mapping allows an activated LUT row for a vector element to be reused multiple times, significantly improving the locality of LUT data stored in the DRAM row buffer within each compute block.

LUT Aggregation: To support multi-precision operations such as INT8 and BF16, PIMPAL introduces the *LUT Aggregation (LAG)* mechanism. For floating-point operations, LAG combines lookup results from two LUTs for exponential and mantissa bits. By using multiple small LUTs, PIMPAL supports high-precision computations while significantly reducing LUT size. For example, the BF16 LUT entries are reduced from 2^{32} elements to 2^{16} elements. When the LUTs are small enough to fit within a single subarray, PIMPAL stores them together and enables partial row activation to access individual LUTs efficiently, minimizing area overhead.

Our experimental results show that PIMPAL achieves a 17x speedup over previous LUT-based PIM architectures and delivers nearly 25% higher performance per area compared to PU-based PIM architectures for GEMV operations in sLLMs.

II. Background and Motivation

A. GEMV Operation in LLMs

LLMs are highly memory-intensive due to their architectural design, which relies heavily on fully connected (FC) layers implemented using General matrix-vector multiplication (GEMV). Although GEMV involves simple Multiply-and-Accumulate (MAC) computations, its low data reuse rate imposes substantial memory bandwidth demands. Table I shows the proportion of inference time spent on FC layers for various sLLMs on the NVIDIA A100 GPU. Despite the high-performance GPU's use of high-bandwidth memory, FC layers account for over 60% of the total model inference time. These experimental results indicate that GEMV constitutes the majority of sLLM inference workloads, emphasizing the need to optimize this operation for better performance.

TABLE I: The sizes of sLLMs and the proportion of GEMV in their inference time when using one input on an Nvidia A100 GPU.

	OPT-350m	**Gemma-2**	**Llama-3.2**	**MobileBert**
Model size	700 MB	2 GB	2 GB	101 MB
FC layers	61.2 %	62.4 %	63.5 %	73.2 %

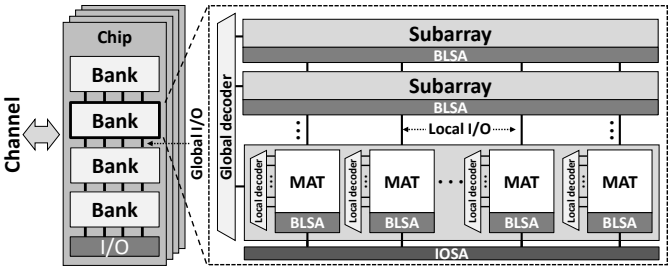


Fig. 1: DRAM Organization.

B. DRAM Organization and Operations

Figure 1 illustrates the hierarchical structure of DRAM. A DRAM channel is a logical construct defined by the memory controller that interfaces with memory. Each channel operates independently, allowing for parallel data transfer. A channel can be connected to one or more DRAM chips, depending on the memory module configuration and standards like DDR, GDDR, or LPDDR. Each chip consists of multiple banks, which can be accessed independently to increase memory bandwidth through bank-level parallelism. Each bank's memory array is further divided into subarrays.

When accessing data, a global decoder selects a subarray, and a local decoder activates a specific row within it. Each subarray consists of multiple MATs, which provide data from the activated row. The row's data is latched in the Bitline Sense Amplifiers (BLSA or Row Buffer). To read or write data, a column decoder selects the column in the BLSA, and the MATs transfer data to or from the IOSA via the Local I/O path. When a row within a bank is activated, it must be precharged to activate another row. However, if the requested row is already activated and its data resides in the BLSA, subsequent accesses bypass the activation and precharge steps. This optimized access, known as a row buffer hit, significantly reduces latency and is essential for achieving low data access times and reducing energy consumption.

C. Limitations of Prior PIM Architectures

1) PU-based PIM Architectures: Processing Unit (PU)-based PIM architectures integrate numerous processing units within or near memory banks, enabling data to be accessed directly from memory cells without relying on the external I/O bus. This design leverages the massive internal memory bandwidth of DRAM, achieving state-of-the-art computation speeds for GEMV acceleration. Recently, DRAM vendors have prototyped PU-based PIM devices using commodity DRAMs, such as HBM2 [15] and GDDR6 [9], demonstrating their potential for high-performance computing applications.

Permanent Capacity Loss: Despite their high performance, these architectures sacrifice approximately 50% of memory

TABLE II: Comparison of LUT-based PIM architectures.

Architecture	LUT Storage	# of row activation per 2K MUL.	LUT bit Precision	INT8 / BF16 GEMV Support
pPIM [23]	Additional Buffer	None	8-bit	No
pLUTo [6]	DRAM cells	256	8-bit	No
ReD-LUT [30]	DRAM cells	1 to 2K	8-bit	No
PIMPAL (This work)	DRAM cells	1	16-bit	Yes

capacity to accommodate processing units [13, 14, 15]. This significant area overhead arises because logic circuits require larger space when implemented using a DRAM fabrication process compared to the standard logic fabrication process [5]. The integration of these processing units occupies space otherwise allocated to memory cells, resulting in a permanent reduction in capacity. To compensate for this loss, additional DRAM chips are required to meet capacity demands, which poses challenges for systems with constrained physical space for memory modules, such as edge devices.

2) LUT-based PIM Architecture: LUT-based PIM architectures leverage Look-Up Tables (LUTs) to enable complex computations with minimal changes to DRAM organization. These architectures store pre-computed data for specific functions and retrieve results directly from the tables, requiring only storage for LUT elements and simple decoding logic. This flexible approach supports various functions by writing different data into the tables. Given these advantages, several LUT-based PIM architectures have been proposed, as summarized in Table II. pPIM adds buffers for LUT storage near DRAM subarray and retrieves elements using operands from BLSA [23]. In contrast, pLUTo [6] and ReD-LUT [30] store LUTs in DRAM cells but access them differently. pLUTo activates all LUT rows in a subarray and reads multiple LUT elements by matching operand values with row numbers, while ReD-LUT activates a single row using one operand as the row address and retrieves the specific element by using another operand as the column address.

Frequent Row Activation: Prior LUT-based PIM designs suffer from low performance due to frequent DRAM row activations, which are time-consuming operations in DRAM systems. A key issue is that these designs fail to exploit row buffer hits during LUT access. For example, pLUTo's row-sweeping operation activates all LUT rows, causing excessive DRAM row activations. Similarly, ReD-LUT activates a single row per computation but produces only one element per activation, leading to high row activation overhead for large datasets. As discussed in Section II-B, DRAM systems can bypass row activation if the requested row is already activated. Therefore, improving spatial and temporal localities of LUT elements within the row buffer is essential to reducing DRAM row activations and enhancing performance.

Low Bit-Precision: Another critical challenge is the limited bit-precision of LUTs, which restricts their applicability to typical DNN workloads. Most models require at least 8-bit MAC operations, as INT8 is the standard arithmetic precision for inference tasks, necessitating a 16-bit LUT to handle the multiplication of two 8-bit operands. However, prior PIM architectures provide only 8-bit LUTs, which is insufficient for core NN operations. While pPIM achieves 8-bit MAC operations by combining outputs, it introduces significant area overhead due to the need for numerous latches and complex routing logic. The increasing adoption of BF16 precision, which delivers FP32-level accuracy in 16 bits, further exacerbates this issue, as it demands large LUTs with massive entry sizes ($2^{16} \times 2^{16} \times 16$-bit = 8GB). Hence, it is essential to provide high bit-precision LUT with low area overhead.

III. PIMPAL

This section introduces ***PIMPAL*** *(Processing-In-Memory architecture with Parallel Arithmetic Lookup)*, a novel LUT-based PIM architecture specifically designed to accelerate the GEMV operations. PIMPAL is structured to have a parallel organization for enabling parallel lookup within a DRAM bank and incorporates two novel mechanisms: *Locality-aware Compute Mapping (LCM)* and *LUT Aggregation (LAG)*.

A. Locality-aware Compute Mapping

To minimize DRAM row activations during GEMV computations, PIMPAL employs *Locality-aware Compute Mapping (LCM)*, which utilizes a column-major GEMV approach illustrated in Figure 2. In this approach, each matrix column is multiplied by its corresponding vector element (steps ❶ to ❸) to generate partial products, which are then accumulated to produce the final results. Using this method, LCM activates a DRAM row containing LUT row for a given vector element (e.g., value 2), as shown in Figure 3. It then retrieves LUT elements from the activated row consecutively for all matrix elements (e.g., values 3, 1, and 2) of the corresponding column. This approach limits LUT row activations to the vector size, as a vector element is reused across all elements in its matrix column during each step. For the example in Figure 2, LCM involves only three LUT row activations.

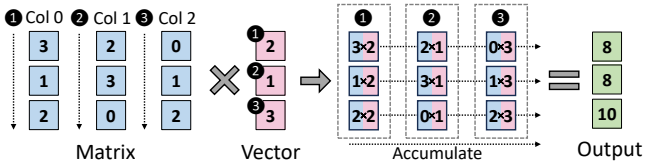


Fig. 2: Column-major GEMV.

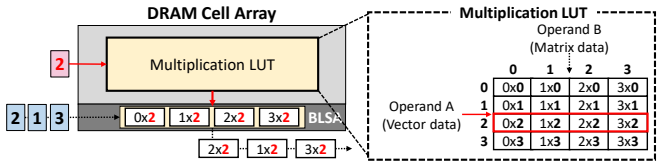


Fig. 3: Locality-aware Compute Mapping leveraging Column-major GEMV. The DRAM row size is assumed to be the same as the LUT row size.

B. PIMPAL Organization and Operation

PIMPAL organizes each DRAM bank into compute blocks, where two subarrays are grouped to form one compute block. Within a compute block, one subarray supplies input data for computations, while the other provides Look-Up Table (LUT) elements, enabling efficient data retrieval and processing. This organization also maximizes computation throughput with parallel LUT lookup by exploiting subarray-level parallelism [11].

Figure 4 illustrates PIMPAL's bank architecture. Each compute block comprises two adjacent subarrays, one storing

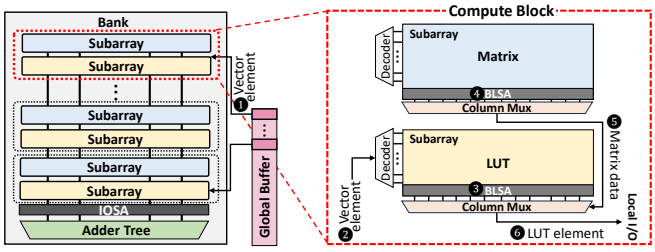


Fig. 4: Bank architecture of PIMPAL.

matrix data and the other storing LUT elements. A global buffer, shared across banks, contains vector elements that are distributed to the compute blocks within the banks (❶). For each vector element (❷), a DRAM row in the LUT subarray is activated, latching the corresponding LUT row in the BLSA (❸). Next, a row in the matrix subarray, storing the matrix column to be multiplied by the vector element, is activated (❹). The selected matrix column element is then used to retrieve a specific LUT element from the BLSA of the LUT subarray (❺). The retrieved LUT element is transferred via the local I/O path to the IOSA (❻). Since the LUT element size (e.g., 16-bit) is smaller than the typical column size, each compute block uses only a subset of the available local I/O path lines. The IOSA receives LUT elements from the compute blocks and transfers them into the adder tree for fast accumulation.

Figure 5 shows overall GEMV processing in compute block 0 for the example GEMV shown in Figure 2. As illustrated in the Figure, the compute block 0 is assigned the computation for matrix column 0. Thus, the first element of the vector is used as the LUT row address. As the value of the element is 2, the DRAM row containing the LUT row 2 is activated (①). Subsequently, the compute block activates the DRAM row corresponding to the matrix column correlated with the vector element in the matrix subarray (②). After activating these two rows, the compute block decodes the matrix column data and transfers it to the LUT subarray via an additional local path. Finally, the column decoder of the LUT subarray uses the matrix column data to select the corresponding element (③) and sends it to the adder tree for accumulation alongside the partial products computed by other compute blocks.

The accumulator in PIMPAL utilizes a specialized adder tree inspired by Newton [13]. This adder tree aligns the mantissa of the multiplier results with the largest exponent among them, allowing the mantissa to be accumulated as integers. This design facilitates fast processing while maintaining a minimal accuracy drop. Furthermore, the adder tree employs 24-bit adders, enabling compatibility with BF16 and 16-bit integer addition when INT8 precision is used as the mantissa.

C. LUT Aggregation

To support high-precision computations while minimizing LUT size and area overhead, PIMPAL employs the *LUT Aggregation (LAG)* mechanism. LAG decomposes complex arithmetic operations, such as high-precision multiplication, into smaller sub-operations processed by multiple small LUTs, then aggregates lookup results from the tables to achieve the desired precision. With this approach, PIMPAL enables efficient support for high-precision arithmetic like BF16. This subsection explains the arithmetic decomposition and the mapping process of multiple LUTs to compute blocks.

1) Arithmetic Decomposition: Floating-point operations, such as multiplication, involve independent calculations for the mantissa and exponent components of the operands. This property enables an efficient Arithmetic Decomposition technique, where separate LUTs are used for each component. Specifically, a dedicated LUT handles the mantissa multiplication, while another handles the exponent addition and bias adjustment. By splitting the computation in this manner, the complexity and size of each LUT are significantly reduced. For example, a BF16 LUT is decomposed into two separate LUTs (Mantissa LUT and Exponent LUT), as shown in Figure 6. With this decomposition, the total size of BF16 LUT is reduced from 8GB to 128KB.

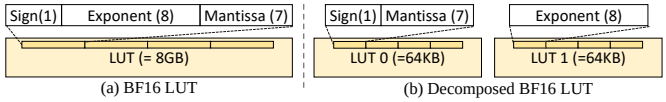


Fig. 6: LUT size reduction with arithmetic decomposition. Arithmetic decomposition reduces the LUT size for BF16 multiplication from 8GB to 128KB.

2) LUT Mapping: We now describe how the LUTs are mapped to compute blocks for different precisions. As described above, the arithmetic decomposition splits a LUT to the exponent and mantissa LUTs. Figure 7a illustrates the data format of LUT elements for INT16 and BF16 data. Figure 7b and Figure 7c show how the compute block stores the LUTs and accesses them for INT16 and BF16 data, respectively. For this explanation, we assume a subarray configuration consisting of 1024 rows, each with a 2KB capacity.

INT16: For supporting the precise multiplication of INT8 operands, it is necessary to support 16-bit precision. Therefore,

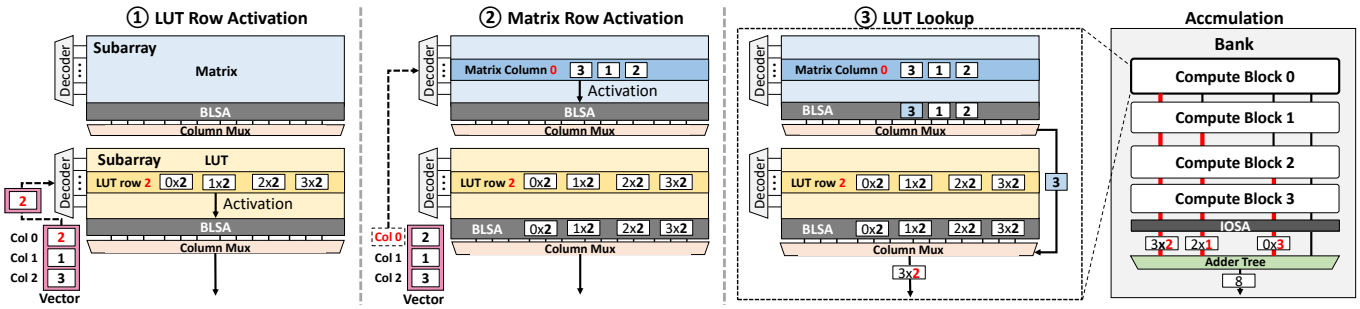


Fig. 5: Overall operation of GEMV processing in the compute block and partial product accumulation. The operation sequence proceeds from left to right. We assume three compute blocks are involved in the processing.

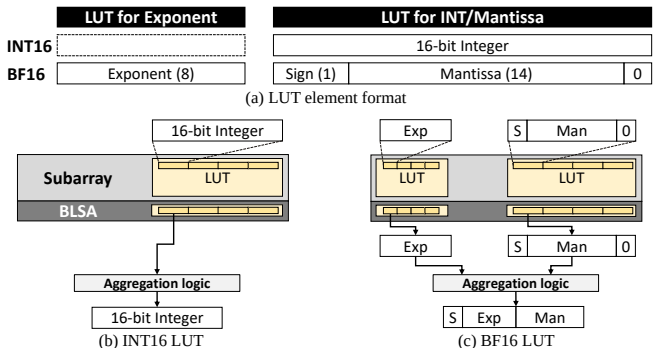


Fig. 7: Multi-precsion support with different LUT mappings within compute block.

PIMPAL stores the LUT elements as 16-bit values. These elements function as the mantissa in the adder tree. Since integer types have no exponent value, they do not use a LUT for the exponent part. This 16-bit LUT can also be used for any function requiring 16-bit data for a single operand. As the size of 16-bit LUT is 128KB ($2^8 \times 2^8 \times$16-bit), the compute block needs only one subarray for storing the LUT (Figure 7b).

BF16: To store BF16 data in the LUT, we must accommodate its 8-bit exponent and 7-bit mantissa. The exponent LUT utilizes an 8-bit format, as the exponent cannot exceed 8 bits in floating point operations. Meanwhile, we employ a 14-bit format for the mantissa LUT to cover the 7-bit mantissa multiplication and account for potential rounding during computations. Additionally, we add a 1-bit sign in the mantissa, resulting in a 15-bit format for the mantissa LUT entries. Adding the sign bit does not significantly increase the number of mantissa LUT entries, as it only requires $2^8 \times 2^8$ entries. To facilitate easy decoding of multiple data formats, we zero-pad the mantissa elements to make them a multiple of 8 bits, resulting in a 16-bit type. Like the INT16 case, a single subarray can store both the mantissa LUT and the exponent LUT (Figure 7c). The compute block looks up these two LUTs and then aggregates the results retrieved from the LUTs to construct the final BF16 result.

As observed above, the compute block stores the LUT only part of the subarray in most cases. To activate the needed part of a DRAM row to fetch a LUT row, we employ a technique called Partial Row Activation (PRA) [16], which allows activation of part of the row within the subarray. PRA activates the row segments in the MATs where the corresponding bit in the PRA latch is set to 1. We refer to this method as Partial LUT Activation (PLA), as depicted in Figure 8. PLA enables multiple LUT rows in different parts to be activated sequentially, preventing LUT rows from overwriting each other.

D. Optimization for LUT fetching

Before starting GEMV operations, all compute blocks require the LUT for multiplication. Storing identical LUT rows in each compute block is inefficient, reducing subarray capacity by 12.5% to 25% with a 16-bit LUT following the DRAM configuration. This overhead is especially problematic for normal memory operations, as it limits available capacity.

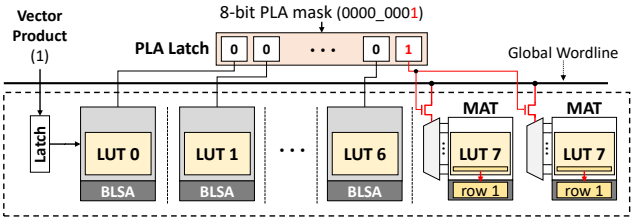


Fig. 8: Partial LUT Activation for fetching 8-bit LUT data. A dotted box represents a subarray.

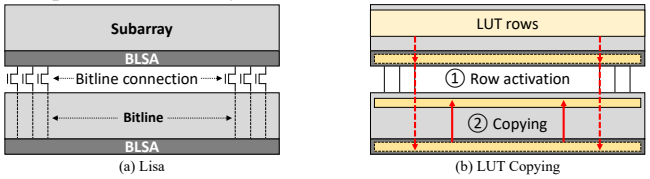


Fig. 9: (a) LISA technique [3] and (b) fast LUT copying with LISA. A red line represents a row activation.

To address this issue, we store the LUT in a single subarray per bank and use the Low-Cost Interlinked Subarrays (LISA) technique to copy it at the start of PIM operations. This technique interconnects the bitline of each subarray with another using a single transistor and transfers the activated row data into the connected subarray's BLSA, illustrated in Figure 9. By adopting this approach, the LUT copying step can be executed quickly at the start of the PIM operation.

IV. Evaluation

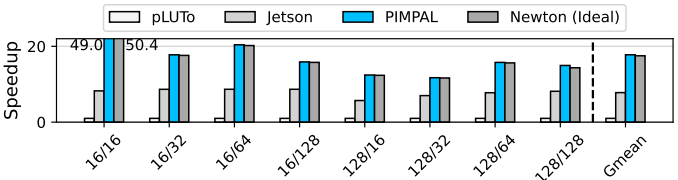
A. Methodology

To evaluate the performance of PIMPAL, we implement PIM architectures on Ramulator, a cycle-level DRAM simulator [7, 12], following the configuration of a memory product used in edge devices (Table III) [10]. We compare two PIM architectures with PIMPAL: Newton and pLUTo. We configure them with the same memory configuration for fairness. We also compare PIMPAL with NVIDIA Jetson Xavier GPU [19], which is popularly used to accelerate NN tasks on edge devices. GPU performance metrics are gathered from real-time tests, with the configurations in Table III.

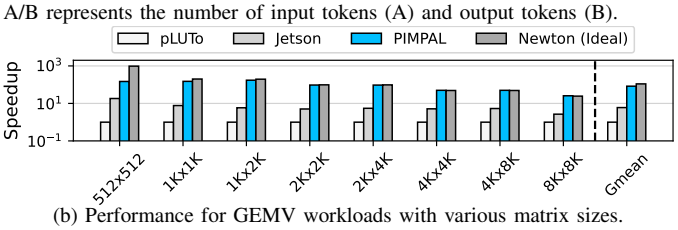
For workloads, we use Gemma-2 2B, a well-known sLLM from Google that supports multiple precisions (from FP32 to INT4) [25]. We evaluate the workload's performance by measuring the end-to-end inference time. For this evaluation, GEMV layers are processed with PIM, while other layers are processed using the Jetson platform. We also use synthetic GEMV workloads commonly found in LLMs. To ensure consistency, we use INT8 precision for Gemma-2, while pLUTo is evaluated at INT4 precision due to its limited LUT size.

TABLE III: Configurations.

LPDDR5-6400 Configuration	
Channel & Rank	4 & 1 per channel
Bank group & Bank	4 per channel & 4 per bank group
Subarray & Row size	64 per bank & 2KB
tCL-tRCD-tRP & tRRD, tFAW	17-18-18 & 5ns, 20ns
Technique	**Parallelism**
Newton (PU-based)	16 MAC units per bank
pLUTo (LUT-based)	32 LUT units per bank
PIMPAL (LUT-based)	16 Compute blocks per bank
NVIDIA Jetson AGX Xavier	
Core	512 CUDA & 64 Tensor core
Memory	64 GB



(a) End-to-end performance for various numbers of tokens. On the X-axis, A/B represents the number of input tokens (A) and output tokens (B).



(b) Performance for GEMV workloads with various matrix sizes.

Fig. 10: Performance Comparison.

B. Performance

Figure 10a shows the normalized performance relative to pLUTo for end-to-end inference tasks. The results indicate that pLUTo has the lowest performance due to frequent LUT row activations. In contrast, Newton and PIMPAL achieve substantial speedups of 17.5x and 17.8x over pLUTo, respectively. For GEMV workloads across various matrix sizes, both architectures show significant performance improvement, as shown in Figure 10b. Newton and PIMPAL have similar overall execution times, as both spend considerable time on operand and LUT element decoding, incurring tCCD DRAM cycles. In smaller GEMV workloads, Newton outperforms PIMPAL because PIMPAL cannot fully utilize open LUT rows with fewer operands. Fortunately, this performance impact is minimal for end-to-end inference due to the extensive number of FC layers with substantial size in LLMs, resulting in only a 2% performance difference between Newton and PIMPAL. Additionally, both architectures outperform the Jetson platform, achieving a 2.24x speedup for the end-to-end inference tasks. As shown in Figure 10b, both architectures efficiently accelerate GEMV, achieving average speedups of 18.3x and 13.8x over Jetson for GEMV workloads.

C. Area Overhead

Table IV presents the area breakdown of the PIMPAL and Newton architectures, both implemented using the same configuration as in the performance evaluation. PIMPAL includes additional components such as the LUT decoder, adder tree, and minor elements. Storage components are modeled using CACTI 7.0, and processing logic is synthesized with Synopsys Design Compiler [2, 18, 24]. By reducing the number of multipliers in the PU components by 40%, PIMPAL significantly lowers the PU area overhead. Figure 11 compares the performance per area of GEMV workloads relative to Newton. PIMPAL demonstrates higher performance per area overall,

TABLE IV: Area breakdown

Area (mm^2)	PIMPAL	Newton
Adder Tree	1.7936	1.7936
LUT decoder	0.0233	-
Multiplier	-	1.2464
Others	0.005215	0.003215
Total	**1.8222**	**3.0432**

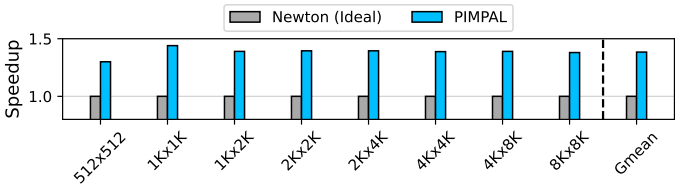


Fig. 11: Performance per area for GEMV workloads with various matrix sizes.

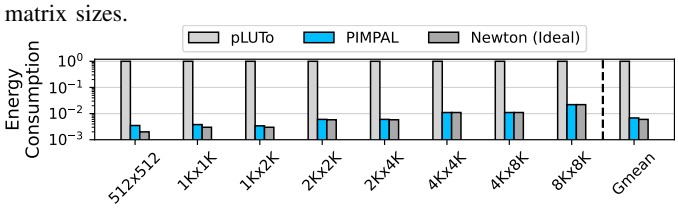


Fig. 12: Energy consumption comparison for GEMV workloads with various matrix sizes.

with a slight decrease in lower-dimension workloads due to the reasons discussed earlier. On average, PIMPAL achieves a 25% improvement in performance-per-area efficiency compared to the representative PU-based PIM architecture.

D. Energy Consumption

Given the importance of energy efficiency on edge devices, we evaluate the energy consumption of each PIM architecture normalized to pLUTo's results. As shown in Figure 12, pLUTo exhibits significantly higher energy consumption compared to the others due to its reliance on frequent row activations, whereas Newton and PIMPAL primarily involve a small number of row activations and column read operations. Although PIMPAL consumes slightly more energy than Newton because of LUT row activations, it mitigates this with the Partial LUT Activation technique, which minimizes row activation energy. As a result, PIMPAL achieves energy efficiency comparable to Newton, with only a 10% difference.

V. Conclusion

This paper presented PIMPAL, a LUT-based PIM architecture for accelerating GEMV operations in small Large Language Models (sLLMs). By introducing LUT Aggregation (LAG) to reduce LUT storage requirements and Locality-aware Compute Mapping (LCM) to minimize DRAM row activations, PIMPAL overcomes key limitations of prior LUT-based PIM designs. Experimental results demonstrate that PIMPAL achieves up to 17.8x higher performance than previous LUT-based designs and delivers 25% better performance-per-area efficiency compared to PU-based PIM architectures, making it a promising solution for high-precision computations in resource-constrained edge environments.

Acknowledgment

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP)-ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (RS-2021-II212052, 25%), the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2023-00228970, 25%/RS-2022-II221170, 50%). Seokin Hong is the corresponding author.

REFERENCES

[1] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 105–117. [Online]. Available: https://doi.org/10.1145/2749469.2750386

[2] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Trans. Archit. Code Optim.*, vol. 14, no. 2, jun 2017. [Online]. Available: https://doi.org/10.1145/3085572

[3] K. K. Chang, P. J. Nair, D. Lee, S. Ghose, M. K. Qureshi, and O. Mutlu, "Low-cost inter-linked subarrays (lisa): Enabling fast inter-subarray data movement in dram," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 568–580.

[4] Z. Chishti and B. Akin, "Memory system characterization of deep learning workloads," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 497–505. [Online]. Available: https://doi.org/10.1145/3357526.3357569

[5] F. Devaux, "The true processing in memory accelerator," in *2019 IEEE Hot Chips 31 Symposium (HCS)*. Los Alamitos, CA, USA: IEEE Computer Society, aug 2019, pp. 1–24. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/HOTCHIPS.2019.8875680

[6] J. D. Ferreira, G. Falcao, J. Gómez-Luna, M. Alser, L. Orosa, M. Sadrosadati, J. S. Kim, G. F. Oliveira, T. Shahroodi, A. Nori, and O. Mutlu, "pluto: Enabling massively parallel computation in dram via lookup tables," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2022, pp. 900–919.

[7] S. R. Group, "Ramulator: A dram simulator," https://github.com/CMU-SAFARI/ramulator, 2015.

[8] H. Ha, A. Pedram, S. Richardson, S. Kvatinsky, and M. Horowitz, "Improving energy efficiency of dram by exploiting half page row access," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[9] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. N. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 372–385.

[10] JEDEC, "Lpddr5," https://www.jedec.org/standards-documents/docs/jesd209-5c, 2023.

[11] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, "A case for exploiting subarray-level parallelism (salp) in dram," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12. USA: IEEE Computer Society, 2012, p. 368–379.

[12] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[13] D. Kwon, S. Lee, K. Kim, S. Oh, J. Park, G.-M. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, V. Kornijcuk, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, G. Kim, B. An, J. Lee, D. Ko, Y. Jun, I. Kim, C. Song, I. Kim, C. Park, S. Kim, C. Jeong, E. Lim, D. Kim, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25v 8gb 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep learning application," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 291–302, 2023.

[14] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, J. Lee, D. Ko, Y. Jun, K. Cho, I. Kim, C. Song, C. Jeong, D. Kwon, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[15] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for pim based on commercial dram technology : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43–56.

[16] Y. Lee, H. Kim, S. Hong, and S. Kim, "Partial row activation for low-power dram system," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2017, pp. 217–228.

[17] Y. Lee, S. Kim, S. Hong, and J. Lee, "Skinflint dram system: Minimizing dram chip writes for low power," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 25–34.

[18] V. S. Naveen Muralimanohar, Ali Shafiee, "Cacti 7.0," https://github.com/HewlettPackard/cacti, 2017.

[19] Nvidia, "Nvidia jetson xavier," https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems/jetson-xavier-series/.

[20] S.-S. Park, K. Kim, J. So, J. Jung, J. Lee, K. Woo, N. Kim, Y. Lee, H. Kim, Y. Kwon, J. Kim, J. Lee, Y. Cho, Y. Tai, J. Cho, H. Song, J. H. Ahn, and N. S. Kim, "An lpddr-based cxl-pnm platform for tco-efficient inference of transformer-based large language models," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 970–982.

[21] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: in-memory accelerator for bulk bitwise operations using commodity dram technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 273–287. [Online]. Available: https://doi.org/10.1145/3123939.3124544

[22] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "Mobilebert: a compact task-agnostic bert for resource-limited devices," 2020. [Online]. Available: https://arxiv.org/abs/2004.02984

[23] P. R. Sutradhar, S. Bavikadi, M. Connolly, S. Prajapati, M. A. Indovina, S. M. P. Dinakarrao, and A. Ganguly, "Look-up-table based processing-in-memory architecture with programmable precision-scaling for deep learning applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 2, pp. 263–275, 2022.

[24] Synopsys, "Synopsys design compiler," https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html.

[25] G. Team, "Gemma 2: Improving open language models at a practical size," 2024. [Online]. Available: https://arxiv.org/abs/2408.00118

[26] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample, "Llama: Open and efficient foundation language models," 2023. [Online]. Available: https://arxiv.org/abs/2302.13971

[27] S. Williams, A. Waterman, and D. Patterson, "Roofline: an insightful visual performance model for multicore architectures," *Commun. ACM*, vol. 52, no. 4, p. 65–76, apr 2009. [Online]. Available: https://doi.org/10.1145/1498765.1498785

[28] W. Xu, Y. Zhang, and X. Tang, "Parallelizing dnn training on gpus: Challenges and opportunities," in *Companion Proceedings of the Web Conference 2021*, ser. WWW '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 174–178. [Online]. Available: https://doi.org/10.1145/3442442.3452055

[29] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "Opt: Open pre-trained transformer language models," 2022.

[30] R. Zhou, A. Roohi, D. Misra, and S. Angizi, "Red-lut: Reconfigurable in-dram luts enabling massive parallel computation," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, ser. ICCAD '22. New York, NY, USA: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3508352.3549469

SpecASR: Accelerating LLM-based Automatic Speech Recognition via Speculative Decoding

Linye Wei[1,2], Shuzhang Zhong[1,2], Songqiang Xu[1,3], Runsheng Wang[2,4,5], Ru Huang[2,4,5], Meng Li[1,2,5*]
[1]Institute for Artificial Intelligence, Peking University, Beijing, China
[2]School of Integrated Circuits, Peking University, Beijing, China
[3]School of Software and Microelectronics, Peking University, Beijing, China
[4]Institute of Electronic Design Automation, Peking University, Wuxi, China
[5]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China
meng.li@pku.edu.cn

***Abstract*—Large language model (LLM)-based automatic speech recognition (ASR) has recently attracted a lot of attention due to its high recognition accuracy and enhanced multi-dialect support. However, the high decoding latency of LLMs challenges the real-time ASR requirements. Although speculative decoding has been explored for better decoding efficiency, they usually ignore the key characteristics of the ASR task and achieve limited speedup. To further reduce the real-time ASR latency, in this paper, we propose a novel speculative decoding framework specialized for ASR, dubbed SpecASR. SpecASR is developed based on our core observation that ASR decoding is audio-conditioned, which results in high output alignment between small and large ASR models, even given output mismatches in intermediate decoding steps. Therefore, SpecASR features an adaptive draft sequence generation process that dynamically modifies the draft sequence length to maximize the token acceptance length. SpecASR further proposes a draft sequence recycling strategy that reuses the previously generated draft sequence to reduce the draft ASR model latency. Moreover, a two-pass sparse token tree generation algorithm is also proposed to balance the latency of draft and target ASR models. With extensive experimental results, we demonstrate SpecASR achieves 3.04×–3.79× and 1.25×–1.84× speedup over the baseline autoregressive decoding and speculative decoding, respectively, without any loss in recognition accuracy.**

I. Introduction

Large language models (LLMs) like GPT [1] and Llama [2] have demonstrated powerful capabilities across a wide range of applications such as text generation, question answering [3], and embodied AI [4]. Recently, LLM has also been applied to automatic speech recognition (ASR) [5]–[10]. By incorporating an audio encoder for speech embedding extraction, LLM-based ASR models have demonstrated impressive recognition accuracy and enhanced support for a broader range of languages, dialects, and accents [7].

However, the improvement of LLM-based ASR is achieved at the cost of much longer decoding latency as LLM significantly increases the model size and compute of the ASR models. For example, the LLMs in BESTOW [10], Speech-Llama [9], and Seed-ASR [7] have 1.1B, 7B, and >10B parameters, respectively. In contrast, the model size of an audio encoder is generally under 1B, and even below 100M. The autoregressive decoding nature of LLM further exacerbates the computation latency. As a result, as shown in Fig. 1, the LLM decoder incurs much larger inference latency compared to the audio encoder, becoming the major efficiency bottleneck of ASR.

To improve the LLM decoding efficiency, speculative decoding has been extensively studied in the context of natural language processing tasks [11]–[16] and has also recently been introduced to ASR models [17]. The core idea of speculative decoding is first to employ a small ASR model to generate draft output tokens autoregressively. Then, the target large ASR model is used to verify the draft sequences altogether. The draft tokens that match the outputs of the target ASR model are accepted, which improves the efficiency of the target ASR model without sacrificing recognition quality. Though effective, existing works often directly apply speculative decoding to ASR but ignore its key characteristics [17], [18]. Hence, they only achieve limited speedup, e.g., $\sim 1.5\times$, which may still be insufficient to meet the tight real-time constraints.

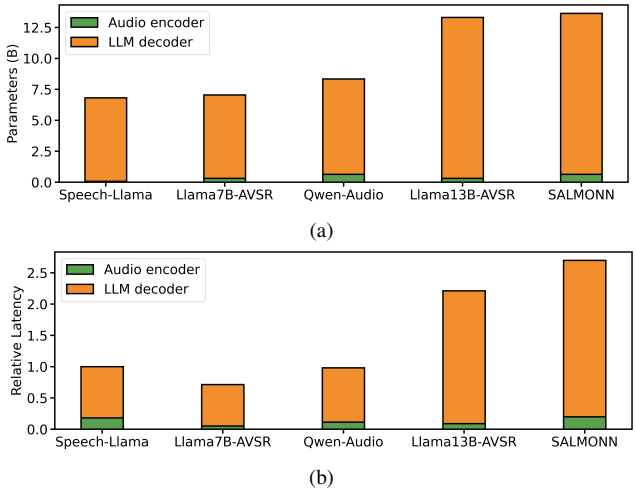


Fig. 1: (a) Parameter ratio and (b) relative latency of audio encoders and LLM decoders in LLM-based ASR models.

In this paper, we observe that ASR is an audio-conditioned generation task, which results in two key characteristics: on one hand, unlike many text tasks, the outputs of large and small ASR models are often highly aligned; on the other hand, even given output mismatches between the large and small models, the downstream decoding of two models can still be aligned due to the audio input. Based on these observations, we propose a highly efficient speculative decoding framework, dubbed SpecASR, specialized for LLM-based ASR. SpecASR features 3 key techniques for efficient ASR, including an adaptive single-sequence prediction algorithm to maximize the decoding-acceptance ratio in each verification step, a draft sequence recycling strategy to reuse the unaccepted draft sequence and reduce

This work was supported in part by NSFC under Grant 62495102 and Grant 92464104, in part by the National Key Research and Development Program under Grant 2024YFB4505004, in part by Beijing Municipal Science and Technology Program under Grant Z241100004224015, and in part by 111 Project under Grant B18001.

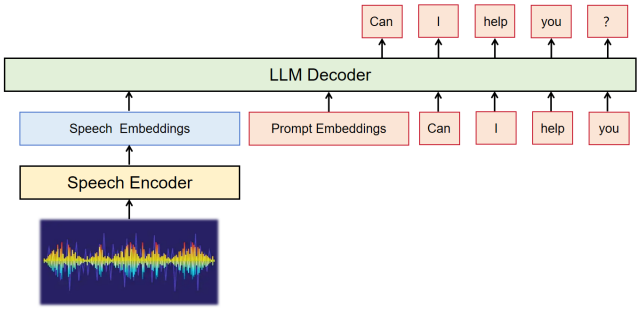


Fig. 2: Cascaded architecture of LLM-based ASR models.

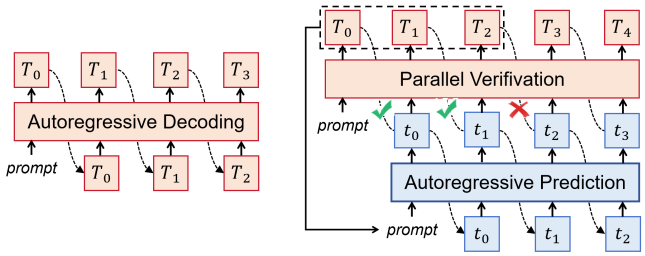


Fig. 3: Autoregressive (left) and speculative (right) decoding.

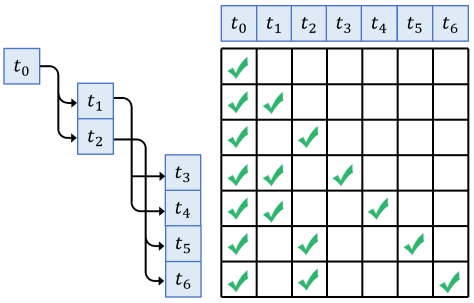


Fig. 4: 2D attention mask for draft token tree.

draft generation latency, and a two-pass sparse token tree algorithm to balance the latency of draft generation and target verification. Our main contributions can be summarized as follows:

- We observe that ASR decoding is audio conditioned and propose SpecASR to improve the ASR decoding efficiency.
- We propose 3 novel speculative decoding techniques, including adaptive single-sequence prediction, draft sequence recycling, and two-pass sparse token tree to leverage ASR characteristics.
- With extensive experimental results, SpecASR achieves 3.04×–3.79× and 1.25×–1.84× latency reduction compared to the baseline autoregressive and speculative decoding algorithms, respectively, with iso-accuracy.

II. Background

A. LLM-based ASR Models

The integration of LLMs into ASR tasks has demonstrated significant advantages in recognition accuracy and contextual awareness. LLM-based ASR models utilize a cascaded architecture consisting of an audio encoder and an LLM decoder, as illustrated in Fig. 2. The audio encoder is responsible for feature extraction and dimensional transformation of the speech input, while the LLM decoder generates text transcriptions through autoregressive decoding.

a) Audio Encoder: In LLM-based ASR models, audio encoders serve as bridges connecting audio signals to LLMs originally designed for text tasks. The processing is conducted in two stages. The first stage encompasses feature extraction and compression of speech frames by the audio encoder as the number of speech frames significantly exceeds the text tokens needed to convey the same semantics. In the following stage, the extracted features are subjected to stacking, enabling projection into the hidden dimensions of LLMs and allowing for prefilling alongside text prompts. Recent studies leverage established ASR architectures such as Conformer [19] or Whisper [20] as audio encoders. As illustrated in Fig. 1, audio encoders typically account for a small fraction of the model.

b) LLM Decoder: The LLM decoder processes audial embeddings alongside text prompts to perform autoregressive decoding of speech transcriptions. Prior studies have shown that text LLMs, such as GPT [1], Llama [2] and Qwen [21], can be effectively utilized in ASR tasks by employing either pre-trained models directly or fine-tuning them with a limited number of parameters via LoRA [22]. However, the complex architecture of LLMs and the autoregressive decoding process result in a long latency, making the decoder a major bottleneck and hence, the main focus of our work.

B. Speculative Decoding

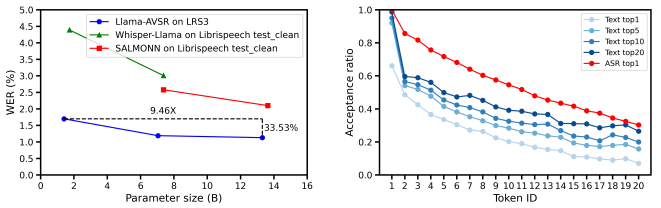
Speculative decoding [11]–[15], [23]–[26] incorporates lightweight draft models to enhance the decoding speed of target models through the "Draft-then-Verify" paradigm. As shown in Fig. 3, the draft model first efficiently generates predictions for multiple forthcoming tokens autoregressively. Then, the target model concurrently verifies all tokens within the draft sequence, terminating at the first token that fails verification and reverting to the draft model for the next round of predictions. Hence, speculative decoding guarantees lossless acceleration of target models' decoding process.

Tree-structure speculative decoding like SpecInfer [13] creatively extends a single draft sequence into a fixed-shape token tree that encompasses multiple candidate draft sequences. As shown in Fig. 4, this method maintains multiple candidate tokens at each decoding step of the draft model, arranging them into a tree-like structure. During verification, the candidate sequences are expanded and merged to facilitate parallel processing. Leveraging a 2D attention mask, the target model effectively retrieves independent verification outcomes for each branch. By promoting greater diversity in predictions at each decoding step, tree-structure speculative decoding achieves a higher success rate during the target model's verification phase, thus accelerating the speculative process. Recent works, such as Medusa [14] and ProPD [23], have further refined the construction of draft token trees by selecting token branches based on the probability distribution. This dynamic tree-structure speculative decoding increases the verification acceptance rate and improves flexibility when applied to various models and tasks.

However, the abandonment of draft sequences that fail verification in speculative decoding results in significant wastage of draft predictions. Furthermore, existing tree-structure speculative approaches primarily focus on enhancing the diversity of draft model decoding, limiting tree depths to prevent the prediction overhead from growing exponentially with the number of forward passes. In the context of ASR tasks, the high alignment between the decoding outputs of the draft model and the target model suggests that draft tokens have the potential for multiple reuses. Also, a token tree that emphasizes both constrained width expansion and proactive depth expansion may yield improved performance. Tab. I presents a comparison between our method and existing speculative decoding approaches.

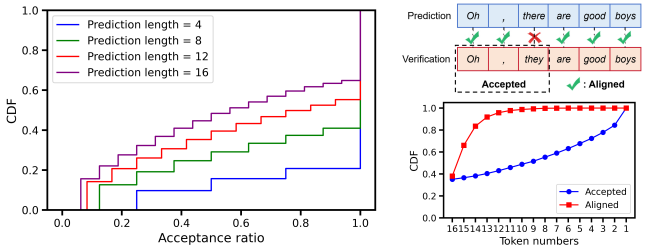
TABLE I: Comparison between our methods with representative speculative decoding methods.

Method	Draft Generation Efficiency	Target Verification Efficiency	Draft Sequence Length	Target Accept Rate	Flexibility
Single Sequence [11], [12]	high	low	medium	low	medium
Fixed Tree [13], [15], [26]	low	high	low	medium	low
Dynamic Tree [14], [23]–[25]	low	high	low	high	high
Ours	high	high	high	high	high



(a) (b)

Fig. 5: (a) WERs of ASR models with multiple scales, (b) comparison of speculative acceptance with top-k logits in ASR/text tasks.



(a) (b)

Fig. 6: (a) Acceptance ratio distribution with different prediction lengths, (b) high alignment of draft and target models in ASR tasks.

III. Motivation

In this section, we discuss the feasibility of applying speculative decoding to ASR tasks, examining the unique characteristics of the speculative decoding process within ASR models and the potential for efficiency improvements. Based on these analyses, we present three key observations that will motivate our subsequent work.

Observation 1: Applying speculative decoding to ASR tasks is effective and is expected to yield higher speedups. As illustrated in Fig. 5a, we evaluate the performance of ASR models with varying parameter sizes across multiple datasets [27], [28]. Using word error rate (WER) as the primary evaluation metric, larger models exhibit a 20%-33% reduction in WER compared to smaller models, albeit with a significant increase in model size. While the performance of smaller models falls short of meeting user expectations in practical ASR applications, they show considerable potential within the context of speculative decoding, achieving WERs as low as 10% or less.

However, speculative decoding for ASR tasks remains an area of limited exploration. Existing approaches typically apply single-sequence prediction or tree-structured speculative decoding techniques, originally developed for text tasks, to ASR models, resulting in modest speedups of only 1.5×–2×. As shown in Fig. 5b, our findings demonstrate that draft models exhibit significantly improved performance in ASR tasks compared to text-based applications. By capitalizing on the strong consistency between the draft and target models, it is possible to employ longer prediction lengths and constrained token tree expansion, thereby improving the verification efficiency of the target model.

Observation 2: In ASR tasks, the acceptance rate of the draft model in speculative decoding exhibits substantial variation, with some draft sequences showing low acceptance rates yet strong alignment with verification sequences. We analyze the distribution of acceptance rates under varying prediction lengths in speculative decoding, as shown in Fig. 6a. A substantial proportion of predictions are fully accepted, highlighting the effectiveness of long-sequence draft decoding. However, the remaining acceptance rates are primarily concentrated at lower values, which can be attributed to variations in pronunciation and acoustic quality across specific speech segments. These factors lead to a localized error distribution, contributing to inefficiencies in draft predictions.

To address this issue, we are motivated to employ dynamic prediction lengths. Our observations reveal that, due to the role of the audio encoder in ASR tasks, which ensures the basic semantic coherence of the transcription process, ASR decoding is audio-conditioned. As shown in Fig. 6b, unaccepted draft sequences demonstrate a high degree of alignment with the verification sequence. This suggests that the computational overhead of draft models can be significantly reduced by reusing unaccepted draft sequences.

Observation 3: In speculative decoding, the relative contributions of the draft and target models to overall latency vary significantly across configurations, such as prediction length and model size. Current tree-structured speculative approaches are based on the assumption that the verification process of the target model is the primary bottleneck in achieving decoding speedups with constrained prediction lengths. As a result, promoting diversity in the draft model's predictions to enhance verification acceptance rates has been shown to effectively accelerate decoding. However, ASR decoding benefits from longer prediction lengths to improve overall efficiency, highlighting the need for a thorough analysis of the computational load distribution between the draft and target models.

As shown in Fig. 7, we evaluate the proportion of latency contributed by draft prediction and target verification across varying prediction lengths and model configurations. As the prediction length increases, the draft model progressively becomes the dominant source of latency. In contrast, for a fixed prediction length, a significant size disparity between the draft and target models often results in the target model being the primary one. This suggests that, depending on specific task requirements, latency bottlenecks may arise from either model. Our proposed approach must effectively balance the trade-off between the draft and target models to achieve optimal speedups.

IV. SpecASR Framework

Building on these analysis, we propose SpecASR, a novel framework designed to accelerate the decoding process of ASR models, as illustrated in Fig. 8. SpecASR incorporates adaptive single-sequence

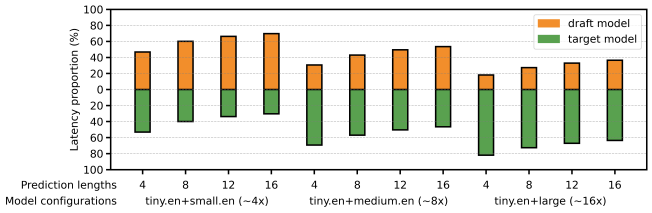


Fig. 7: Decoding latency proportion on LibriSpeech clean-test.

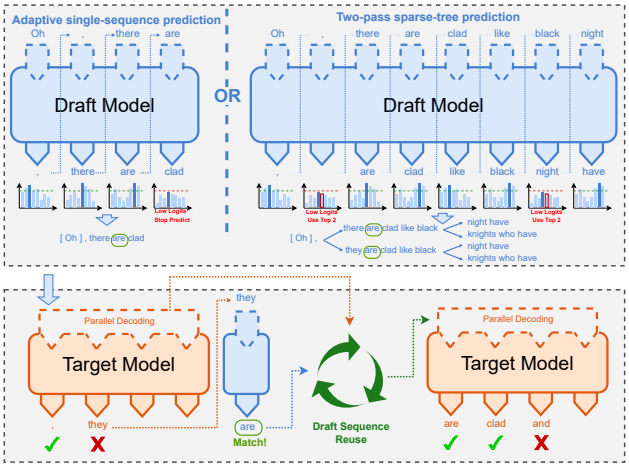


Fig. 8: The overview framework of SpecASR.

prediction, dynamically adjusting the length of draft sequences in each prediction round to enhance verification efficiency. To mitigate the computational burden on the draft model, SpecASR incorporates a draft sequence recycling strategy. This approach leverages the opportunity to merge unaccepted tokens during decoding, dynamically adjusting the length of draft sequences in each regeneration phase. Furthermore, we introduce a two-pass sparse-tree prediction method as an alternative decoding strategy to accommodate diverse target model configurations in ASR tasks. This approach effectively balances latency between the draft and target models, enabling optimal speedup, particularly for larger-scale target models.

A. Adaptive Single-sequence Prediction

The strong alignment observed between the decoding processes of large and small models in ASR tasks motivates the adoption of longer prediction lengths to reduce the number of verification rounds. Unlike prior studies, which generally employ limited prediction lengths [29], [30], we extend the maximum prediction length to 24 tokens. This extension fully exploits the draft model's competitive decoding capabilities, enhancing overall efficiency. Before the draft sequence reaches the predefined length, we proactively identify positions with a high probability of verification failure during the prediction process. At these positions, the draft model's decoding is truncated, and the generated tokens are promptly passed to the target model for verification. An analysis of the logits distribution for accepted and rejected tokens during verification reveals a strong positive correlation between acceptance probability and normalized logits. Leveraging this insight, we introduce a threshold-based mechanism to truncate predictions with logits falling below a specified value, enabling early verification of the draft sequence. This adaptive approach to prediction length facilitates extended prediction and acceptance lengths while preserving the efficiency of the draft model.

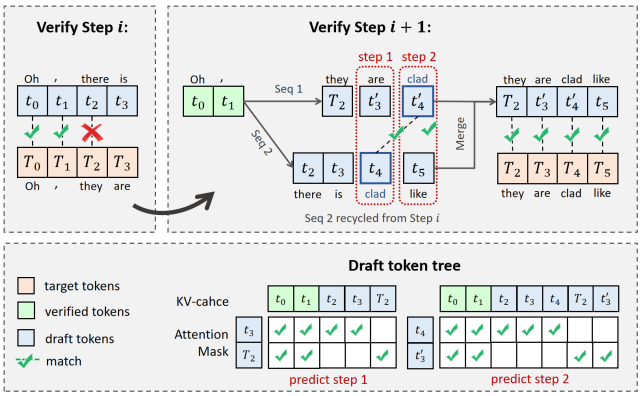


Fig. 9: Draft sequence recycling strategy.

B. Draft Sequence Recycling Strategy

Given that the logits of unaccepted draft tokens span a wide range of values, even outputs with normalized logits exceeding the set threshold may fail verification, leading to the rejection of a substantial number of tokens. Building on the insights presented in Section III, we propose a strategy to recycle these draft tokens. As illustrated in Fig. 9, unlike previous approaches that construct draft token trees to improve verification acceptance rates [31]–[33], we employ a masked token tree to enable parallel prediction and token reuse within the draft model. Our recycling strategy is composed of two key stages: draft regeneration and merging.

In the first stage, the complete draft sequence submitted for verification is retained as sequence 1, while the partially verified tokens are stored as sequence 2 to construct a draft token tree. By applying an attention mask to the unaccepted tokens, the draft model performs parallel decoding, simultaneously extending the draft sequence and regenerating tokens that failed verification. This approach effectively conceals the regeneration delay within the ongoing predictions of the draft model. In the second stage, the positions of all tokens are tracked and marked. During each step of the regeneration process, the tokens generated in sequence 2 are compared with those in sequence 1 at corresponding or adjacent positions. Upon identifying a match, the two branches of the draft token tree are merged, enabling the reuse of previously decoded content and alleviating the prediction workload on the draft model.

C. Two-pass Sparse-tree Prediction

The preceding discussion builds on Observation 3 in Section III, which highlights that the draft model dominates the latency in speculative decoding with extended draft prediction lengths. However, this observation also notes that as the parameter size of the target model increases, which is exactly the trend of LLM decoder in ASR tasks, the verification overhead of the target model is anticipated to become a significant bottleneck. While adaptive single-sequence prediction enhances verification efficiency by adjusting prediction lengths dynamically, further acceleration is possible, particularly in configurations characterized by a significant size disparity between the draft and target models.

Tree-structured speculative decoding enhances verification acceptance rates by constructing a token tree within the draft model, thereby facilitating diverse predictions. However, its effectiveness is constrained by limitations on prediction length. To address this challenge, we propose a sparse token tree with limited-width expansions. As illustrated in Fig. 10, the process begins with single-

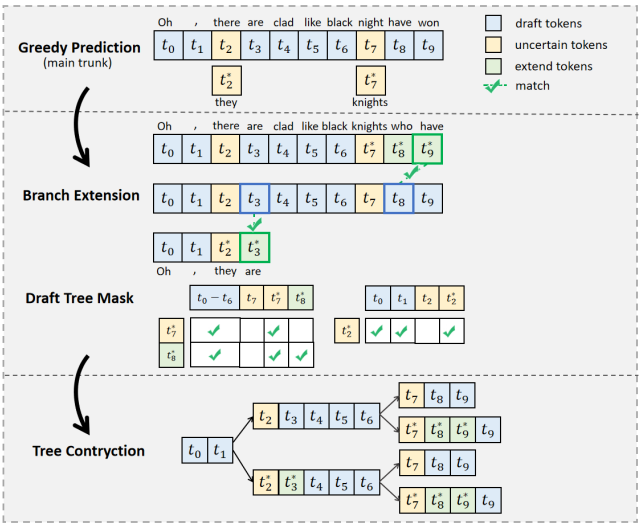


Fig. 10: Two-pass sparse-tree prediction mechanism.

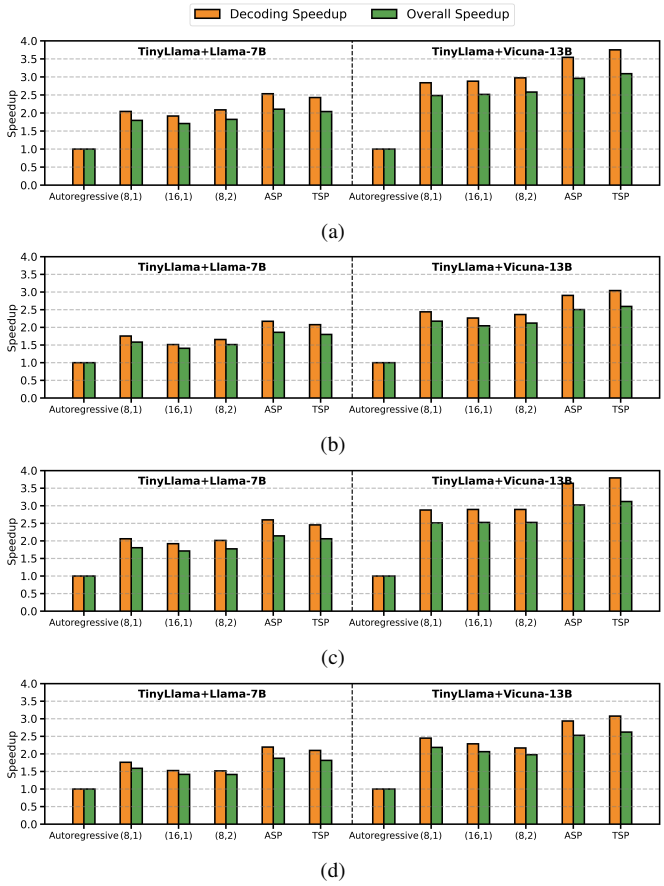


Fig. 11: Speedup comparison with baseline methods, including autoregressive decoding, speculative decoding with 8/16 prediction length and 1/2 beams (denoted as (8, 1), (16, 1), (8, 2) in the figure) on datasets: (a) test-clean, (b) test-other, (c) dev-clean, (d) dev-other.

sequence greedy decoding, followed by the identification of potential verification failures based on output logits. Rather than truncating generation upon detecting uncertain tokens in single-sequence prediction, we mark their positions and tokens with top-k highest probability, encouraging the draft model to continue generating tokens. Upon completion of the single-sequence generation, which serves as the "main trunk" of the token tree, we perform multi-branch exploration exclusively at the points of uncertainty in the sequence. This exploration extends forward based on the candidate tokens which identified earlier, constituting the second draft decoding pass. To minimize the computational cost of this exploration, we employ the draft sequence recycling strategy. Specifically, when a branch can be concatenated to the "main trunk" or any previously generated branches, we refrain from further extending it and instead generate new branches by appending them to the "main trunk" or the existing branches. Our experiments demonstrate that extending the draft token tree by tokens with the second highest probability yields effective results, as discussed in the following section.

In the subsequent verification phase, we employ the attention mask from SpecInfer [13] to parallelize the verification of candidate sequences. For segments that fail verification, we apply the recycling strategy to regenerate them and mark the points of uncertainty to construct the draft token tree in the next iteration.

V. Experimental Evaluation

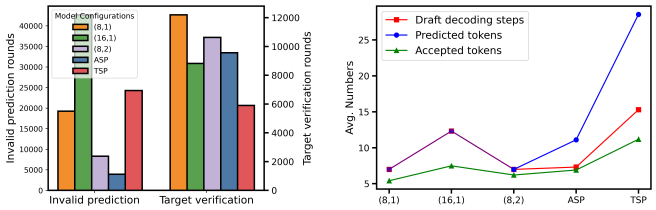
A. Experimental Setup

The proposed SpecASR is deployed on NVIDIA RTX A6000, with its performance evaluated on LibriSpeech dataset [27]. Recent studies on LLM-based ASR models often employ Llama or Vicuna [34] as LLM decoders. However, ASR models based on these architectures are not yet publicly available. As an alternative, we utilize the open-source Whisper models, selecting tiny.en and medium.en versions as draft and target models, respectively, and record the decoding trajectories of Whisper-based SpecASR. Subsequently, we simulate these trajectories using TinyLlama as the draft model and Llama-7B/Vicuna-13B as target models. We observe that the word error rate (WER) gap between TinyLlama and Llama-7B/Vicuna-13B, as reported by [35], is smaller than that between Whisper tiny.en and medium.en on the LibriSpeech dataset. This suggests that SpecASR is likely to achieve better alignment between draft and target models when deployed with Llama/Vicuna models, thereby enabling superior acceleration performance compared to our experimental results.

B. Main Results

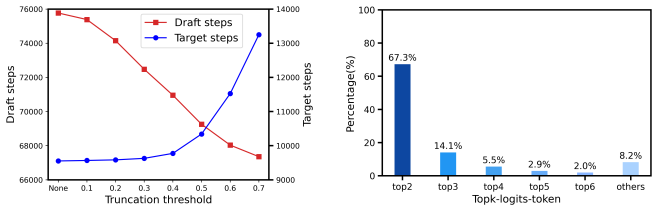
We evaluate SpecASR with adaptive single-sequence prediction (ASP) and two-pass sparse-tree prediction (TSP) on LibriSpeech test-clean, test-other, dev-clean, and dev-other datasets, as shown in Fig. 11. For comparison, we include baseline models employing autoregressive and speculative decoding, configured with (prediction length, beam size) pairs of (8, 1), (16, 1), and (8, 2). A normalized logits threshold of 0.4 is applied to identify uncertain predicted tokens. For two-pass sparse-tree prediction, we choose to extend the draft token tree by incorporating the token with the second highest probability. Detailed discussions are provided later in this section.

a) Speedups across different model configurations and datasets: Using Llama-7B as the simulated target model, SpecASR achieves speedups of 2.08×–2.60× compared to autoregressive decoding and 1.21×–1.45× relative to the baseline speculative decoding. For Vicuna-13B, the speedups increase to 3.04×–3.79× over autoregressive decoding and 1.25×–1.84× over the baseline speculative decoding methods. These results underscore the versatility of the single-sequence and sparse-tree decoding strategies proposed in SpecASR, demonstrating their ability to effectively adapt to various model configurations. As illustrated in Fig. 12, adaptive single-sequence



(a) (b)

Fig. 12: Comparison of speculative methods on test-clean: (a) the number of rounds for draft prediction and target verification, (b) the average number of draft decoding steps, predicted tokens per round, and accepted tokens per round.



(a) (b)

Fig. 13: (a) Step changes for draft prediction and target verification with different truncation thresholds in single-sequence speculative decoding, (b) the percentage of accepted tokens with different ranks in the draft models' output logits except for top1.

prediction reduces 74.1% of ineffective prediction steps compared to baseline speculative decoding, through early truncation and draft reuse. This results in a 94.4% decoding-acceptance ratio per round. This approach alleviates the load on the draft model, demonstrating improved performance when the parameter disparity between the draft and target models is relatively small, as seen with TinyLlama and Llama-7B. In contrast, two-pass sparse-tree prediction enhances the diversity of the draft model's generation, leading to a slight decrease in the decoding-acceptance ratio. However, the average accepted length per verification round increases by 106.6%, resulting in a more substantial reduction in latency, particularly when the target model is significantly larger, as in the case of Vicuna-13B.

b) Speedup on different datasets: Our evaluation included noisy datasets, such as test-other and dev-other, to examine the impact of decoding accuracy on speculative efficiency. Using Vicuna-13B as the target model, SpecASR achieves a 3.04×–3.07× speedup over autoregressive decoding on these noisy datasets. Compared to the clean datasets, there is 19% performance degradation. This is because the draft model exhibits a significant decrease in recognition accuracy compared to the target model when processing these challenging datasets. Consequently, the number of accepted tokens per round decreases, resulting in a lower prediction-acceptance ratio and a subsequent decline in performance. This also highlights the effectiveness of using the Whisper model to simulate Llama-based LLMs, as TinyLlama exhibits a smaller word error rate gap compared to Llama-7B and Vicuna-13B. We anticipate that SpecASR will achieve even better speedup when deployed with Llama-based ASR models.

C. Impact of Key Parameters

a) Truncation threshold: Fig. 13a illustrates the step count for both the draft and target models during single-sequence prediction with varying truncation thresholds. When the threshold is low, there are few tokens with logits below this threshold, so that the numbers of draft generation and target verification steps remains almost unchanged. As the threshold increases, the number of the draft model's generation steps decreases, while the number of the target model's verification steps increases, as some correct predictions with low logits are truncated. Further increasing the threshold leads to the truncation of additional correct predictions, causing a sharp rise in the number of target model verifications and reducing the benefits of truncation. Our experiments indicate that a threshold of 0.4 is optimal, although this may vary depending on the model.

b) Tokens with top-k highest probability: As discussed in Section IV, expanding tokens with the second highest probability yields optimal results. As shown in Fig. 13b, we analyzed the rank of the token in draft model's output that corresponds to target model's actual decoding when the top-1 token predicted fails verification. Our analysis revealed that over two-thirds of these tokens are ranked as the second-highest choice in the draft model's predictions. Given the substantial overhead of extending more branches, expanding the tree with second highest probability is the most effective solution.

TABLE II: Ablation study on the average decoding latency of every 10s audio on the LibriSpeech test-clean dataset.

Methods	Draft(ms)	Target(ms)	Total(ms)
baseline speculative	231.06	254.48	485.54
+adaptive single-sequence prediction	236.23	191.20	427.43
+draft sequence recycling	189.48	199.52	389.00
+two-pass sparse-tree prediction	244.62	123.17	367.79

D. Ablation Study

To further evaluate the effectiveness of SpecASR in accelerating ASR decoding, we progressively integrate its core techniques into the baseline speculative decoding under the Whisper tiny.en+medium.en model configurations. We then evaluate their contributions to reducing the latency of the draft model, target model, and overall system, as summarized in Tab. II. The results indicate that adaptive single-sequence prediction notably enhances the efficiency of target verification with negligible additional cost to draft model computation. By implementing the draft sequence recycling strategy, the reuse of unaccepted tokens significantly reduces draft model latency. Furthermore, due to the substantial discrepancy between the draft and target models, the two-pass sparse-tree prediction approach reduces target verification latency by over 50% compared to the baseline speculative decoding method, while introducing only a slight increase in draft model latency, ultimately achieving the greatest speedup.

VI. Conclusion

In this paper, we propose SpecASR, a method designed to accelerate ASR models, particularly LLM-based ASR systems. We analyze the effectiveness of speculative decoding in ASR tasks and identify opportunities for improvement in traditional speculative decoding. Building on these observations, we introduce adaptive single-sequence prediction and two-pass sparse-tree prediction to adapt to different model configurations, maximizing speculative decoding efficiency through the recycling of draft sequences. Compared to baseline autoregressive decoding and speculative decoding, our approach achieves 3.04×–3.79× and 1.25×–1.84× speedup without compromising the recognition accuracy.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] Y. Zhuang, Y. Yu, K. Wang, H. Sun, and C. Zhang, "Toolqa: A dataset for llm question answering with external tools," *Advances in Neural Information Processing Systems*, vol. 36, pp. 50 117–50 143, 2023.

[4] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.

[5] J. Wu, Y. Gaur, Z. Chen, L. Zhou, Y. Zhu, T. Wang, J. Li, S. Liu, B. Ren, L. Liu *et al.*, "On decoder-only architecture for speech-to-text and large language model integration," in *2023 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2023, pp. 1–8.

[6] F. Seide, M. Doulaty, Y. Shi, Y. Gaur, J. Jia, and C. Wu, "Speech reallm–real-time streaming speech recognition with multimodal llms by teaching the flow of time," *arXiv preprint arXiv:2406.09569*, 2024.

[7] Y. Bai, J. Chen, J. Chen, W. Chen, Z. Chen, C. Ding, L. Dong, Q. Dong, Y. Du, K. Gao *et al.*, "Seed-asr: Understanding diverse speech and contexts with llm-based speech recognition," *arXiv preprint arXiv:2407.04675*, 2024.

[8] Q. Fang, S. Guo, Y. Zhou, Z. Ma, S. Zhang, and Y. Feng, "Llama-omni: Seamless speech interaction with large language models," *arXiv preprint arXiv:2409.06666*, 2024.

[9] Y. Fathullah, C. Wu, E. Lakomkin, J. Jia, Y. Shangguan, K. Li, J. Guo, W. Xiong, J. Mahadeokar, O. Kalinli *et al.*, "Prompting large language models with speech recognition abilities," in *ICASSP 2024-2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2024, pp. 13 351–13 355.

[10] Z. Chen, H. Huang, O. Hrinchuk, K. C. Puvvada, N. R. Koluguri, P. Żelasko, J. Balam, and B. Ginsburg, "Bestow: Efficient and streamable speech language model with the best of two worlds in gpt and t5," *arXiv preprint arXiv:2406.19954*, 2024.

[11] C. Chen, S. Borgeaud, G. Irving, J.-B. Lespiau, L. Sifre, and J. Jumper, "Accelerating large language model decoding with speculative sampling," *arXiv preprint arXiv:2302.01318*, 2023.

[12] Y. Leviathan, M. Kalman, and Y. Matias, "Fast inference from transformers via speculative decoding," in *International Conference on Machine Learning*. PMLR, 2023, pp. 19 274–19 286.

[13] X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Zhang, R. Y. Y. Wong, A. Zhu, L. Yang, X. Shi *et al.*, "Specinfer: Accelerating generative large language model serving with tree-based speculative inference and verification," *arXiv preprint arXiv:2305.09781*, 2023.

[14] T. Cai, Y. Li, Z. Geng, H. Peng, J. D. Lee, D. Chen, and T. Dao, "Medusa: Simple llm inference acceleration framework with multiple decoding heads," *arXiv preprint arXiv:2401.10774*, 2024.

[15] Y. Li, F. Wei, C. Zhang, and H. Zhang, "Eagle: Speculative sampling requires rethinking feature uncertainty," *arXiv preprint arXiv:2401.15077*, 2024.

[16] Y. Xiong, R. Zhang, Y. Li, T. Wu, and L. Zou, "Dyspec: Faster speculative decoding with dynamic token tree structure," *arXiv preprint arXiv:2410.11744*, 2024.

[17] Y. Segal-Feldman, A. Shamsian, A. Navon, G. Hetz, and J. Keshet, "Whisper in medusa's ear: Multi-head efficient decoding for transformer-based asr," *arXiv preprint arXiv:2409.15869*, 2024.

[18] S. Gandhi, P. von Platen, and A. M. Rush, "Distil-whisper: Robust knowledge distillation via large-scale pseudo labelling," *arXiv preprint arXiv:2311.00430*, 2023.

[19] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *arXiv preprint arXiv:2005.08100*, 2020.

[20] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International conference on machine learning*. PMLR, 2023, pp. 28 492–28 518.

[21] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," *arXiv preprint arXiv:2309.16609*, 2023.

[22] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[23] S. Zhong, Z. Yang, M. Li, R. Gong, R. Wang, and R. Huang, "Propd: Dynamic token tree pruning and generation for llm parallel decoding," *arXiv preprint arXiv:2402.13485*, 2024.

[24] Y. Li, F. Wei, C. Zhang, and H. Zhang, "Eagle-2: Faster inference of language models with dynamic draft trees," *arXiv preprint arXiv:2406.16858*, 2024.

[25] Z. Chen, A. May, R. Svirschevski, Y. Huang, M. Ryabinin, Z. Jia, and B. Chen, "Sequoia: Scalable, robust, and hardware-aware speculative decoding," *arXiv preprint arXiv:2402.12374*, 2024.

[26] S. Yang, S. Huang, X. Dai, and J. Chen, "Multi-candidate speculative decoding," *arXiv preprint arXiv:2401.06706*, 2024.

[27] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[28] T. Afouras, J. S. Chung, and A. Zisserman, "Lrs3-ted: a large-scale dataset for visual speech recognition," *arXiv preprint arXiv:1809.00496*, 2018.

[29] Z. Xiao, H. Zhang, T. Ge, S. Ouyang, V. Ordonez, and D. Yu, "Parallelspec: Parallel drafter for efficient speculative decoding," *arXiv preprint arXiv:2410.05589*, 2024.

[30] X. Liu, L. Hu, P. Bailis, A. Cheung, Z. Deng, I. Stoica, and H. Zhang, "Online speculative decoding," *arXiv preprint arXiv:2310.07177*, 2023.

[31] Z. Sun, A. T. Suresh, J. H. Ro, A. Beirami, H. Jain, and F. Yu, "Spectr: Fast speculative decoding via optimal transport," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[32] W. Jeon, M. Gagrani, R. Goel, J. Park, M. Lee, and C. Lott, "Recursive speculative decoding: Accelerating llm inference via sampling without replacement," *arXiv preprint arXiv:2402.14160*, 2024.

[33] Y. Cheng, A. Zhang, X. Zhang, C. Wang, and Y. Wang, "Recurrent drafter for fast speculative decoding in large language models," *arXiv preprint arXiv:2403.09919*, 2024.

[34] W.-L. Chiang, Z. Li, Z. Lin, Y. Sheng, Z. Wu, H. Zhang, L. Zheng, S. Zhuang, Y. Zhuang, J. E. Gonzalez *et al.*, "Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, march 2023," *URL https://lmsys. org/blog/2023-03-30-vicuna*, vol. 3, no. 5, 2023.

[35] Z. Ma, G. Yang, Y. Yang, Z. Gao, J. Wang, Z. Du, F. Yu, Q. Chen, S. Zheng, S. Zhang *et al.*, "An embarrassingly simple approach for llm with strong asr capacity," *arXiv preprint arXiv:2402.08846*, 2024.

BiNeuroRAM: Energy-Efficient ReRAM-Based PIM for Accurate Bipolar Spiking Neural Network Acceleration

Jun Yan Lee[⋆⋊], Chen Nie[⋆⋊], Kang You[⋆], Yueyang Jia[†], Rui Yang[⋆†◇], Zhezhi He[⋆♠]
[⋆] Intelligent Computing Research Group, SEIEE, Shanghai Jiao Tong University, Shanghai, China
[†]University of Michigan – Shanghai Jiao Tong University Joint Institute, Shanghai, China
[◇]State Key Laboratory of Radio Frequency Heterogeneous Integration, Shanghai Jiao Tong University, Shanghai, China
{junyan.lee, chen.nie, kang_you, 975842069, rui.yang, zhezhi.he}@sjtu.edu.cn
[⋊]Equal contribution [♠]Corresponding author

***Abstract*—ReRAM is a promising non-volatile memory for neuromorphic accelerators, yet it faces challenges such as high sensing power and accuracy degradation. This work proposes BiNeuroRAM, a novel spiking neural network (SNN) accelerator leveraging ReRAM-based processing-in-memory (PIM), with three key contributions: (1) It is the first to support higher-accuracy spike-tracing bipolar-integrate-and-fire (ST-BIF) neurons, achieving 80.9% accuracy on ImageNet, 8.4% higher than the previous state-of-the-art; (2) It introduces a low-power voltage sense amplifier (LPVSA) that reduces ReRAM read power by 14.7∼58.2×, enhancing energy efficiency; (3) It employs an asynchronous micro-architecture that fully exploits the event-driven nature of SNNs. Experimental results show that BiNeuroRAM improves throughput density and energy efficiency by 2.08× and 2.09× on ImageNet with ResNet-18, compared to traditional integrate-and-fire (IF) neuron-based SNN accelerators.**

***Index Terms*—Neuromorphic hardware, spiking neural network, processing-in-memory, asynchronous circuit, bipolar neuron**

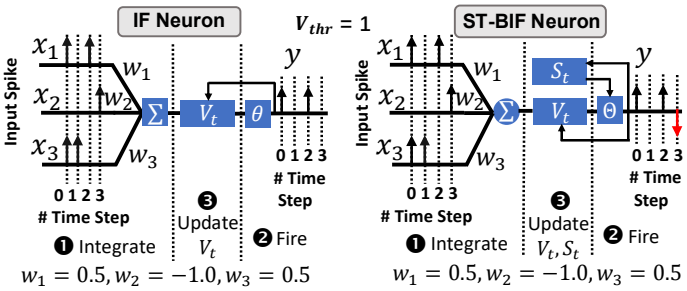


Fig. 1: The neural dynamics of IF and ST-BIF neurons. ST-BIF neuron fires bipolar spikes to eliminate overfiring imperfection [14] and prevent accuracy degradation.

I. Introduction

Recent advances in artificial intelligence (AI) have driven the demand for high-performance and low-power algorithms, making spiking neural networks (SNNs) a promising solution for energy-efficient AI systems [1]. SNN emulates the communications of neurons in the human brain to propagate with discrete asynchronous events (*i.e.*, spikes), enabling significant energy savings over traditional artificial neural networks (ANN). This potential attracts attention from both industry [2], [3] and academia [4], [5], accelerating the development of SNN-based neuromorphic hardware.

Prior studies primarily focus on the hardware [2]–[10] and algorithm [11]–[14] designs, which have achieved prominent advances in terms of performance (throughput, efficiency, *etc.*) and accuracy. [6]–[10] leverages processing-in-memory (PIM) architecture to break the "memory wall" and mitigate the communication overhead, leading to boosted performance over the ASIC accelerators [2]–[4]. Among the PIM-based neuromorphic accelerators, ReRAM-based designs [8], [9] present higher density, while SRAM-based designs [6], [7] achieve lower energy for the ADC-free charge-based computing scheme. The SNN algorithms, such as IF [11] and leaky-IF (LIF) [12] neurons, are continuously optimized to reduce energy and latency [11]–[14].

Despite the success of prior works, our investigations reveal that challenges remain in neuromorphic architecture design. First, existing PIM-based designs [6]–[10] predominantly adopt IF/LIF neurons, which suffer from over-firing issues [14]. The membrane potential of an IF/LIF neuron is hard-reset after the neuron firing spikes, leading to information loss and conversion mismatches (detailed in Section II-B), ultimately resulting in significant accuracy degradation. Second, while ReRAM-PIM [9] improves parallelism and throughput using small-sized arrays, they incur substantial area and power overhead by analog-digital-converters (ADCs). Although current sense amplifiers (CSA) are employed in [8] to eliminate ADC and reduce access time in large arrays, their benefits are marginal in small arrays [15], and continuous current sensing leads to increased power consumption. Lastly, most existing PIM-based SNN accelerators adopt synchronous architectures, where continuous clocking for membrane potential updates further contributes to excessive power consumption.

As a result, a more energy-efficient SNN accelerator with higher inference accuracy is highly desired. To this end, three key challenges must be addressed: 1) The limited computational expressiveness of IF/LIF neurons constrains scalability and hinders accuracy improvements; 2) Although small-sized ReRAM arrays enhance throughput, both ADC-based analog computing [9] and CSA-based digital computing [8] impose significant energy overheads; 3) As the system scales, reliance on power-hungry global clocks becomes a major bottleneck, conflicting with the inherently asynchronous behavior of SNNs.

This work proposes BiNeuroRAM, a novel asynchronous ReRAM-based PIM architecture that supports SNNs with emerging spike-tracing bipolar integrate-and-fire (ST-BIF) neurons [14], targeting high accuracy and energy efficiency. Unlike conventional hard-reset IF/LIF neurons, ST-BIF neurons require soft-reset, which is fundamentally incompatible with the hard-reset analog charge-based accumulation in [6]–[8]. BiNeuroRAM retains the analog charge-based accumulation from [16] but replaces the power-hungry ADC with asynchronous self-trigger converters (STC). Furthermore, we design a low-power voltage sense amplifier (LPVSA) dedicated to the small-sized ReRAM array sensing offering a compact and efficient design. Compared to SOTA SNN-based neuromorphic designs [2], [7]–[9], BiNeuroRAM achieves superior performance, the highest accuracy, and the lowest latency. Our contributions are:

- BiNeuroRAM is the first neuromorphic hardware that realizes ST-BIF neurons with delicate circuit designs. Negative spikes are supported to neutralize the over-firing energy without breaking the ANN-SNN conversion equivalence, thereby improving accuracy.
- BiNeuroRAM features a novel asynchronous micro-architecture

with an LPVSA tailored for energy-efficient PIM operations on small-size ReRAM arrays, enabling high parallelism and improved throughput. The ADCs are replaced by asynchronous self-trigger converters (STC), reducing area and power consumption.
- To fully exploit the event-driven nature of SNNs, BiNeuroRAM introduces sparsity-aware optimization on both input spikes and weight bits, reducing the power and deriving better efficiency.
- Experiments demonstrate that BiNeuroRAM outperforms state-of-the-art neuromorphic accelerators. BiNeuroRAM achieves 8.36% higher accuracy on ImageNet with ResNet-50, along with $2.08\times$ higher throughput density and $2.09\times$ better energy efficiency.

II. Background and Related Work

A. Spiking Neural Network

1) Introduction to SNN: Spiking Neural Networks (SNNs) are biologically-inspired neural networks that use time-dependent spikes for communication, offering energy efficiency and temporal dynamics, suitable for neuromorphic computing and real-time, event-driven tasks.

2) SNN Neuron Models: IF neuron is one of the simplest models and widely applied in neuromorphic accelerators [2]–[5]. However, "over-firing" (firing redundant spikes) [13] is introduced by the IF neuron, resulting in accumulated output errors and accuracy degradation. Besides, overfiring can incur additional power consumption and decrease the overall energy efficiency. LIF eliminates the overfiring by membrane potential leaking, but breaks the conversion equivalence and suffers from severe accuracy loss. To overcome the challenge, ST-BIF neuron is proposed [13], [14] to neutralize the over-fired positive spikes with negative ones thus strictly maintaining the conversion equivalence, as elaborated in Fig. 1. The dynamics of ST-BIF neuron are:

$$\begin{aligned} V_t &= V_{t-1} + V_t^{\text{in}} - V_{\text{thr}} \cdot \Theta(V_{t-1} + V_t^{\text{in}}, V_{\text{thr}}, S_{t-1}) \\ S_t &= S_{t-1} + \Theta(V_{t-1} + V_t^{\text{in}}, V_{\text{thr}}, S_{t-1}) \\ \Theta(V, V_{\text{thr}}, S) &= \begin{cases} 1; & V \geq V_{\text{thr}} \ \& \ S < S_{\max} \\ 0; & \text{other} \\ -1; & V < 0 \ \& \ S > S_{\min} \end{cases} \end{aligned} \tag{1}$$

where V_t is the membrane potential of the neuron at time-step t; V_{thr} is the threshold voltage for a neuron to fire a positive spike; S_t is the value of spike tracer at time-step t; V_t^{in} is the input voltage at time-step t; Θ is the firing behavior function of ST-BIF neuron.

Fig. 1 presents the example neural dynamics of both vanilla IF and ST-BIF neurons. The ST-BIF neuron differs from the vanilla IF neuron from two perspectives: (1) ST-BIF neuron fires bipolar (positive/negative) spikes; (2) ST-BIF neuron ensures the accumulated charge of output spikes always equals the output of the Q-ReLU function until the firing stops, in virtue of the spike tracer S_t.

3) SNN Applications: SNNs have been applied to various datasets, showcasing their strengths in energy-efficient and real-time processing for static vision task, neuromorphic vesion task, and natural language processing (NLP) task. In image recognition [13], [14], SNNs especially with ST-BIF neuron performs well on datasets like MNIST, CIFAR-10, and ImageNet, achieving competitive accuracy while being power-efficient. They excel in event-based vision tasks using datasets like CIFAR10-DVS [14] for gesture recognition, leveraging asynchronous event-driven computation. VISTREAM [17] as ST-BIF neuron SNN is also an energy efficient solution in visual streaming perception (VSP) task. While in Natural Language Processing (NLP) tasks, such as text classification and machine translation are available with spike-based models like IF neuron based Spikformer [18] and ST-BIF neuron based SpikeZIP-TF [14]. Their ability to handle temporal and event-driven data makes them ideal for low-power, real-time applications and competitive with ANNs solutions.

B. PIM for SNN Acceleration

PIM architecture for SNN acceleration has been widely explored and demonstrated to achieve remarkable performance [6]–[10]. Prior PIM [6]–[8] designs leverage analog charge-based neuron circuits by accumulating the membrane potential by charging the capacitor, comparing it with threshold voltage via a converter, and finally hard-reset (discharge) to zero after firing a spike. Such hard-reset-based design does not require ADC, however suffers from a significant accuracy degradation [13], [14]. Neuro-CIM [7] performs in-SRAM multiplication and charge-based accumulation, but is still power hungry due to the clock-driven comparator. Tempo-CIM [8] proposes charge-pump-based neuron circuits and uses time-domain converters to perform SNN operations but sacrifice SNN accuracy. [6] presents a clock-free architecture using a capacitor-based integration circuit but is limited to 4-bit weight accumulation only. NeuRRAM [9] proposes an analog neuron that supports various AI models with forward, recurrent, and backward dataflow, however using an area-power-expensive ADC. ANP-I [10] proposed a digital PIM with asynchronous design to achieve outstanding energy efficiency. Nevertheless, it is only demonstrated with toy applications with a limited number of synapses.

C. Motivation

1) Neuron Accuracy: Enhancing the accuracy of SNN algorithms is crucial for realizing their full potential in real-world applications, where energy efficiency, real-time processing, and high accuracy are essential. Addressing challenges such as spike-based computation and over-firing issues, the novel ST-BIF neuron models, when combined with advanced learning techniques, can significantly improve SNN performance, enabling them to achieve or even exceed the accuracy of conventional neural networks.

2) ReRAM-Based PIM: ReRAM is an emerging non-volatile memory that offers advantages like high density, low power, and non-volatility, making it ideal for PIM architectures where computation is performed directly in memory, minimizing data movement and energy consumption. However, optimizing several aspects of ReRAM-based PIM is crucial for improving overall system efficiency, scalability, and performance. 1) Enhancements in read operation, including optimizing sense amplifiers for better power efficiency and accuracy, are necessary to handle small voltage swings and reduce power consumption during memory access. 2) Addressing variability in resistance states, reducing write endurance issues, and improving parallel processing capabilities are critical for maximizing energy efficiency and ensuring robust performance in large-scale SNN neuromorphic systems. These optimizations lead to energy-efficient, scalable, and real-time AI systems for edge computing, robotics, and neuromorphic applications.

3) Asynchronous Step-by-Step SNN: Asynchronous step-by-step processing in SNN-based hardware lies in improving efficiency, scalability, and real-time performance. In traditional synchronous systems, all neurons update at fixed time intervals, leading to inefficient power usage and unnecessary computations. Asynchronous step-by-step processing allows neurons to spike and update only when triggered by an event, such as reaching a threshold, making it event-driven. This reduces power consumption, as neurons process information only when necessary, and minimizes latency by responding immediately to spikes, without waiting for a clock cycle. Additionally, this approach scales well with larger networks by enabling parallel, distributed computation, making it suitable for real-time applications like robotics, sensory processing, and neuromorphic computing. In summary, asynchronous step-by-step SNN hardware can achieve high

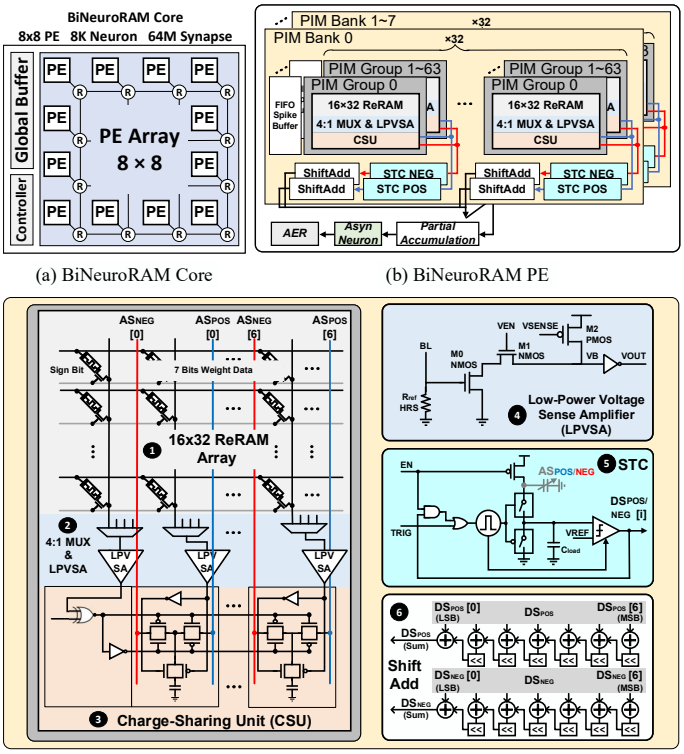


Fig. 2: (a) Architecture overview of BiNeuroRAM core, where 64 processing elements (PEs) are routed using a mesh-based network-on-chip (NoC); (b) Micro-architecture of BiNeuroRAM PE with 1024×1024 ReRAM array PIM bank, FIFO spike buffer, computing peripherals, and neuron. The column-wise in-memory compute units are connected to STC and output the membrane potential; (c) Detail circuit design of PIM group, LPVSA, STC, and ShiftAdd.

energy efficiency and faster response time, making it ideal for low-power AI applications.

III. BiNeuroRAM Architecture

The architecture overview of BiNeuroRAM is depicted in Fig. 2(a), along with the computation dataflow in Fig. 3. The BiNeuroRAM core consists of a global buffer, a controller, and 64 PEs connected using a mesh-based NoC. The global buffer temporarily stores the inputs and outputs, while the detailed configurations are given in Table I. In summary, one BiNeuroRAM Core supports 8K neurons and up to 64M synaptic weights. Note that BiNeuroRAM takes inference style of step-by-step (*aka.* multiple-time-step) asynchronously, which means the neuron membrane will constantly accumulate every time step across all SNN layers.

A. Hybrid Dataflow of BiNeuroRAM Architecture

In BiNeuroRAM architecture, PIM is leveraged using ReRAM to store weight values, enabling efficient and low-power neural computations. As illustrated in the left figure of Fig. 3, the weights of each kernel are stored in a stationary manner within the ReRAM cells. These weights are only accessed and read when the input streaming activates the corresponding wordline (WL). This weight stationary dataflow approach maximizes weight reuse by keeping the weights fixed in place, significantly reducing the energy-intensive operations associated with repeatedly moving weight data between memory and processing units.

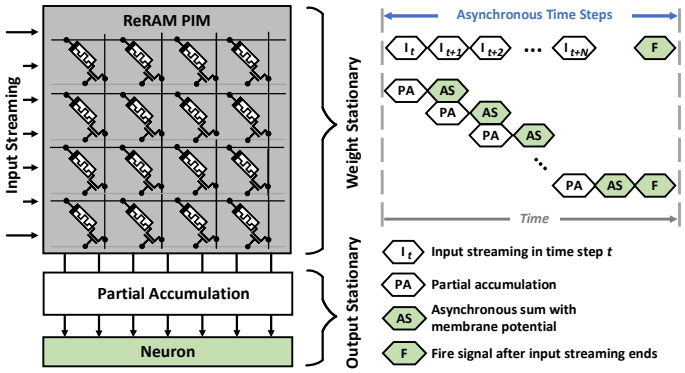


Fig. 3: Dataflow overview of BiNeuroRAM architecture, where it adapts weight stationary dataflow using PIM with output stationary dataflow to constantly accumulate the membrane potential before firing a spike.

Moreover, BiNeuroRAM operates in an inference style that is step-by-step and asynchronous. During this process, the membrane potential in the neuron is accumulated iteratively as the input streaming progresses. The inference continues until the entire set of input streaming is complete, at which point a spike is fired. This firing mechanism aligns naturally with the principles of output stationary dataflow, where the outputs (or membrane potentials) remain stationary in the neuron while the inputs are streamed through the system. The right figure of Fig. 3 illustrates this process in detail; each input stream accesses the ReRAM PIM to read the corresponding weights. These weights are partially accumulated, and the results are accumulated again to update the membrane potential in the neuron. This iterative process continues until all input streams have been processed. Once the computation is complete, a fire signal is generated and sent to the neuron. Upon receiving this signal, the neuron evaluates the accumulated membrane potential against the constraints defined in Eq. (1) and fires a spike if the conditions are met.

B. Microarchitecture of BiNeuroRAM PE

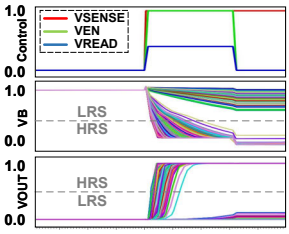
For each BiNeuroRAM PE as illustrated in Fig. 2(b), it is composed of 8 PIM banks, 128 ways accumulators (*i.e.*, 16 ways per bank), and 128 neurons (*i.e.*, 16 neurons per bank). PIM banks are designed to perform weight-stationary SNN inference in a mixed-signal fashion. The accumulator is used to accumulate the partial sums generated by the PIM banks, and then the neurons are leveraged for output firing. An address event representation (AER) circuit converts the output neuron spike events to spike address representation.

1) PIM Bank: In Fig. 2(b), it shows each 8 PIM bank is made up with 64 × 32 PIM groups, while each PIM group is composed by 16 × 32 1-transistor-1-ReRAM (1T1R) bit cells sub-array, leading to a total capacity of 128-kB (1024×1024). All the sub-arrays are sensed via our ❹ low-power voltage sense amplifier (LPVSA). Note that within one PIM bank, PIM groups are physically halved by row driver for sufficient drivability to activate WLs. In addition, the PIM bank consists of 448-ways self-trigger converter (STC) and 16 ways of Shift and Add units (ShiftAdd). Fig. 2(c) shows the detail circuit of ❺ STC and ❻ ShiftAdd.

2) PIM Group: The PIM group plays a major role in handling charge-sharing matrix multiplication (MM) operations. Fig. 2(c) shows the detail circuit of PIM group. PIM group consists of 3 major parts for enabling the charge-sharing process: ❶ 16 × 32 ReRAM array, ❷ 4:1 MUX & LPVSA, and ❸ charge-sharing unit (CSU).

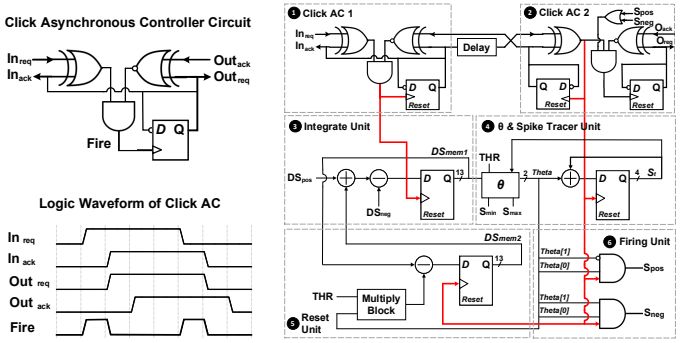
Corner	TT	SS	FF
R_{read}	LRS / HRS		
Latency (ns)	0.8	1.2	0.3
Power (uW)	1.8 / 0.066	1.64 / 0.056	5.09 / 1.03
Energy (fJ)	1.44 / 0.053	1.97 / 0.067	1.53 / 0.31

(a) Evaluation



(b) Monte Carlo Analysis

Fig. 4: LPVSA (a) evaluation over different corners and (b) Monte Carlo analysis for accurate functionality.



(a) Asynchronous Controller (b) ST-BIF neuron circuit design

Fig. 5: (a) Click-based asynchronous controller integrated into the (b) ST-BIF neuron circuit design with asynchronous clock.

❶ 16×32 **ReRAM array** will store 4 different signed 8-bit synaptic weights data in a single row. The first 4 columns store the sign-bit of the 4 weights, while the remaining columns store the 7-bit unsigned integer (*i.e.*, representing $-127 \sim 127$). This design can provide a dense memory capacity and allow sufficient space for our sensing circuit.

❷ **4:1 MUX** & **LPVSA** will select the correct ReRAM array column based on the AER spike input to sense data. The different currents generated by the varied conductance of the HRS/LRS ReRAM cells will be sensed by our proposed LPVSA connected at the end of the BL. The details of our proposed LPVSA will be described in ❹.

❸ **Charge-sharing unit (CSU)** is integrated into the PIM group to allow charge-sharing operation. It consists of two parts: multiplication unit and 6T1C charge decision unit. The multiplication unit includes an XNOR-gate connecting both signed bits of activation and weight. This helps to determine the polarity of the accumulated results and allows the charge decision unit to activate the corresponding capacitive analogy-summation (AS) wire: blue positive-polarity (AS_{pos}) or red negative-polarity (AS_{neg}).

C. Sensing Circuits

1) Low-Power Voltage Sense Amplifier (LPVSA): As shown in ❹, LPVSA is proposed for a small ReRAM array to replace CSA in heavy load BL. The proposed LPVSA utilizes the discharge voltage on the M2 PMOS and then senses via a low static current path by the M0 NMOS. Firstly, the M2 PMOS is disabled by VSENSE to precharge the sensing circuit to V_{DD}. When an input spike enters, M1 NMOS will pull up by VEN to stop charging and M2 PMOS will pull down by VSENSE signal to start sensing. Meanwhile, the sensing reference resistance R_{ref} is set to HRS as a voltage divider. Since the M0 NMOS are adjusted to a small size, a weak voltage level of ≈400mV (VREAD) can pull up the transistor for grounding when the input is LRS. Finally, the sensing process is completed and the inverter amplifies the precharged voltage, VB for output results, VOUT. If the input is HRS, the gate voltage of M0 is 200mV, which is insufficient to drive the M0 NMOS. Fig. 4a evaluated the performance of our LPVSA when sensing HRS/LRS over different corners. We also completed Monte Carlo analysis to ensure accurate functionality of LPVSA, as shown in Fig. 4b.

2) Self-Trigger Converter (STC): A ❺ STC [19] is integrated to eliminate the usage of the power-hungry clock from ADC when conducting MM operation. From Fig. 2(b), AS_{POS} and AS_{NEG} conduct accumulation and conversion using two different STCs ($STC\ POS$ and $STC\ NEG$). Initially, in the absence of triggering either the start EN or TRIG signal, V_{DD} will precharge the $C_{\text{AS(pos)}}$ and $C_{\text{AS(neg)}}$, while C_{load} will reset to V_{SS}. When there is a spike activity, the pulse generator is triggered and stops the charging-resetting process of $C_{\text{AS(pos)}}$, $C_{\text{AS(neg)}}$, and C_{load}. Then, it starts comparison between V_{AS} (V_{load}) and V_{ref}. If V_{load} is greater than V_{ref}, the pulse generator will trigger again for the next comparison. While C_{load} will reset and charge-share with C_{AS} again until V_{ref} is larger than V_{load}, then a stop signal is passed to the next stage. A maximum of 64 conversions will be imposed if all 64 capacitors are activated in a single AS line. The number of conversions at the $STC\ POS$ and $STC\ NEG$ also referred to the digital output of $C_{\text{AS(pos)}}$ and $C_{\text{AS(neg)}}$ activated. Lastly, the total conversion pulses from STC will pass to ShiftAdd to accumulate the total membrane potential.

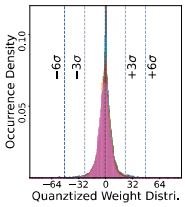
D. ST-BIF Neuron Circuit

1) Asynchronous Controller: Instead of using a synchronous global clock, hand-shake signals in asynchronous logic are more energy efficient and compatible with the multiple-time-step data flow for SNN. Fig. 5a shown the design and logic waveform of Click [20] asynchronous controller (AC). The *req* and *ack* handshake signals are used for data transmission between the sender and receiver. The sender transmits the *req* signal when data are available, while the receiver captures the data and returns an *ack* signal. In the logic waveform, when In_{req} toggles, indicating a valid input event, the input channel fills up, and the fire signal is activated. The rising edge of the fire signal then triggers the DFF to invert the Out_{req} and In_{ack} signals, which resets the phase-decoupled Click signal to its initial state.

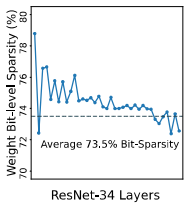
2) Neuron Circuit: As shown in Fig. 5, the neuron circuit is designed to mimic the ST-BIF bipolar neuron activity based on the neuron model described in Eq. (1). Both ❶ AC1 and ❷ AC2 will adopt a handshake signal to ensure data synchronization. When ❶ AC1 detects In_{req} signal, ❸ integration unit will accumulate the new signed DS_{POS} and DS_{NEG} as the digital membrane potential, DS_{mem1}. Then, DS_{mem1} enters the θ block in ❹ θ & spike tracer unit to check the firing conditions in Eq. (1). A *Theta* value of signed 2b'01 will be generated if $DS_{\text{mem1}} >$ THR and $S_{\text{t}} < S_{\text{max}}$; a *Theta* value of 2b'11 will be generated if $DS_{\text{mem1}} < 0$ and $S_{\text{t}} > S_{\text{min}}$; Otherwise, a *Theta* value of 2b'00 will be generated when no conditions are satisfied. The ❷ AC2 will fire the DFF in ❹ to record the theta value (spike fired) in the 4-bit spike tracer unit as S_{t}. If a non-zero theta value is generated, it will enable ❺ reset unit for soft-reset by adding or deducting DS_{mem1} with the threshold value, THR. The ❷ AC2 fire signal will control the DFF for updated membrane potential, DS_{mem2}, and pass to ❸ for the next integration. In the meantime, the non-zero theta value and ❷ AC2 fire signal conduct AND3 operation at the ❻ firing unit to fire positive, S_{pos} or negative, S_{neg} spike output. Once the

Models	Spike Ratio (%)		
	-1	0	+1
MLP	0.62	90.1	9.28
ResNet12	1.5	94.2	4.3
ResNet18	4.7	84.3	11.0
ResNet20	3.8	86.9	9.3
ResNet34	4.1	86.0	9.9
ResNet50	6.9	80.6	12.5
Average	3.6	87.0	9.4

(a)



(b)



(c)

Fig. 6: (a) Ratio of -1/0/+1 spikes in four representative models; (b) Distribution of 8-bit quantized ResNet-34 weights; (c) Layer-wise bit-sparsity of quantized ResNet-34 weights.

neuron operation is completed, ❷ AC2 will pass the output request, O_{req} signal to the next AC receiver.

E. Sparsity Optimization

1) Input Sparsity: Since SNN inference operation is sparse and event-driven, a power gating method is applied to the CSU to keep the CSU in idle status when the input spikes (activations) are absent. According to our profiling results on the classical ResNet model [21] listed in Fig. 6a, the input spikes are sparse for 87.0% on average. Such power gating can lead to an average of 9.2× power reduction.

2) Weight Sparsity: Since the ReRAM read circuit consumes different energy *w.r.t* the resistance of the ReRAM cell, mapping the HRS (lower power) to more frequently occurring data can significantly reduce the average power. We analyze the data distribution of 8-bit quantized ResNet-34 networks and obtain the ratio of bit-0 and 1, as shown in Fig. 6c. According to the chart, the frequency of bit-0 is 2.8× higher than bit-1 on average. Therefore, should be mapped to lower power consumption HRS.

IV. Evaluation

A. Experimental Setup

We conclude all the key elements below for our BiNeuroRAM architecture and build a dedicated in-house cycle-accurate simulator to evaluate the overall performance. The digital circuits are designed in RTL and synthesized using Synopsys Design Compiler with 28nm standard cell library. We use Cadence Innovus to perform place-and-route, Synopsys VCS for gate-level verification, and Synopsys PrimeTime PX for power analysis. The analog modules are designed and verified using Spectre simulator, with the area obtained at the layout level. The analysis of the NoC modules is conducted using ORION [22] with a 32-bit bus width. Note that, the evaluation of ORION is of 32nm technology originally, and we scale the results to 28nm in this work. The circuit model of the ReRAM model is based on our experimentally-calibrated model [23] in Verilog-A. NVSim [24] is leveraged to model the memory architectural performance, including the read/write latency and energy on the data path. The buffer is evaluated using a foundry 28nm memory compiler. Further details of the BiNeuroRAM configurations and specifications are summarized in Table I.

B. Dataset and Baseline

The SNN parameters for different datasets that are used for inference evaluation along with the network models, namely MNIST [25] 512-256-10 MLP (accuracy and latency only), CIFAR-10 ResNet-12 (W1), CIFAR-10 ResNet-20 (W2) [26], ImageNet ResNet-18 (W3), ImageNet ResNet-34 (W4), and ImageNet ResNet-50 (W5) [27]. All the SNN models are pre-trained via the ANN-to-SNN conversion training method introduced in [14]. For SNN inference, the input spikes are analog-coded [14] with inference style of step-by-step

TABLE I: BiNeuroRAM configurations and specifications.

Component	Configuration	Area (mm^2)	Power (mW)
PIM Bank Spec.			
ReRAM	1024×1024 1-bit/cell LRS: 15kΩ HRS: 265kΩ	0.0088	0.0865
Compute Logic	64×128	0.0126	0.0526
STC	448	0.0085	0.0339
PIM PE Spec.			
Bank	8	0.238	1.384
Neuron	128	0.0433	0.0285
Accumulator	128	0.0035	0.0021
Router	Weight: 32-bits	0.0081	1.489
BiNeuroRAM Core Spec.			
PE	64	18.8	185.8
NoC	Topology: Mesh Weight: 32-bits	0.074	0.0013
Controller	-	1.125	31.44
Global Buffer	32 KB	0.061	6.365
Total	-	**20.03**	**223.7**

(aka. multiple-time-step) asynchronously. All the neuron activations are based on the proposed bipolar-based, ST-BIF neuron inference with 2-bit signed input and 8-bit signed weight. For CIFAR-10 datasets, the neuron inference with 4-bit signed weight provides fair comparisons between SOTA. The throughput (TOP/S) is evaluated using a similar method as Neuro-CIM [7]. As recent works [7]–[9] on SNN architecture and circuits focus on IF and LIF SNN activation, it is not fair to compare them with BiNeuroRAM bipolar SNN. Therefore, we design an ADC-based [28] bipolar neuromorphic accelerator (BiNeuroRAM without LPVSA & STC) as the baseline for further ablation studies. Furthermore, we mainly compare the overall performance of BiNeuroRAM with SOTA SNN accelerators (TrueNorth [2], Neuro-CIM [7], NeuRRAM [9], and Tempo-CIM [8]).

C. Performance Comparison

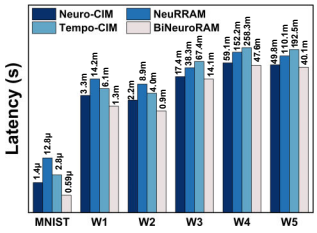
Table II summarizes the performance with SOTA accelerators (TrueNorth, Neuro-CIM, NeuRRAM, and Tempo-CIM). In contrast, BiNeuroRAM is the only design that supports bipolar neuron polarity and benefits from high inference accuracy with low latency. The energy efficiency gain of BiNeuroRAM comes from the low-power design of our LPVSA and STC with high sparsity SNN operation.

1) Inference Accuracy & Latency: As discussed previously, the bipolar neuron model of ST-BIF can mitigate the accuracy drop of SNN converted by the ANN-to-SNN algorithm, in return for a promising inference accuracy. In Fig. 7, BiNeuroRAM outperforms SOTA works in terms of accuracy and latency with different datasets. Moreover, our results for MNIST dataset with 3-layer MLP reaches an accuracy of 99.2%, CIFAR-10 dataset with ResNet-20 can achieve an accuracy of 89.2%, and ImageNet dataset with ResNet-50 achieve an accuracy of 80.9%. Moreover, the latency of SOTAs is evaluated using normalized results based on the throughput ratio. The result in Fig. 7b shows that our latency for different inference workloads is the lowest among SOTAs.

2) Energy Breakdown: Fig. 8 shows the energy breakdown of BiNeuroRAM for different models and datasets. STC's highest energy consumption in every evaluation is primarily due to its extended operation period and elevated power requirements. The significant power demand of STC also results in a high toggling rate for generating pulses. Although with the high power requirement, Fig. 9a addresses STC as the better solution than ADC in our architecture. ReRAM and LPVSA have the second-highest energy consumption,

Dataset	Model	Accelerator	Acc (%)
MNIST	7-layer	NeuRRAM	99.0
	3-layer	Tempo-CIM	96.9
	3-layer	**Our Work**	**99.2**
CIFAR-10	ResNet-12	Neuro-CIM	86.4
	ResNet-12	**Our Work(W1)**	**93.7**
	ResNet-20	NeuRRAM	85.7
	ResNet-20	**Our Work(W2)**	**89.2**
ImageNet	ResNet-18	**Our Work(W3)**	**69.5**
	ResNet-34	**Our Work(W4)**	**72.2**
	ResNet-50	Neuro-CIM	72.5
	ResNet-50	**Our Work(W5)**	**80.9**

(a) Accuracy



(b) Latency

Fig. 7: BiNeuroRAM (a) accuracy and (b) latency compared with SOTA in different models and datasets.

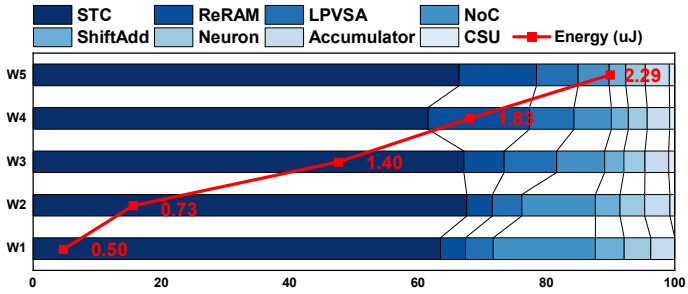


Fig. 8: BiNeuroRAM normalized energy breakdown over different models and datasets.

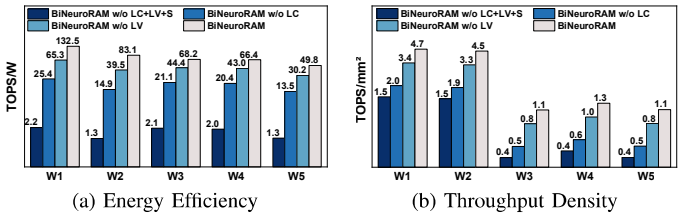
primarily due to their support for constant read operations on large numbers of small-size ReRAM arrays. Furthermore, a large percentage of power from NoC is due to the high static power of the router. The figure also demonstrates the total energy consumption when inference using different datasets and models. BiNeuroRAM achieves energy consumption of 0.5μ - 0.73μJ for CIFAR-10 classification and 1.4μ - 2.29μJ for ImageNet classification.

3) Performance Analysis: The overall performance for energy efficiency and throughput density of BiNeuroRAM design is tabulated in Table II. In addition, Fig. 9 illustrates the energy efficiency and throughput density over different datasets and evaluation models respectively. When performing CIFAR-10 classification with ResNet-12, BiNeuroRAM can achieve 132.5TOPS/W, 1.1× higher than Neuro-CIM. BiNeuroRAM also outperforms Neuro-CIM by 2.1× for ImageNet classification with our energy efficiency up to 68.2TOPS/W at ResNet-18. Our throughput density varies across different datasets for up to 4.72TOPS/mm^2, achieving 2.1×, 16.9×, and 6743× breakthrough with Neuro-CIM, Tempo-CIM, and TrueNorth, respectively.

D. Ablation Study

We conduct an ablation study to analyze the benefits of *STC* & *LPVSA* circuit design for improving energy efficiency. Therefore, we design an ADC to replace STC and LVPSA in BiNeuroRAM as the baseline design. The comparison results are illustrated in Fig. 9. On average, the energy efficiency and throughput density of BiNeuroRAM can be improved by 46.1× and 3.1× respectively over ADC-based design.

Furthermore, we focus on the benefits of LPVSA. Compared to traditional CSA [30] and voltage sense amplifier (VSA) [29], our LPVSA can reduce the ReRAM read power by 58.2× and 14.7× respectively. LPVSA also able to reduce area overhead by 24× and 8.8× respectively. In the system level performance evaluation in Fig. 9, BiNeuroRAM with LPVSA can improve average by 4.2× and 2.3× for CSA, 1.8× and 1.4× for VSA of energy efficiency and throughput



(a) Energy Efficiency (b) Throughput Density

Fig. 9: BiNeuroRAM (a) energy efficiency and (b) throughput density performance compared without LPVSA (replaced with VSA [29] - LV; & replaced with CSA [30] - LC), and STC (replaced with ADC [28] - S).

TABLE II: Comparison with previous SNN chip designs.

	TrueNorth TCAD'15 [2]	Neuro-CIM JSSC'23 [7]	NeuRRAM Nature'22 [9]	Tempo-CIM JETCAS'23 [8]	**BiNeuroRAM This Work**
Application	SNN	SNN	ANN/SNN	SNN	SNN
Technology	28nm CMOS	28nm CMOS	130nm CMOS	40nm CMOS	28nm CMOS
Bitcell	12T SRAM	8T SRAM	ReRAM	ReRAM	ReRAM
Macro Size	16×256	64×160	256×256	256×64	1024×1024
Neuron operation	Digital	Charge-based Mixed-Signal	Analog	Charge Pump -based Analog	Charge-based Mixed-Signal
Area (mm^2)	430	2.9	159	0.084	20.03
Activation Precision	-	1∼8	1/2/4/8	4	2
Weight Precision	-	1/4/8	4	2/4/8	1∼8
Inference Accuracy (MNIST)	-	-	99.0$^{\triangleleft}$(↓0.2%)	96.9$^{\star\times}$(↓2.3%)	**99.2**$^{\star\times}$
Inference Accuracy (CIFAR-10@RN-12)	-	86.4 (↓7.3%)	-	-	**93.7**
Inference Accuracy (CIFAR-10@RN-20)	-	-	85.7 (↓3.5%)	-	**89.2**
Inference Accuracy (ImageNet@RN-50)	-	72.5 (↓8.4%)	-	-	**80.9**
Energy Efficiency (TOPS/W)	0.4	32.61† - 124.15$^{\diamond}$	67.8$^{\star\divideontimes}$ - 416.5$^{\star\divideontimes}$	168.4$^{\star\times}$	**49.8**‡ - **132.5**$^{\diamond\ddagger}$
Throughput Density (TOPS/mm^2)	0.0007	2.26$^{\diamond}$	0.0079$^{\star\divideontimes}$ - 0.0622$^{\star\divideontimes}$	0.28$^{\star\times}$	**1.15**‡ - **4.72**$^{\diamond\ddagger}$
Neuron Type	20 Types	IF	IF	LIF	**IF, ST-BIF**

$\star$: Normalized to 28nm and 0.9V based on $P \propto CV^2$, for fair comparison; †: ImageNet @ ResNet-18
$\diamond$: CIFAR-10 @ ResNet-12; $\times$: MNIST 3-Layers MLP; $\triangleleft$: MNIST 7-Layers MLP;
$\divideontimes$: Measured for performing 256×256 matrix-vector multiplications;
‡: Evaluate using similar method as Neuro-CIM [7] (# of op = 2×# of accumulations/time window)

density, respectively. In short, the proposed LPVSA and STC show a significant design for energy-efficient SNN PIM.

V. Conclusion

We present a high-accuracy bipolar-based neuromorphic design using emerging NVM ReRAM sense via our energy efficient LPVSA, namely BiNeuroRAM, with asynchronous microarchitecture and sparsity optimization techniques to improve energy efficiency. The design of LPVSA significantly improved overall system-level energy performance by enabling the implementation of small array size, which increased compute parallelism. Overall, BiNeuroRAM represents a significant breakthrough in integrating bipolar SNN neurons with PIM architecture, achieving up to 7.3% and 8.4% higher accuracy on the CIFAR-10 and ImageNet datasets, respectively, compared to SOTA SNN PIM designs. Additionally, it delivers energy efficiency improvements of up to 1.1× and 2.1× for CIFAR-10 and ImageNet, respectively, over the current best-performing SNN PIM designs. To our knowledge, BiNeuroRAM is the first bipolar spiking SNN neuromorphic accelerator with asynchronous sparse processing-in-memory, achieving competitive energy efficiency and high accuracy compared to IF neuromorphic hardware designs.

VI. Acknowledgment

This work was supported in part by National Key Research and Development Program of China under Grant 2022YFB4500200 and the National Natural Science Foundation of China under Grant 62102257.

REFERENCES

[1] Kaushik Roy, Akhileshc Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784), 2019.

[2] Filipp Akopyan, Jun Sawada, Andrew Cassidy, Rodrigo Alvarez-Icaza, John Arthur, Paul Merolla, et al. Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip. *IEEE TCAD*, 34(10):1537–1557, 2015.

[3] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, et al. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, 2018.

[4] Lei Deng, Guanrui Wang, Guoqi Li, Shuangchen Li, Ling Liang, Maohua Zhu, et al. Tianjic: A unified and scalable chip bridging spike-based and continuous neural computation. *IEEE JSSC*, 55(8):2228–2246, 2020.

[5] Steve B. Furber, Francesco Galluppi, Steve Temple, and Luis A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, 2014.

[6] Ying Liu, Zhixuan Wang, Wei He, Linxiao Shen, Yihan Zhang, Peiyu Chen, et al. An 82nw 0.53pj/sop clock-free spiking neural network with 40μs latency for alot wake-up functions using ultimate-event-driven bionic architecture and computing-in-memory technique. In *2022 IEEE ISSCC*, volume 65, pages 372–374, 2022.

[7] Sangyeob Kim, Sangjin Kim, Soyeon Um, Soyeon Kim, Kwantae Kim, and Hoi-Jun Yoo. Neuro-cim: Adc-less neuromorphic computing-in-memory processor with operation gating/stopping and digital–analog networks. *IEEE Journal of Solid-State Circuits*, 58(10):2931–2945, 2023.

[8] Jingwen Jiang, Keji Zhou, Jinhao Liang, Fengshi Tian, Chenyang Zhao, Jianguo Yang, et al. Tempo-cim: A rram compute-in-memory neuromorphic accelerator with area-efficient lif neuron and split-train-merged-inference algorithm for edge ai applications. *IEEE JETCAS*, 2023.

[9] Weier Wan, Rajkumar Kubendran, Clemens Schaefer, Sukru Burc Eryilmaz, Wenqiang Zhang, Dabin Wu, et al. A compute-in-memory chip based on resistive random-access memory. *Nature*, 608:504–512, 2022.

[10] Jilin Zhang, Dexuan Huo, Jian Zhang, Chunqi Qian, Qi Liu, Liyang Pan, et al. Anp-i: A 28-nm 1.5-pj/sop asynchronous spiking neural network processor enabling sub-0.1-uj/sample on-chip learning for edge-ai applications. *IEEE JSSC*, 59(8):2717–2729, 2024.

[11] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in Neuroscience*, 2017.

[12] Youngeun Kim, Yuhang Li, Abhishek Moitra, Ruokai Yin, and Priyadarshini Panda. Sharing leaky-integrate-and-fire neurons for memory-efficient spiking neural networks. *Frontiers in Neuroscience*, 17, 2023.

[13] Yangfan Hu, Qian Zheng, Xudong Jiang, and Gang Pan. Fast-snn: Fast spiking neural network by converting quantized ann. *arXiv preprint arXiv:2305.19868*, 2023.

[14] Kang You, Zekai Xu, Chen Nie, Zhijie Deng, Qinghai Guo, Xiang Wang, and Zhezhi He. Spikezip-tf: Conversion is all you need for transformer-based snn. In *2024 ICML*, 2024.

[15] Meng-Fan Chang, Albert Lee, Pin-Cheng Chen, Chrong Jung Lin, Ya-Chin King, Shyh-Shyuan Sheu, and Tzu-Kun Ku. Challenges and circuit techniques for energy-efficient on-chip nonvolatile memory using memristive devices. *IEEE JETCAS*, 5(2), 2015.

[16] Zhewei Jiang, Shihui Yin, Jae-Sun Seo, and Mingoo Seok. C3sram: An in-memory-computing sram macro based on robust capacitive coupling computing mechanism. *IEEE JSSC*, 55(7):1888–1897, 2020.

[17] Kang You, Ziling Wei, Jing Yan, Boning Zhang, Qinghai Guo, Yaoyu Zhang, and Zhezhi He. Vistream: Improving computation efficiency of visual streaming perception via law-of-charge-conservation inspired spiking neural network. In *2025 CVPR*, 2025.

[18] Zhaokun Zhou, Kaiwei Che, Wei Fang, Keyu Tian, Yuesheng Zhu, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer v2: Join the high accuracy club on imagenet with an snn ticket. *arXiv preprint arXiv:2401.02020*, 2024.

[19] Ying Liu, Yufei Ma, Ninghui Shang, Tianhao Zhao, Peiyu Chen, Meng Wu, et al. 30.2 a 22nm 0.26nw/synapse spike-driven spiking neural network processing unit using time-step-first dataflow and sparsity-adaptive in-memory computing. In *2024 IEEE ISSCC*, 2024.

[20] Ad Peeters, Frank te Beest, Mark de Wit, and Willem Mallon. Click elements: An implementation style for data-driven compilation. In *2010 IEEE ASYNC*, pages 3–14, 2010.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE CVPR*, pages 770–778, 2016.

[22] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *DATE*, pages 423–428, 2009.

[23] Yueyang Jia, Sheng Shen, Maosong Xie, Pengcheng Zhang, Minliang Shen, and Rui Yang. A consistent model for gradual, abrupt, and abnormal reset phenomena in bipolar/unipolar metal oxide rrams. *IEEE TED*, 71(5):3142–3149, 2024.

[24] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. Nvsim: A circuit-level performance, energy, and area model for emerging non-volatile memory. *IEEE TCAD*, 31(7), 2012.

[25] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[26] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[27] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE CVPR*, pages 248–255, 2009.

[28] Juzheng Liu, Mohsen Hassanpourghadi, and Mike Shuo-Wei Chen. A 10gs/s 8b 25fj/c-s 2850um2 two-step time-domain adc using delay-tracking pipelined-sar tdc with 500fs time step in 14nm cmos technology. In *2022 IEEE ISSCC*, volume 65, pages 160–162, 2022.

[29] Meng-Fan Chang, Jui-Jen Wu, Tun-Fei Chien, Yen-Chen Liu, Ting-Chin Yang, Wen-Chao Shen, et al. Low vddmin swing-sample-and-couple sense amplifier and energy-efficient self-boost-write-termination scheme for embedded reram macros against resistance and switch-time variations. *IEEE JSSC*, 50(11):2786–2795, 2015.

[30] Running Guo, Ruolan Jia, Stefan Pechmann, Marc Reichenbach, and Amelie Hagelauer. A high-speed low-power sense amplifier for the rram array with multi-level reading function using 130-nm technology. In *2023 ICECS*, pages 1–4, 2023.

PoP-ECC: Robust and Flexible Error Correction against Multi-Bit Upsets in DNN Accelerators

Taewon Park†, Saeid Gorgin‡, Dongwhee Kim†, Jaeho Shin‡, Michael B. Sullivan§, and Jungrae Kim‡
†Samsung Electronics, South Korea ‡Sungkyunkwan University, South Korea § NVIDIA, USA
taeng1.park@samsung.com, gorgin81@skku.edu, dongwhee.kim@samsung.com, wogh533@g.skku.edu, misullivan@nvidia.com, dale40@skku.edu

***Abstract*—Deep Neural Networks (DNNs) in safety-critical systems require high reliability. Many systems deploy Error Correction Codes (ECCs) to protect DNNs from memory errors. However, continuous process scaling increases memory errors in severity and frequency, necessitating strong protection against Multi-Bit Upsets (MBUs). This paper proposes *Parities of Parities ECC (PoP-ECC)*, a novel two-tier memory protection scheme designed to provide robust, efficient, and flexible protection against MBUs. PoP-ECC generates *Virtual Parities (VPs)*, which are used to compute second-level parities called Parities of Parities (PPs). This two-level ECC structure allows for dynamic error correction tailored to varying error patterns, ensuring system reliability with minimal memory overhead. Our evaluation demonstrates that PoP-ECC can tolerate significantly higher MBU ratios compared to state-of-the-art solutions, with negligible delay, area, and power overhead.**

***Index Terms*—Reliability, ECC, SRAM, Deep Neural Network.**

I. Introduction

With the proliferation of AI, Deep Neural Networks (DNNs) are widely used in safety-critical systems, including automotive [2], robotics [30], and aerospace [46] applications. DNN reliability is critical in such systems to ensure safety and prevent systems from failure. While DNNs are inherently resilient to small data noises, errors in high-bit positions can significantly impact their accuracy, posing a significant challenge to reliability.

One of the major concerns in DNN reliability is SRAM errors. A particle strike on SRAM can change the stored value, whereas combinational logic is less vulnerable to errors with temporal and logical masking [31]. Therefore, SRAM errors are already relatively common in modern systems [47], [48], and high-reliability systems protect their SRAMs with Error Correction Codes (ECCs). Moreover, DNN accelerators have a large amount of on-chip SRAM (e.g., 81% of chip area in Eyeriss [7]) to avoid high-energy off-chip memory accesses. With a large chip area and high error vulnerability, DNN requires strong SRAM protection for reliable operation.

Although most DNN reliability studies focus on protection against random bit errors, shrinking process technology demands strong protection against Multi-Bit Upsets (MBUs). With smaller feature sizes and lower supply voltage, the amount of charges to flip an SRAM cell (i.e., critical charge) keeps decreasing. As a result, many studies report that the probability of multiple adjacent errors is increasing [35], [32].

A common solution to DNN reliability is using ECC. However, the large SRAMs in DNN accelerators make the conventional 12.5% ECC redundancy a significant overhead to overall chip size. To hide the storage overhead, some studies exploit the resiliency of DNN against small noises. They utilize aggressive quantization to reduce data size and make room for redundancy storage. For instance, Weight Nulling [37] saves one bit per weight and utilizes the bit as a parity to detect single-bit errors. If an error is detected, its erroneous weight is regarded as zero. Value-aware Parity Insertion ECC [29] replaces redundant under-utilized bits in a data block with ECC redundancy. These schemes can improve memory protection efficiency by re-purposing weight bits for redundancy and maintaining reliability levels in some models. However, they primarily target single-bit errors and provide weak protection against MBUs.

This paper proposes a novel memory protection scheme, called PoP-ECC, to enhance DNN reliability against MBUs. PoP-ECC starts by generating a *Virtual Parity (VP)* for each weight. Next, it creates a second level of redundancy, referred to as *Parities of Parities (PPs)*, based on these VPs. Instead of storing the VPs, PoP-ECC stores only the PPs alongside the weights. The decoding process re-generates VPs from the received data, restores corrupted VPs using PPs, and restores corrupted weights using VPs.

This two-level ECC provides both flexibility and efficiency in addressing MBU challenges. The VP generation can be tailored to handle various MBU patterns, with this paper focusing on Double Adjacent Errors (DAEs) due to their severity. Additionally, generating parities from VPs, which are smaller in size than their corresponding weights (e.g., 4 bits versus 6 or 8 bits), significantly reduces redundancy overhead compared to directly generating parities from weights.

Building on these strengths, this paper explores the integration of weight quantization with PoP-ECC to design an efficient SRAM protection scheme. The proposed method, Q+PoP, leverages the inherent robustness of DNNs to small data noises by applying quantization, thereby freeing up memory space for PoP-ECC redundancy. Through channel-wise quantization, Q+PoP minimizes accuracy degradation while ensuring sufficient space for PoP-ECC. This results in a highly efficient and reliable memory protection scheme tailored to modern DNN accelerators.

Our evaluation shows the efficiency and strength of Q+PoP. Although it quantizes weights to 6-bit, the channel-wise quantization leads to less than 0.5% in 2 out of 3 DNN models (1.02% on average). In terms of reliability, PoP-ECC

can maintain the accuracy without decreasing by $> 0.1\%$ percentage point even if the bit error ratio (BER) increases up to 3×10^{-3}; the ratio is up to $3.16\times$ higher than state-of-the-art schemes. More importantly, it can tolerate up to $31.62\times$ higher DAE ratio than the state-of-the-art schemes. By providing an order of magnitude stronger protection against MBUs, PoP-ECC can improve DNN reliability significantly, as overall reliability is dominated by the weakest protection.

The major contributions of this paper are as follows:

- This paper introduces a novel ECC scheme called *Parities-of-Parities ECC (PoP-ECC)*, which protects virtual data instead of raw data. PoP-ECC can customize error correction to target MBU patterns and reduces redundancy, making it highly efficient for modern SRAM reliability challenges.
- Our evaluation demonstrates that PoP-ECC tolerates up to 31.62× higher DAE ratios than state-of-the-art schemes, with less than 0.1 percentage points accuracy degradation compared to error-free cases, ensuring strong protection with minimal impact on performance.
- The proposed *Q+PoP* integrates channel-wise quantization with PoP-ECC to enhance the reliability of DNN inference. This approach provides robust protection against MBUs while preserving the SRAM footprint, balancing efficiency and reliability.

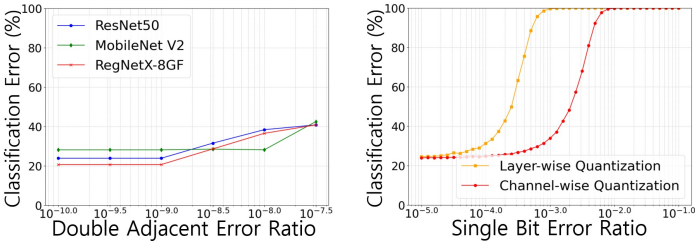
II. Background and Motivation

This section introduces the impacts of SRAM errors and quantization on DNN reliability, which motivated this study.

A. DNNs and SRAM Errors

DNN processors commonly equip a large amount of SRAMs. The primary design target of the processors is energy efficiency, yet external DRAM access consumes more than an order of magnitude higher energy than on-chip SRAMs [22]. Accordingly, many DNN processors size up on-chip SRAM buffers and optimize reuse patterns to keep DRAM accesses to minimum [7], [28], [23], [18], [16], [50], [14], [19]. Eyeriss[7], for instance, utilizes 81% of its core chip area (i.e., the area without I/O pads) for its global buffer, scratchpad memory, and FIFOs. At the extreme, many DNN accelerators do not feature an external DRAM interface [4], [8], [15], [13], [27], [17]. For example, Cerebras WSE2 has 40GB on-chip SRAM over a wafer-scale chip [4].

Meanwhile, SRAM is more error-vulnerable than logic cells, which have temporal masking and logical masking [31]. Therefore, functional safety standards for autonomous vehicles and industrial robotics also require very strong SRAM reliability against soft errors (ISO 26262 [25] and IEC 61508 [24]). In addition, computers in real-world systems are subject to elevated temperatures, and they often operate at low voltages to reduce energy usage; these factors can increase the soft error rate in SRAMs [41]. Combined with the large chip area, the higher vulnerability makes SRAM a dominant threat to DNN processing.



(a) DNN reliability against double adjacent errors

(b) Reliability of layer- vs. channel-wise quantization

Fig. 1: DNN top-1 accuracy degradation with memory errors.

Although most DNN SRAM reliability studies focus on single-bit errors, protection against multi-bit errors is increasingly important. Shrinking process technology and decreasing supply voltage reduces the amount of charges to upset a memory cell so that a high-energy particle strike can affect multiple nearby cells, a symptom known as *Multi-cell Upset (MCU)*. In a recent 5nm FinFET technology, 25% of alpha particle events and 24% of terrestrial neutron events are MCUs [36]. And the ratio increases to 78% and 93%, as neutrons' energy increases to 21 MeV and 85 MeV, respectively. Other experiments present similar or higher MCU ratios [33], [45], [35], [9], [34]. An MCU can affect multiple bits in a word, or spread errors across different words; the former is called as *Multi-bit Upset (MBU)* [1].

The cost of ignoring MBUs in DNN protection can be severe. Errors can corrupt the upper bits of DNN data to change the value significantly. To evaluate the cost, we conducted a DNN accuracy evaluation with DAE injections. We assume all single errors are correctable and inject DAEs only with a uniform random distribution and varying ratios. Errors in activation or weight can affect DNN reliability. However, prior work reports that weights are less error-tolerant than activations because new inputs continuously update activations while weights are reused many times [31], [39]. Therefore, we inject DAEs exclusively into the weights. The result (Fig. 1a) shows that the accuracy is significantly affected by DAEs at a relatively modest ratio of 10^{-8}.

B. DNNs and Quantization

DNN inference is known to be resilient to small noises. Quantization [11] exploits this resilience and converts DNN data in real values into low bit-width numbers, significantly reducing memory usage and computational costs. A popular quantization scheme is the uniform quantization with scaling [26]. It uses a scale parameter to divide a given range of real values into integer values and a zero-point parameter to specify the offset. These parameters are extracted from the data set to avoid mapping to unused values.

There are two ways of applying uniform quantization: layer-wise quantization applies to all weights of each layer, whereas channel-wise quantization applies to all weights of each filter. Channel-wise quantization has different scales and zero-point parameters for each filter and can provide better data resolution and DNN accuracy. Table I compares DNN

TABLE I: Accuracy degradation of quantization methods

Model	Baseline (FP32)	8bit		6bit	
		Channel	Layer	Channel	Layer
ResNet50	76.13%	(-0.03%)	(-0.12%)	(-0.41%)	(-3.00%)
MobileNet V2	71.88%	(-0.79%)	(-1.66%)	(-2.47%)	(-14.97%)
RegNet-X-8GF	79.34%	(+0.04%)	(-0.05%)	(-0.17%)	(-2.00%)
Average		(-0.26%)	(-0.61%)	(-1.02%)	(-6.66%)

accuracy with different quantization schemes. It shows that channel-wise quantization can provide better accuracy with smaller quantization errors, and the accuracy gap increases with more aggressive quantization (i.e., 6-bit quantization).

Figure 1b shows the ResNet50 [21] classification errors with varying bit error ratios. In this experiment, we assume quantization parameters are stored reliably and inject errors only into quantized weights. The result shows that channel-wise quantization is more robust than layer-wise quantization because it has a smaller range for quantized weights, and an error cannot change the value outside the range. Therefore, Q+PoP utilizes channel-wise quantization to improve accuracy and reliability.

III. Prior Works

A. Error Correction Codes

Error Correction Codes (ECCs) can correct errors to enhance memory reliability. Most memory subsystems utilize linear block codes with simpler decoding to reduce read latency. Linear block codes can be classified based on their symbol size: binary ECC and non-binary ECC. Binary ECC codes correct errors on a bit basis. The most well-known binary ECC is Hamming code [20]. It can correct the one-bit error (i.e., Single Error Correction (SEC)) in n-bit codewords using $\lceil log_2(n+1) \rceil$ redundant bits. Single Error Correction - Double Error Detection (SEC-DED) codes extend SEC by adding one-bit parity. Binary BCH codes [3] are other popular binary ECC schemes. BCH codes can efficiently correct random multi-bit errors. For n-bit codewords, BCH codes require $t \times \lceil log_2(n+1) \rceil$ redundant bits to correct t-bit errors.

Non-binary ECC codes can correct errors with a symbol basis. Among non-binary symbol ECCs, Reed Solomon (RS) codes are popular in memory ECC [40]. RS codes can correct t symbol errors using $2t$ redundant symbols if the codeword is no bigger than $2^m - 1$ symbols, where m is the symbol size. Although RS codes can correct multiple erroneous bits within a symbol within a single correction, which is good at burst errors, they are relatively inefficient in correcting randomly distributed bit errors.

Some prior work proposed applying ECC in multiple levels. Methods such as LOT-ECC [49], RATT-ECC [5], and Configurable-ECC [6] employ two-tiered ECC schemes for DRAM. In these approaches, errors that cannot be corrected by the first tier are addressed by the second tier, which distributes redundancy across multiple tier-1 blocks to minimize overhead.

B. DNN Reliability Enhancement via ECC

Many studies proposed weight protection schemes to enhance DNN reliability. The simplest way of protecting the weights is applying ECC to the weights and storing the redundant check bits in extra storage. However, SRAMs occupy a large area in DNN processors, and reducing the protection cost is one of the main challenges for DNN protection. Quantization is the most effective technique for eliminating protection storage costs, leveraging the inherent resilience of DNNs.

Weight Nulling [37] replaces the least significant bit of a half-precision floating-point weight with one-bit parity to detect errors. A weight with a detected error is regarded as zero and does not participate in computation. Therefore, it cannot protect against a high error ratio as more errors nullify more weights, potentially leading to degraded accuracy. Moreover, multi-bit errors within a weight can escape the parity check, leading to uncorrected weight corruption.

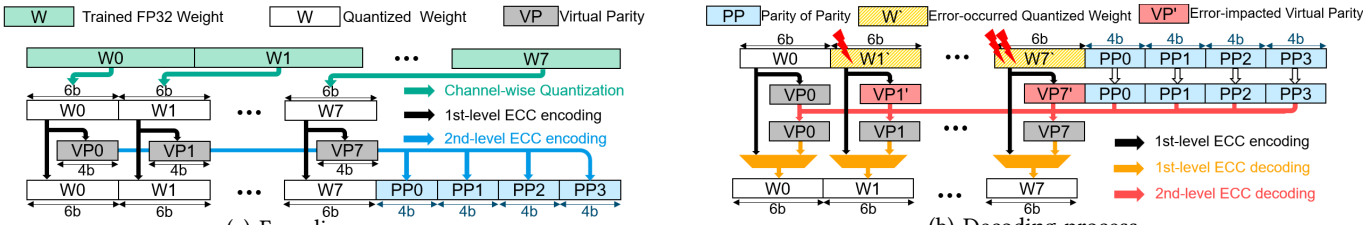
In-place Zero-space ECC [12] utilizes that most weights have small absolute values. It first retrains DNN models with much smaller weights near zero, allowing efficient representation with 7-bit quantized values. The remaining non-informative 1-bit is used for redundancy. Then, eight 7-bit quantized weights are grouped and encoded with a (64, 57) SEC-DED code to form a 64-bit block. This can provide correction for single-bit errors, while many soft errors are multi-bit upsets (MBUs). Additionally, retraining requires significant computation and may impact accuracy.

Value-aware Parity Insertion ECC [29] is one of the strongest DNN protection schemes. Similar to [12], it leverages a narrow weight distribution for aggressive quantization and redundancy storage. Weights are represented using a custom 1-bit sign and 7-bit magnitude format, assuming that most weights fall below 0.5. It allows the two most significant bits to be used for storing parity. If weights fall outside the range of $[-0.5, 0.5]$, the two least significant bits are truncated and used for parity. This scheme provides Double Error Correction (DEC) to 64-bit blocks, improving error tolerance. However, it may reduce DNN accuracy, particularly when weights fall outside the expected range or due to the loss of weight values. Moreover, its DEC capabilities may be insufficient in a higher error ratio of MBUs.

IV. Parities of Parities ECC (PoP-ECC)

PoP-ECC is a strong and flexible scheme to protect data from severe memory errors. In designing PoP-ECC, we aim to protect data from memory MBUs with minimal costs. PoP-ECC is a two-level ECC; the first level encoding extracts *Virtual Parities (VPs)* from each value, and the second level protects VPs using *Parities of Parities (PPs)*. VPs are used for the second-level parity generation only and are not stored in memory. One of the key differences of PoP-ECC is that it protects virtual data, called VP, whereas prior works protect raw data. Extracting a VP from raw data is flexible and can be customized to the target MBUs and applications.

PoP-ECC is a general method and operates efficiently across various precisions and sizes. Without loss of generality, in Q+PoP, we use a lower-precision representation for



(a) Encoding process (b) Decoding process

Fig. 2: An overview of Q+PoP

clarity and consistency with prior works, reducing from 8-bit to 6-bit, to minimize protection costs and allocate space for PoP-ECC redundancy. Using 6-bit precision, 16 bits are saved per 64-bit block, allowing Q+PoP to utilize the saved space for storing redundant protection bits. We choose VP as a 4-bit parity to correct all single-/double-bit errors on the upper 5 bits of a quantized value; we do not protect the least significant bit because errors in the bit position are not critical to accuracy in our evaluation.

A. Encoding Process

Figure 2a illustrates the encoding process of PoP-ECC within the Q+PoP framework. Prior to encoding, data precision is reduced to 6-bit integers using channel-wise quantization. The encoding process in PoP-ECC operates in two stages. In the first stage, a 4-bit Virtual Parity (VP) is generated from the 6-bit quantized data. This stage allows for flexible customization of the ECC encoding scheme to address varying MBU severity and patterns. The generated 4-bit VP is designed to correct all single- and double-bit errors in the upper 5 bits of the quantized weight. Notably, the lowest bit is left unprotected, as experiments reveal that errors in this position have negligible impact on accuracy.

The upper 5 bits can have five single-bit errors and ten ($=_5C_2$) double-bit errors, and the 4-bit VP uses its 15 non-zero syndromes to correct them. The encoding utilizes the 5×9 generator matrix to generate a 4-bit VP from the upper 5-bits of a quantized weight. It generates unique syndromes for each single- and double-bit error on the weight. The parity check matrix (4×9) has identical columns in the data and parity parts, which is unusual. However, the virtually stored parities are error-free and cannot generate a syndrome the same as data errors. It can use a table (optimized during synthesis) to locate errors from the syndromes and restore the weight. VP generation increases the encoding and decoding latency slightly with only one level of XOR gates.

In the second stage, PoP-ECC utilizes RS code to protect VPs on a symbol basis. Because each VP is 4-bit, it applies 4-bit RS encoding to the eight VPs to generate four redundant symbols (i.e., PPs) and stores the data and PPs. It is worth mentioning that PoP-ECC differs from concatenated codes [10] in that its second-level ECC encodes VPs only and does not directly protect data. Moreover, it does not store the VP. The total block size remains 64-bit ($8 \times \{$6-bit quantized weights$\}$ + $4 \times \{$4-bit PPs$\}$), whereas the symbol count is 12 (8 VPs and 4 PPs).

B. Decoding Process

Figure 2b illustrates the decoding process of PoP-ECC, with some errors on the stored block. In the example, protected data, W1 and W7, have a single- and double-error, respectively. On receiving the erroneous block, PoP-ECC first regenerates VPs using the same generator matrix as the encoding (i.e., first-level ECC of PoP-ECC). The errors in this example belong to the upper 5 bits of weights, and some regenerated VPs (VP1 and VP7) differ from the originals in the encoding. The second-level ECC applies the RS decoding on VPs and PPs, and it can restore the two corrupted VPs using two symbol corrections. After restoring the VPs, PoP-ECC utilizes the VPs to restore the weights; VP1 can correct the single error on W1, while VP7 can correct the double error on W7. The VP regeneration and the VP-based correction may affect the decoding latency, yet our evaluation shows they do not increase the latency significantly ($\leq 0.18ns$ in UMC 28nm process, Section V-D).

Without an error, the regenerated VPs will be the same as the original VPs, leaving no correction by VPs and PPs. Similarly, if errors corrupt the lowest bits of data only, the regenerated VPs will be the same as the original VP (the bit does not participate in VP generation), and the errors will be neither detected nor corrected. However, our experiments show small noises from these errors do not significantly affect accuracy. Compared to the selective error correction on quantized weights, PoP-ECC can correct any number of errors on a PP using the symbol-based correction unless the number of erroneous weights and erroneous PPs exceed its correction capability (i.e., 2).

In addition to stronger protection, PoP-ECC can provide flexible protection. If some error patterns are dominant (e.g., DBU) or some bits are more/less critical (e.g., the least significant bit is non-critical), it can tailor the VP generation scheme. Moreover, PoP-ECC can control the number of PPs (e.g., 2 PPs for single symbol correction and 4 PPs for double symbol correction) to deal with different error rates.

V. Evaluation

We evaluate the accuracy and reliability of Q+PoP, which integrates PoP-ECC, and compare its performance to state-of-the-art schemes. The results demonstrate that Q+PoP, leveraging channel-wise quantization, achieves superior base accuracy, ensuring minimal accuracy loss in error-free scenarios while enhancing reliability. In terms of fault tolerance, Q+PoP can withstand significantly higher ratios of single-bit

TABLE II: Accuracy degradation of the protection schemes

Model	Baseline (FP32)	Weight Nulling (FP16)	VAPI (8b Sign/Mag. Q.)	Q+PoP (6b Ch-wise Q.)
ResNet50	76.13%	(0.00%)	(-4.25%)	(-0.41%)
MobileNet V2	71.88%	(-0.01%)	(-16.07%)	(-2.47%)
RegNet-X-8GF	79.34%	(+0.01%)	(-2.01%)	(-0.17%)
Average		(0.00%)	(-7.44%)	(-1.02%)

errors and DAEs with stronger correction and error-robust quantization. Despite these advancements in accuracy and reliability, Q+PoP incurs minimal hardware overhead, thanks to its use of small symbols.

A. Methodology

We implemented Q+PoP and two state-of-the-art DNN protection schemes (Weight Nulling and Value-aware Parity Insertion (VAPI)) in the PyTorch framework. We do not include In-place Zero-space ECC because it provides weak protection (SEC-DED) and requires retraining the model. All evaluated schemes have 64-bit block sizes. Weight Nulling has four FP16 weights in the block and replaces the least significant bits with one-bit parity to detect errors in the weight. VAPI has eight weights in the 1-bit sign and 7-bit magnitude format and utilizes two magnitude bits to store ECC redundancy.

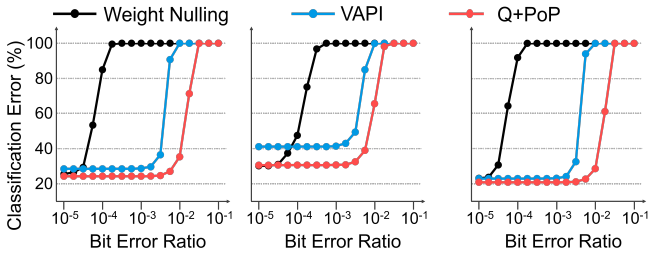
We use three representative DNN models (ResNet50 [21], MobileNet V2 [43], and RegNetX-8GF [38]) with Torchvision pretrained weights. The input dataset is ILSVRC-2012 [42], and we measure top-1 classification accuracy over 50,000 validation images. To measure the accuracy, we run the models with quantized weights in FP32 format. We map quantized weights to their closest FP32 values. This provides a common execution environment across different data types (i.e., FP16 in Weight Nulling, sign/magnitude in VAPI, and INT6 in Q+PoP), while the high precision (24-bit significand in FP32) minimizes conversion losses down to 2^{-24} of the original values and minimizes its impact on accuracy.

B. Accuracy

We first run the DNN models without errors and measure the inference accuracy. All evaluated schemes apply aggressive quantization to create room for redundancy. To quantify the accuracy cost of the quantization, we measure DNN accuracy without injecting errors.

Table II compares the inference accuracy without errors. Weight Nulling provides almost the same accuracy as the pretrained weights because replacing the least significant bit of FP16 with one-bit parity does not affect accuracy significantly. VAPI sacrifices accuracy significantly (up to 16.07% in MobileNet) in our evaluation.

Meanwhile, Q+PoP shows an average of 1.02% degradation. Despite aggressive 6-bit quantization, its channel-wise quantization can flexibly deal with diverse weight scales. 2 out of 3 models have $< 0.5\%$ degradation, while MobileNet V2 shows the biggest degradation of 2.47%. MobileNet is a depthwise-separable CNN, reported as more difficult to quantize than other CNNs [51].



(a) ResNet50 (b) MobileNet V2 (c) RegNetX-8GF

Fig. 3: DNN reliability comparison against varying single-bit error ratios.

C. Reliability

Then we run the DNN models with errors to evaluate reliability. We inject two types of errors—Single Error (SE) and DAE, representing Single-Bit Upsets and MBUs, respectively. Errors are applied only to weight blocks, as weights are crucial to DNN reliability [39], [29]. Each target block is randomly selected, formatted to the protection scheme's layout, encoded with ECC, and parity bits are stored in designated positions. Errors are then injected at random positions within the block, including parity bits, simulating real-world scenarios.

After injecting errors, the ECC decoding process is applied to the block. Successful correction restores the quantized weight, while miscorrection worsens the error by introducing new ones. Detected but uncorrectable errors leave the weight unchanged, except in cases of Weight Nulling, which nullifies it. Decoded weights are converted back to FP32, with uncorrected weight errors altering values based on the scheme's format. Errors in parity are ignored to mimic real scenarios. The process is repeated 50 times with various error injections, and the average classification error is calculated.

1) Reliability against Single-Errors: Figure 3 presents the results of the SE injection experiments. We sweep the Bit Error Ratio (BER) from 10^{-5} to 10^{-1} and measure the inference accuracy.

For lower BERs (e.g., $< 10^{-5}$), the accuracy results are almost identical to those without errors in Table II. At these relatively low ratios, Q+PoP can provide significantly higher accuracy than VAPI (-1.02% vs. -7.44% accuracy loss) with its channel-wise quantization. Weight Nulling provides almost no accuracy loss. However, it is not energy-efficient due to its doubled memory footprint (16-bit vs. 8-bit per weight) and energy-expensive floating-point operations (FP16).

The accuracy of Weight Nulling decreases quickly as the ratio approaches 10^{-4}. At the ratio, 1.6% of FP16 weights have single-bit errors and are nullified. Moreover, the accuracy quickly approaches 0% as the ratio increases. Increasing errors not only nullifies more weights with single errors but also generates undetectable errors from double errors on a weight. The one-bit parity cannot detect an even number of errors and degrades accuracy significantly at a high BER or a relatively low DAE ratio (Section V-C2).

Q+PoP can tolerate the highest BER of up to near 3×10^{-3}.

TABLE III: The maximum BERs which do not reduce DNN accuracy by 0.1 percentage point than error-free cases.

Model	VAPI	Q+PoP	Ratio
ResNet50	$10^{-3.2}$	$10^{-2.8}$	2.51×
MobileNet V2	$10^{-3.2}$	$10^{-3.1}$	1.26×
RegNet-X-8GF	$10^{-3.2}$	$10^{-2.7}$	3.16×

TABLE IV: The maximum DAE ratios which do not reduce DNN accuracy by 0.1 percentage point than error-free cases.

Model	VAPI	Q+PoP	Ratio
ResNet50	$10^{-4.2}$	$10^{-3.1}$	12.59×
MobileNet V2	$10^{-4.3}$	$10^{-3.4}$	7.94×
RegNet-X-8GF	$10^{-4.5}$	$10^{-3.0}$	31.62×

It can correct not only single and double errors but also parts of triple and quadruple errors as long as they belong to two quantized weights, and each weight has up to 2 errors. The stronger correction allows Q+PoP to tolerate higher error ratios than VAPI with DEC, and the channel-size quantization minimizes accuracy degradation from uncorrectable errors. To quantify the improved reliability, we checked error ratios at which the accuracy degraded by 0.1 percentage point than no error cases. Table III shows Q+PoP can tolerate up to 3.16× higher BERs than VAPI.

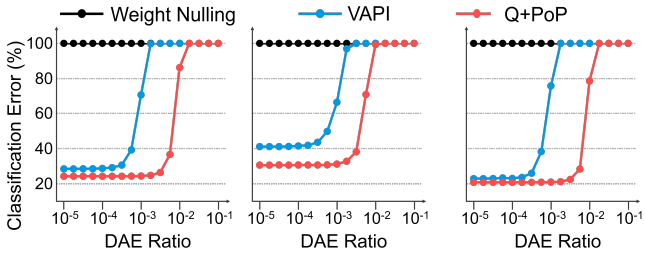
2) Reliability against Double-Adjacent Errors: Figure 4 presents the results of DAE injection experiments. Like the SE results, Q+PoP is significantly more robust than Weight Nulling and VAPI. Weight Nulling cannot detect double-bit errors within a weight and provides protection as bad as no protection against DAEs (almost 0% accuracy at 10^{-5} and higher ratios).

VAPI can correct one DAE using its DEC BCH codes to tolerate up to 8×10^{-5}. However, at 10^{-3}, the chances of having two DAEs within a 64-bit block increase to 6.3%, and the accuracy decreases to near 0%. Using the same amount of redundancy, PoP-ECC can tolerate up to 10^{-3} by correcting up to two DAEs within a 64-bit block. Combined with better base accuracy, the less accuracy degradation make Q+PoP a accurate and strong solution for reliable DNN processing. Table IV compares the maximum DAE ratio, which does not reduce DNN accuracy by 0.1 percentage point than error-free cases. It shows Q+PoP can tolerate up to 31.62× higher DAE ratios than VAPI.

Overall, Weight Nulling provides the highest baseline accuracy, yet the accuracy is based on double memory footprint (16-bit) and energy-expensive floating point operations. VAPI can tolerate relatively-high error ratios (e.g., $10^{-3.2}$ BER and $10^{-4.2}$ DAE ratio), yet the baseline accuracy can be poor due to its inflexible quantization (e.g., up to 16.07% accuracy loss, Table II). Q+PoP provides both high accuracy and reliability with 1.02% baseline accuracy degradation, which is comparable to INT8 quantization schemes. And it maintains the fair accuracy up to very high error ratios.

D. Hardware Costs

To evaluate the hardware overheads of PoP-ECC, we implemented its encoder and decoder in Verilog. The decoder follows an algorithm in [44] for faster decoding. We synthesize the Verilog models using Synopsys Design Compiler and UMC 28nm SVT library.



(a) ResNet50 (b) MobileNet V2 (c) RegNetX-8GF

Fig. 4: DNN reliability comparison against varying Double-Adjacent Error (DAE) ratios.

The encoding takes $0.65ns$, which is small enough to be implemented in 1 cycle in modern technology nodes. At a deeper look, VP generation requires only one level of 2-input XOR gates and takes $0.22ns$. PP generation mostly comprises XOR trees and takes the remaining $0.43ns$. The decoding takes $1.56ns$, which can be split into error detection and error correction. The detection part is similar to the encoding and takes $0.8ns$. We expect this will not increase the read latency than SEC-DED. The correction part takes an additional $0.8ns$, which can be skipped in no error cases.

The area overhead is $92\mu m^2$ (0.3K NAND2 gates) for the encoder and $1760\mu m^2$ (6K NAND2 gates) for the decoder. Most of the increase in the decoder comes from the RS decoder. The power consumption of the encoder and decoder are 0.08mW and 1.2mW, respectively.

VI. Conclusion

This paper introduces PoP-ECC, a novel two-level ECC scheme designed to address growing reliability challenges in SRAMs. By leveraging Virtual Parities and Parities of Parities, PoP-ECC provides a robust and flexible solution for mitigating MBUs. Building on the strengths of PoP-ECC and channel-wise quantization, we also propose Q+PoP, a robust and efficient DNN protection scheme. Our evaluation demonstrates that Q+PoP achieves better base accuracy and can tolerate up to 31.62× higher DAE ratios compared to state-of-the-art schemes, all with negligible overhead in delay, area, and power.

Acknowledgment

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2019-II190421, Artificial Intelligence Graduate School Program (Sungkyunkwan University), 50%), and the Technology Innovation Program (or Industrial Strategic Technology Development Program-Public-Private Joint Investment Advanced Semiconductor Talent Development Project) (RS-2023-00237136, Development of CXL/DDR5-based memory subsystem for AI accelerators, 50%).

References

[1] *Measurement and Reporting of Alpha Particle and Terrestrial Cosmic Ray-Induced Soft Errors in Semiconductor Devices*, JESD89B, Joint Electronic Device Engineering Council, 2006.

[2] M. Borg *et al.*, "Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry," *Journal of Automotive Software Engineering*, vol. 1, no. 1, 2019.

[3] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and control*, vol. 3, no. 1, 1960.

[4] Cerebras Systems, "The Future of AI is Here," https://www.cerebras.net/product-chip/, 2023.

[5] H.-M. Chen *et al.*, "RATT-ECC: Rate Adaptive Two-Tiered Error Correction Codes for Reliable 3D Die-Stacked Memory," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 13, no. 3, 2016.

[6] H.-M. Chen, S.-Y. Lee, T. Mudge, C.-J. Wu, and C. Chakrabarti, "Configurable-ecc: Architecting a flexible ecc scheme to support different sized accesses in high bandwidth memory systems," *IEEE Transactions on Computers*, vol. 68, no. 5, 2019.

[7] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, 2017.

[8] M. Demler, "Hailo Illuminates Low-Power AI Chip," The Linley Group Microprocessor Report from June 24, 2019.

[9] V.-P. Fang and A. Oates, "Cell Level Soft Error Rate Simulations on Planar and FinFET Processes," in *Proceedings of the CPMT SER Workshop*, 2014.

[10] G. D. Forney, "Concatenated Codes," Massachusetts Institute of Technology, Tech. Rep. 440, 1965.

[11] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[12] H. Guan *et al.*, "In-place zero-space memory protection for CNN," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NIPS)*, 2019.

[13] L. Gwennap, "Graphcore Makes Big AI Splash," 2018, The Linley Group Microprocessor Report from September 17, 2018.

[14] ——, "Habana Wins Cigar for AI Inference," 2018, The Linley Group Microprocessor Report from February 18, 2018.

[15] ——, "Alibaba Takes AI Performance Lead," 2019, The Linley Group Microprocessor Report from November 25, 2019.

[16] ——, "Graphcore Doubles IPU Performance," 2020, The Linley Group Microprocessor Report from August 3, 2020.

[17] ——, "Groq Rocks Neural Networks," 2020, The Linley Group Microprocessor Report from January 6, 2020.

[18] ——, "Tenstorrent Scales AI Performance," 2020, The Linley Group Microprocessor Report from April 13, 2020.

[19] ——, "Quadric's Flexible Edge Accelerator," 2021, The Linley Group Microprocessor Report from July 5, 2021.

[20] R. W. Hamming, "Error detecting and error correcting codes," *The Bell system technical journal*, vol. 29, no. 2, 1950.

[21] K. He *et al.*, "Deep residual learning for image recognition," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2015.

[22] M. Horowitz, "Energy Table for 45nm Process," in *Stanford VLSI Wiki*, 2014.

[23] S. Hosseininoorbin *et al.*, "Exploring edge tpu for deep feed-forward neural networks," *Internet of Things*, vol. 22, 2023.

[24] International Electrotechnical Commission, "Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems," https://webstore.iec.ch/publication/22273, 2010.

[25] International Organization for Standardization, "ISO 26262:2018 Road Vehicles—Functional Safety," https://www.iso.org/standard/68383.html, 2018.

[26] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2018.

[27] A. Jani, "Mythic Unveils Analog-AI Product," 2020, The Linley Group Microprocessor Report from November 30, 2020.

[28] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[29] S.-S. Lee and J.-S. Yang, "Value-Aware Parity Insertion ECC for Fault-Tolerant Deep Neural Network," in *Proceedings of the Conference on Design, Automation, and Test in Europe (DATE)*, 2022.

[30] S. Levine *et al.*, "End-to-End Training of Deep Visuomotor Policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, 2016.

[31] G. Li *et al.*, "Understanding Error Propagation in Deep Learning Neural Network (DNN) Accelerators and Applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2017.

[32] R. K. Maity, J. Samanta, and J. Bhaumik, "Single and double-adjacent error correcting code (SDECC) with lower design overheads and miscorrection rate for SRAMs," *Microsystem Technologies*, 2023.

[33] D. G. Mavis *et al.*, "Multiple bit upsets and error mitigation in ultra-deep submicron srams," *IEEE Transactions on Nuclear Science*, vol. 55, no. 6, 2008.

[34] B. Narasimham *et al.*, "Scaling Trends and Bias Dependence of the Soft Error Rate of 16 nm and 7 nm FinFET SRAMs," in *Proceedings of the International Reliability Physics Symposium (IRPS)*, 2018.

[35] A. Neale and M. Sachdev, "Neutron Radiation Induced Soft Error Rates for an Adjacent-ECC Protected SRAM in 28nm CMOS," *IEEE Transactions on Nuclear Science*, vol. 63, no. 3, 2016.

[36] N. J. Pieper, Y. Xiong, A. Feeley, J. Pasternak, N. Dodds, D. R. Ball, and B. L. Bhuva, "Study of multi-cell upsets in sram at a 5-nm bulk finfet node," *IEEE Transactions on Nuclear Science*, 2023.

[37] M. Qin *et al.*, "Robustness of Neural Networks against Storage Media Errors," 2017. [Online]. Available: https://arxiv.org/abs/1709.06173

[38] I. Radosavovic *et al.*, "Designing network design spaces," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2020.

[39] B. Reagen *et al.*, "Ares: A Framework for Quantifying the Resilience of Deep Neural Networks," in *Proceedings of the Design Automation Conference (DAC)*, 2018.

[40] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the society for industrial and applied mathematics*, vol. 8, no. 2, 1960.

[41] M. Riera, R. Canal, J. Abella, and A. Gonzalez, "A Detailed Methodology to Compute Soft Error Rates in Advanced Technologies," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016.

[42] O. Russakovsky *et al.*, "Imagenet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, 2015.

[43] M. Sandler *et al.*, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2018.

[44] D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for reed-solomon decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 5, 2001.

[45] N. Seifert, B. Gill, S. Jahinuzzaman, J. Basile, V. Ambrose, Q. Shi, R. Allmon, and A. Bramnik, "Soft Error Susceptibilities of 22 nm Tri-Gate Devices," *IEEE Transactions on Nuclear Science*, vol. 59, no. 6, 2012.

[46] M. Shirobokov, S. Trofimov, and M. Ovchinnikov, "Survey of machine learning techniques in spacecraft control design," *Acta Astronautica*, vol. 186, 2021.

[47] V. Sridharan *et al.*, "Feng shui of supercomputer memory positional effects in dram and sram faults," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2013.

[48] ——, "Memory errors in modern systems: The good, the bad, and the ugly," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, 2015.

[49] A. N. Udipi *et al.*, "LOT-ECC: Localized and Tiered Reliability Mechanisms for Commodity Memory Systems," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012.

[50] B. Wheeler, "Flex Logix Shows First AI Chip," The Linley Group Microprocessor Report from November 16, 2020.

[51] S. Yun and A. Wong, "Do All MobileNets Quantize Poorly? Gaining Insights into the Effect of Quantization on Depthwise Separable Convolutional Networks Through the Eyes of Multi-scale Distributional Dynamics," in *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 2021.

Principle-based Dataflow Optimization for Communication Lower Bound in Operator-Fused Tensor Accelerator

Lei Xu, Chen Yin, Zelong Yuan, Weiguang Sheng, Jianfei Jiang, Qin Wang, Naifeng Jing*
Shanghai Jiao Tong University, Shanghai, China
xlxl1027@sjtu.edu.cn, sjtuj@sjtu.edu.cn

***Abstract*—Although design space exploration (DSE) is good at finding dataflow for optimal memory access in tensor accelerators, it is very timing-consuming and lacks architecture insight. In this study, we for the first time propose several principles for dataflow optimization that provides lower bound of memory communication for tensor operators such as matrix multiplication. Through these principles we can calculate the best tiling, scheduling and mapping for both intra- and inter-operator dataflow. In addition, we can identify all the tensor-wise opertor fusion that are profitable in memory communication, so we propose FuseCU, a new architecture that supports these profitable fusion which can be applied to existing spatial architectures for data movement saving. Experimental results show that FuseCU delivers 63.6%, 62.4% and 38.7% data movement saving and 1.33×, 1.25× and 1.14× speedup compared to the TPUv4i, Gemmini and Planaria designs without increasing buffer size or bandwidth. Additionally, FuseCU is open-sourced.**

***Index Terms*—dataflow, tensor, principle, fusion, processing elements.**

I. Introduction

Memory access is becoming a major bottleneck in performance and a key factor in the energy consumption of tensor applications like Transformers. To address this challenge, various dataflow optimization techniques have been proposed. Prior research has largely focused on intra-operator dataflow, where data movement is optimized within a single tensor operator, especially in convolution and matrix multiplication (MM) operators [1]–[9]. More recently, researchers are shifting towards inter-operator dataflow—also known as operator fusion—to exploit data reuse across operator boundaries [10]–[15]. Unlike intra-operator dataflow, which repeatedly accesses memory across operator boundaries as illustrated in Fig. 1(a), inter-operator dataflow can enhance data reuse, and potentially eliminate memory communication for intermediate data in an ideal case as in Fig. 1(b).

However, achieving optimal dataflow is nontrivial. Dataflow should taking tiling, scheduling and hardware mapping into overall consideration, and thus create a vast design space. Classic dataflow optimization largely relies on searching-based design space exploration (DSE). However, there are two problems with searching-based optimization:

1) It is time-consuming. If fusion is introduced, the design space expands from a single operator to multiple operators, possibly making search time even unacceptable [14], [15].
2) It lacks architecture insight. As a black box optimization, [11], [13]–[15], searching-based optimization sheds limited insight on architecture innovations.

In this paper, we propose a principle-based dataflow optimization that provides lower bounds for memory communication. It gives optimal dataflow analytically in one-shot. More importantly, these principles offer us new architecture and dataflow insights, so we can propose an efficient operator-fused tensor accelerator called FuseCU, which supports all profitable fusion on tensor operators. To the best of our knowledge, this is the first work to derive the optimal inter-operator dataflow through a principle-based analytical method and the first to unlock the full potential of tensor operator fusion on compute units through architecture innovation. Our key contributions are summarized as follows:

- We propose principle-based dataflow optimization achieving the lower bound for memory communication. These principles are derived from analyzing various dataflow strategies under different tensor and buffer sizes, and directly tell us an efficient dataflow on tensor operators like MM.
- We apply these principles on tensor operator fusion, which guides us to conclude several but all the profitable tensor-wise operator fusion regarding memory communication. This provides us with insights into enabling efficient fusion on compute units through effective mapping and architecture innovations.
- We propose FuseCU, a novel architecture which has sufficient flexibility to support all the profitable fused dataflow. Operators like fused MM can be mapped on a number of PEs without adding any buffers or registers to existing spatial architectures.

We validate our principle-based optimization against the state-of-the-art searching-based dataflow optimizer DAT [15], build FuseCU on TPUv4i [5] and evaluate them using seven attention-based models, which consist of different-sized MM operators. The experimental results show that FuseCU saves 63.6%, 62.4%, and 38.7% data movement compared to TPUv4i [5] and SOTA designs [16], [17], and 1.33×, 1.25×, and 1.14× speedup on average, respectively.

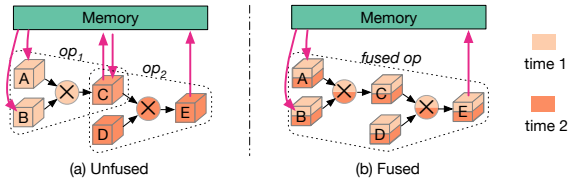


Fig. 1. Operator fusion further reduces memory traffic than unfused operators.

II. Previous Works and Background

A. Dataflow Definition

Dataflow mainly consists of three key parts: tiling, scheduling, and mapping.

- **Tiling** determines the size of the tiles of tensors held in the buffer. As shown in Fig. 2(a), tiling stores only a portion of the whole tensor in the buffer. This tile can aviod memory access by keep data locally. By carefully choosing the tile size of each tensor, better tiling can increase data reuse in buffers.
- **Scheduling** determines the order of tile computations to further exploit data reuse. As shown in Fig. 2(b), different loop orders lead to different tile movements or computation sequences. Memory access can be minimized by keeping unchanged tiles in the buffer. In existing works, this is referred to as the "*stationary*", where unchanged tensor tiles between adjacent computations remain in the buffer.
- **Mapping** determines the assignment of the tiles to the underlying hardware resources. As shown in Fig. 2(c), this involves

*This work is supported by the National Natural Science Foundation of China under Grant No.U24B20165 and No.92264108. Naifeng Jing is the corresponding author.

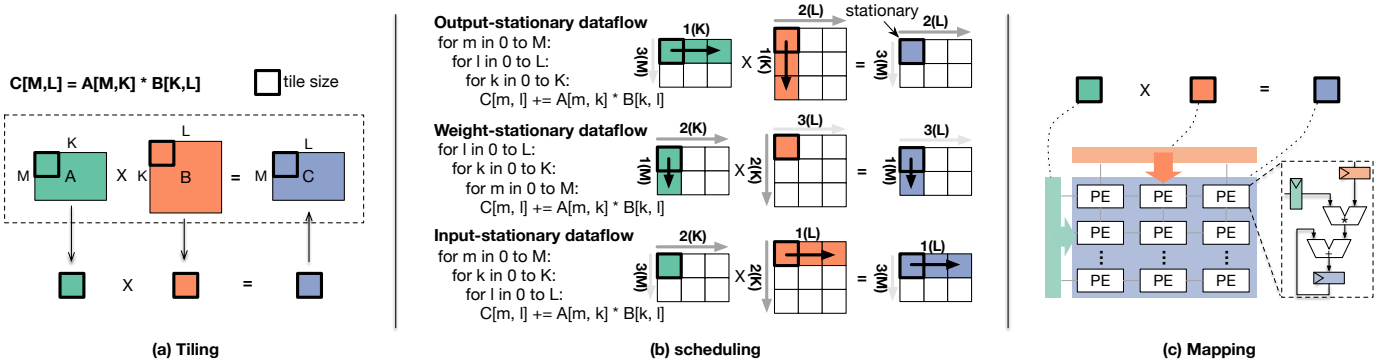


Fig. 2. Three key parts in dataflow for operator $A \times B = C$. We use term X(D) to denote the *scheduling* by loop order "X" on dimension "D" as in the left code. For example in output-stationary (OS), "1(K)" implies the innermost loop on dimension K. We use the arrows to indicate tile movement.

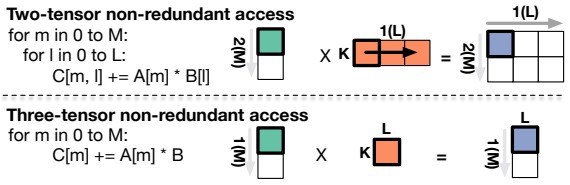


Fig. 3. Two-NRA and Three-NRA dataflow for larger buffer size.

assigning tiles to processing element (PE) array ports and registers, with specific options tailored to the hardware. In this example, C is mapped to registers, while A and B are mapped to the ports, requiring dedicated wiring in the PE.

In practice, the tiling and scheduling optimize the communication between memory and buffer, while mapping optimizes the communication between buffer and PE.

B. Dataflow Optimization

Table I provides a summary of SOTA dataflow optimization frameworks. For intra-operator dataflow, they typically leverage searching-based DSE to optimize tiling, scheduling, and mapping jointly, as each of these aspects impacts the others [1]–[4], [7]. For inter-operator dataflow, several works have proposed tensor operator fusion within memory and employed DSE to select optimal dataflow [11], [12], [18]. Due to the extensive design space, they often separate the mapping phase from tiling and scheduling, or work with incomplete spaces. Additionally, existing works focus on tensor operator fusion on memory without addressing fusion across compute units or PEs, as mapping onto the PE array is hardware-dependent. In summary, existing works generally rely on searching for optimization and overlook compute-level fusion.

III. Principles for Dataflow Optimization

In this section, we propose principles for dataflow optimization, which guild us to select the best dataflow regarding tiling and scheduling for the least memory access (MA), given the specific tensor size of the operator and the buffer size (BS) of the hardware. The principles also guide the best mapping methods in Sec. IV-A. We will use operator MM as an example.

A. Intra-operator Dataflow

For MM, there are three distinct BS that influence optimal dataflow. Ideally, if BS were infinite, all tensors would be fetched only once, achieving the ideal minimal MA for this operator. However, due to limited BS, additional MA is required on the same data, which is termed as redundant access in this paper and should be minimized. As BS increases, the dataflow transitions from single-tensor non-redundant access to two-tensor non-redundant access, and finally to three-tensor non-redundant access.

*1) Single-tensor Non-redundant Access (**Single-NRA**):* For small BS, only the stationary tensor can avoid redundant access. As shown in Fig. 2(b), the stationary tensor is the only one accessed a single time. Two primary decisions for optimal dataflow in this case are selecting the stationary tensor and determining the tiling strategy.

Taking the output-stationary dataflow in Fig. 2(b) as an example, tensor C is kept in the buffer to minimize redundant memory access during adjacent computations, while tensors A and B are re-accessed. Memory accesses for each tensor are calculated as the product of tile sizes and iteration counts. For each dimension D, let T_D represent the tile size. For tensors A and B, the memory accesses are $MK\frac{L}{T_L}$ and $KL\frac{M}{T_M}$ respectively, and for C, it is ML. Thus, total MA can be expressed as:

$$\text{MA} = MKL \times \left(\frac{1}{T_L} + \frac{1}{T_M}\right) + ML \tag{1}$$

To minimize MA, T_L and T_M, which are stationary tensor dimensions, should be maximized. This requires minimal T_K as the total buffer size is limited. Given the BS constraint:

$$T_M T_K + T_K T_L + T_M T_L \leq \text{BS} \tag{2}$$

Minimum MA is achieved at $T_M = T_L = T$. This analysis applies similarly across other stationary dataflow, yielding the same conclusion. Comparing MAs for different stationary dataflow, as in Eq. 1, the initial term is constant as $MKL \times (2/T)$, so the difference lies in the stationary tensor's MA component. With small BS, MA for the stationary tensor is relatively minor compared to others. But, choosing the smallest tensor as stationary minimizes MA.

Principle 1 *Tiling: maximize tile size for stationary tensor dimensions, minimize for non-stationary ones.*
Scheduling: choose the smallest tensor to be stationary.

*2) Two-tensor Non-redundant Access (**Two-NRA**):* With a larger BS, a second tensor can achieve non-redundant access by not tiling one dimension. As shown in Fig. 3, compared to the output-stationary dataflow in Fig. 2(b), the loop over dimension K is removed by keeping this dimension untiled. Consequently, tensors A (input) and C (output) are both accessed without redundancy.

The MA in this dataflow is:

$$\text{MA} = MKL \times \frac{1}{T_M} + MK + ML \tag{3}$$

TABLE I
SUMMARY OF THE SOTA DATAFLOW OPTIMIZERS.

Features	Intra-operator	Inter-operator				
	[1], [3], [6], [7]	Chimera [12]	SET [13]	Flat [11]	DAT [14], [15]	This Work
Full tiling & scheduling optimization space	×	×	×	×	✓	✓
Tiling & scheduling optimization scheme	Searching-based	Searching-based	Searching-based	Searching-based	Searching-based	Principle-based
Mapping optimization scheme	Searching with fixed patterns	Replaceable micro kernels	Not discussed	Not discussed	Not discussed	Principle-based
Fusion medium	No fusion	Memory	Memory	Memory	Memory	Compute unit

To minimize MA, T_M, which is the dimension not in the redundant access tensor, should be maximized under the buffer size constraint, requiring minimal T_L. The new BS constraint becomes:

$$T_M K + K T_L + T_M T_L \leq \text{BS} \tag{4}$$

The untiled dimension size significantly limits T_M, so choosing the smallest dimension to remain untiled is essential.

Principle 2 *Tiling: maximize the tile size for the dimension not in the redundant access tensor, minimize for others. Scheduling: untile/unroll the smallest dimension.*

*3) Three-tensor Non-redundant Access (**Three-NRA**):* With a sufficiently large buffer, all three tensors achieve non-redundant access by leaving two dimensions untiled. As shown in Fig. 3, only one dimension is tiled, and its tile size does not impact MA. To achieve non-redundant access for all three tensors with minimal buffer size, we should untile the smallest tensor.

Principle 3 *Tiling: do not care. Scheduling: untile/unroll the smallest tensor.*

4) Choosing the Best Dataflow: With the tiling and scheduling principles established for each dataflow, the next step is to determine the optimal dataflow based on the BS compared to the tensor size.

To compare Single-NRA and Two-NRA, we derive the relationship between BS and MA under optimal tiling and scheduling for each dataflow according to our priciples and compare the MAs for the same BS. We omit the analysis details due to limited space. This analysis reveals that when the BS is small, Single-NRA outperforms Two-NRA. However, as BS increases, the optimal dataflow shifts from Single-NRA to Two-NRA. The shift point lies within the range $D^2_{\text{min}}/4$ to $D^2_{\text{min}}/2$, where D_{min} denotes the smallest tensor dimension.

The comparison between Two-NRA and Three-NRA is more straightforward: Three-NRA is preferred when the BS is sufficiently large to accommodate the minimal tensor. At this point, each tensor is accessed only once, achieving the ideal minimal MA for intra-operator dataflow.

Therefore, we classify BS into four categories based on their relation to tensor dimension sizes:

$$\begin{aligned}
&\text{Tiny buffer:} && \text{BS} \leq D^2_{\text{min}}/4 && \Rightarrow \text{Single-NRA}\\
&\text{Small buffer:} && D^2_{\text{min}}/4 < \text{BS} \leq D^2_{\text{min}}/2 && \Rightarrow \text{Single/Two-NRA}\\
&\text{Medium buffer:} && D^2_{\text{min}}/2 < \text{BS} \leq \text{Tensor}_{\text{min}} && \Rightarrow \text{Two-NRA}\\
&\text{Large buffer:} && \text{BS} > \text{Tensor}_{\text{min}} && \Rightarrow \text{Three-NRA}
\end{aligned}$$

Note that for small buffers, both Single-NRA and Two-NRA dataflow can be used, with the optimal one selected based on specific conditions. In all other cases, the best dataflow can be directly chosen based on buffer size and tensor dimension sizes.

Example: Consider the MM of $A_{1024,768} \times B_{768,768} = C_{1024,768}$ in BERT [19] by assuming a BS of 512 KB. Given that BS $> D^2_{\text{min}}/2 = 768^2/2 = 294,912$ and BS $< Tensor_{\text{min}} = B = 589,824$, the optimal dataflow is Two-NRA. According to Principle 2, we do not tile the K-dimension, using the same scheduling as shown in the top part of Fig. 3. Next, according to the BS constraint as in Eq. 4, we maximize T_M to 512 and minimize T_L to 1. This analytical dataflow result enables non-redundant access for both tensors A and C, while minimizing memory access for tensor B to $2KL$, which matches the best dataflow searched using DSE as reported in [2], [15].

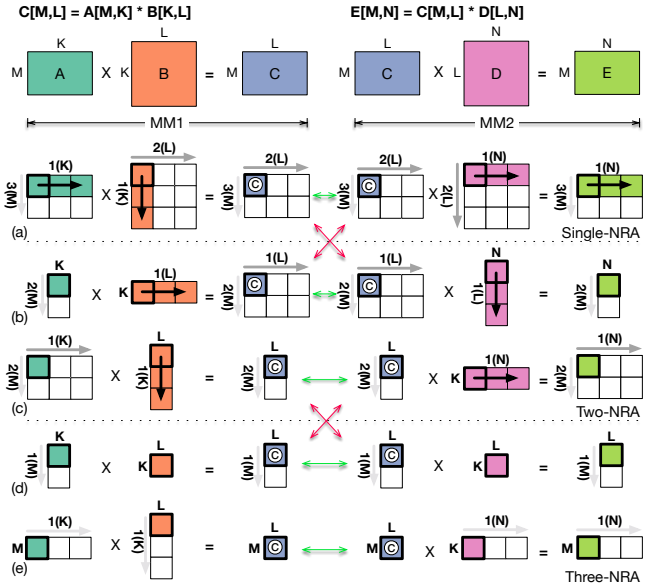


Fig. 4. Dataflow fusion patterns. The green arrows indicate profitable fusion, while the red arrows indicate the fusable but non-profitable fuison.

B. Inter-operator Dataflow

When operator fusion is introduced, the tiling and scheduling of the intermediate tensor need to be considered under the inter-operator scope. For example, in Fig. 4, the tile size and tile movement of tensor C should be identical in MM1 and MM2 if they are fused. This constraint may hinder intra-operator dataflow optimization, leading to significant MA deterioration. Therefore, it is necessary to determine whether the fusion is profitable with respect to MA.

We first give all the dataflow where operator fusion can be applied, than indicate the profitable ones.

1) Fusiablity: Tensor operators can only be fused if there is *no redundant access to the intermediate tensor* within their intra-operator dataflow. Redundant access is prohibited because the intermediate tensor is not stored in memory in fused dataflow.

There are three methods to achieve no redundant access in intra-operator dataflow: ① making the tensor stationary, ② un-tiling a dimension, ③ keeping the entire tensor in buffer. In Single-NRA, only ① is applicable. In Two-NRA, both ① and ② are applicable. In Three-NRA, ② and ③ are applicable.

By permuting and combining these methods, we can derive all feasible dataflow for fusion, as illustrated in Fig.4, which include all fused dataflow in recent works [11], [13]–[15].

As shown in Fig. 4(a), in the Single-NRA dataflow, fusion is possible only by setting MM1 as Output-Stationary (OS) and MM2 as Input-Stationary (IS), ensuring the intermediate tensor C is stationary in both operators. In Fig. 4(b) and (c), for the Two-NRA dataflow, fusion can similarly be achieved using OS-IS, or by unrolling dimension L of tensor C. Finally, as depicted in Fig. 4(d) and (e), in the Three-NRA dataflow, fusion can be enabled by either unrolling a dimension of C or retaining the entire intermediate tensor C on-chip.

There are also fusions for two operators with different NRA dataflow. We indicate them as red arrows in Fig. 4, which use making C stationary to fuse Single-NRA and Two-NRA, and un-tiling L to fuse Two-NRA and Three-NRA.

2) Profitablity: We find that fusion with the same NRA is profitable, while fusion with different NRA is non-profitable.

Although it is possible to fuse MMs with different NRA dataflow, this approach is not profitable. Consider the example of fusing Single-NRA MM1 in Fig. 4(a) with Two-NRA MM2 in (b). According to Principle 1, Single-NRA aims to maximize the dimension L, while according to Principle 2, Two-NRA minimizes L. To enable fusion, we have to choose a medium T_L, which is suboptimal for both MM1 and MM2. Although fusion eliminates the MA of tensor C, it increases the MA for other tensors. As the redundant memory accesses of other tensors dominate the overall MA, this fusion results in increased MA.

In contrast, operators sharing the same NRA dataflow follow consistent tiling principles, allowing fusion without disrupting the optimal intra-operator dataflow. We define this type of fusion as "profitable", as illustrated by the green arrows in Fig. 4.

Principle 4 *Only fuse tensor operators with the same NRA dataflow.*

Principle 1-4 can be extended to other tensor operators, as all tensor operators can be represented as for-loops, verying only on the number of loop levels while sharing consistent derivation. For the fusion of more than two operators, we can apply Principle 4 to each pair of connected operators to determine whether to fuse them.

IV. Tensor Operator Fusion Architecture

While the principles provide the minimal MA and the corresponding dataflow, these dataflow cannot be implemented on processing units (PEs) in conventional architectures.

First, unlike the mapping of intra-operator dataflow, which can be obtained from existing work, there is no work that indicates how to map all the profitable fused dataflow onto PEs.

Second, unlike many tensor accelerators that only support intra-operator dataflow and rely on a single type of stationary and tile size, accelerators designed for tensor operator fusion require *flexible stationary* and *adaptive tile size*.

A. Fused Dataflow Mapping

Mapping needs to decide which dimension maps across time and which dimensions map across PEs [20]. We refer to the tile whose dimensions all map across PEs as the "stationary tile" and the tile with one dimension mapping across time as the "moving tile", as shown in Fig. 5. The stationary tile size have to match the array size. otherwise, it will cause low utilization. While the moving tile size is less constrained.

According to the principles for tiling optimization, we identify two distinct tile sizes for the intermediate tensor C in the profitable fused dataflow in Fig. 4:

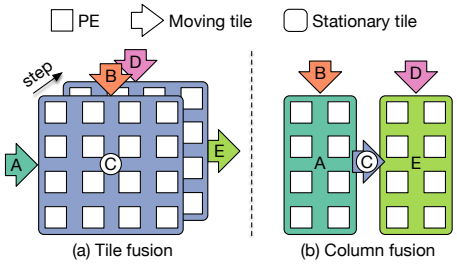


Fig. 5. Fused dataflow mappings.

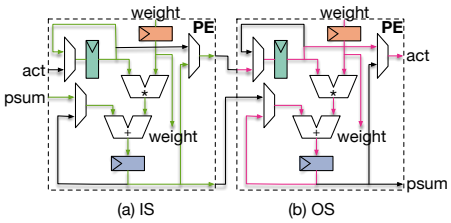


Fig. 6. PE design. Green wire: datapath for IS. Red wire: datapath for OS.

- **Tile-like**, as in Fig. 4(a), (c), and (e), the optimal tile size (ⓒ) is optimized through either maximizing or leaving untiled, according to Principle 1-3 respectively.
- **Column-like**, as in Fig. 4(b) and (d), the optimal tile size has one dimension maximized and the other minimized, according to Principle 2.

The tile-like size is well-suited as stationary tile for mapping across PEs, while the column-like size would cause low utilization if mapped as stationary tile. Therefore, we propose two fused dataflow mappings: *tile fusion* and *column fusion*.

In *tile fusion* (Fig. 5(a)), the tensor tile ⓒ is mapped as stationary tile on PEs. The computation begins with OS for $A \times B = C$, followed by IS for $C \times D = E$.

In *column fusion* (Fig. 5(b)), the PEs are divided into two parts, with the intermediate tensor tile ⓒ placed between them, mapped as moving tile. The first part uses IS, while the second part uses OS. Each column of C (ⓒ), produced by the first part, is consumed by the second part, enabling efficient pipelined computation.

For example, if we need to map the MMs with the corresponding tile sizes: $A(128,1) \times B(1,128) = C(128,128)$ and $C(128,128) \times D(128,1) = E(128,1)$, which is common in Single-NRA fused dataflow, we would use the tile fusion mapping as in Fig. 5(a). If the MMs with tile sizes $A(128,128) \times B(128,1) = C(128,1)$ and $C(128,1) \times D(1,128) = E(128,128)$, which is common in Two-NRA fused dataflow, we would use the column fusion mapping as in Fig. 5(b).

By leveraging both tile and column fusion, we provide comprehensive support for all profitable fused dataflow on PEs.

B. Operator-Fused Tensor Accelerator

To achieve flexible stationary and adaptive tile size, we propose the X-Stationary (XS) PE and FuseCU architecture.

First, we propose a novel XS PE to support flexible stationary. As shown in Fig. 6, additional multiplexers (MUXs) are used to adjust the datapath, supporting both IS and OS dataflow. Weight-Stationary (WS) can also be supported by simply swapping the positions of activations and weights. Column fusion is facilitated by the MUX in the activation output, allowing for selection between the input activation and the quantized result.

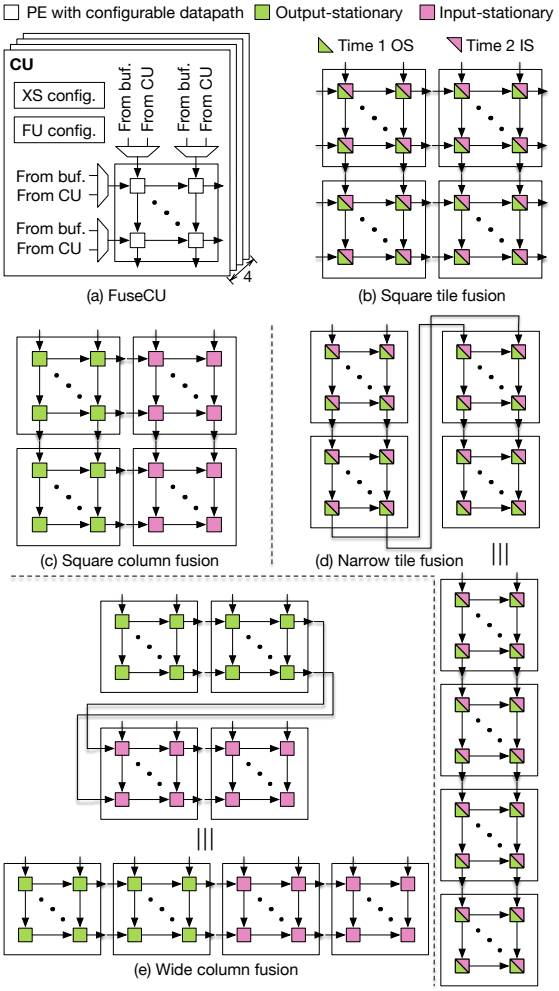


Fig. 7. FuseCU design. Wide tile fuison and narrow column fusion is omited for simplicity.

Next, we address the need for adaptive tile size. Adaptive tile size is essential for the untiled dimension in fused dataflow, as it is determined by the tensor operator.

Assuming we have an $N \times N$ PE array, our principles reveal that the widest untiled dimension size to support is $2N$.

As shown in Fig. 4, the untiled dimension occurs only in Two-NRA and Three-NRA dataflow. From Sec. III-A4, we determine the BS required for these dataflow to be optimal: $BS > D_{\text{min}}^2/4$. Since BS corresponds to the register size now, which is the number of PEs ($N \times N$), we derive $D_{\text{min}} < 2N$ from the inequality $N^2 > D_{\text{min}}^2/4$. This implies that an un-tiling strategy is optimal only when $D_{\text{min}} < 2N$, and therefore, we only need to support untiled dimension sizes smaller than $2N$. This insight guides the architecture design of FuseCU.

We propose FuseCU, which supports adaptable array configurations capable of handling all profitable fused dataflow. As shown in Fig. 7(a), FuseCU consists of four Compute Units (CUs). MUXes are added to the ports of the PE array in each CU, allowing the edge PEs to select data either from memory or adjacent CUs. The XS configuration controls the stationary within PEs, while the Fusion (FU) configuration manages connections between CUs.

Consequently, as shown in Fig. 7(b) and (c), FuseCU can achieve both tile fusion and column fusion by setting corresponding stationary

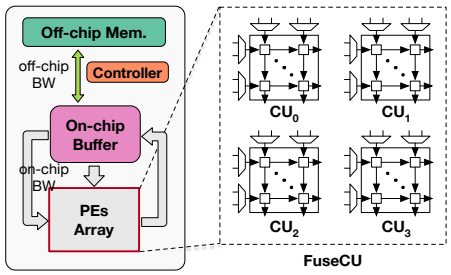


Fig. 8. Spatial accelerator architecture modeling.

TABLE II
TRANSFORMER MODEL PARAMETERS

Model	# of Heads	Seq. Length	Hidden Size
Bert	12	1024	768
GPT-2	12	2048	768
Blenderbot	16	256	1024
XLM	16	1024	2048
DeBERTa-v2	24	1024	1536
LLaMA2	32	4096(256-16K)	4096
ALBERT	64	1024	4096

patterns, and enable adaptable array sizes—square, narrow, and wide—by configuring different connections, as illustrated in Fig. 7(c-e). For instance, in Fig. 7(d), we configure the all CUs as OS-IS, connecting the bottom-left CU with the top-right CU to achieve narrow tile fusion. In Fig. 7(e), we configure the top two CUs as IS and the bottom two as OS, connecting the top-right CU with the bottom-left CU to achieve wide column fusion.

Note that by introducing the configurable array size, we support untiled dimension size of up to $2N$, and with flexible stationary, we enable all optimal inter-operator dataflow on compute units.

FuseCU can be easily applied to existing spatial architectures, as it does not modify any existing logic within the PE array. It simply adds MUX and wires to enable greater flexibility.

V. Evaluation and Results

A. Experiment Setup

Spatial architecture parameters. We evaluate our principles and FuseCU on spatial architectures, as illustrated in Fig. 8. For the memory component, we use various buffer sizes, ranging from 32KB to 32MB, to validate our principles. For the compute component, we use the configuration from TPUv4i [5], setting the total number of PEs to 128 × 128 × 4 and the on-chip bandwidth to 1TB/sec.

Architecture modeling. We adopt DAT [15] for inter-operator dataflow and MAESTRO [2] for intra-operator dataflow to obtain memory access and cycle counts. These are open-source models widely used in dataflow optimization research [3], [7], [21]. For compute modeling, we implement FuseCU in Chisel to generate Verilog RTL, which is open-sourced at https://github.com/lxu28973/fuseCU.git. The RTL is then synthesized using Synopsys Design Compiler to obtain area estimates.

TABLE III
SPATIAL ARCHITECTURE ATTRIBUTES

Platform	Stationary Flex.	Tiling Flex.	Tensor Fusion
TPUv4i	×	low	×
Gemmini	✓	low	×
Planaria	×	high	×
UnfCU	✓	middle	×
FuseCU	✓	middle	✓

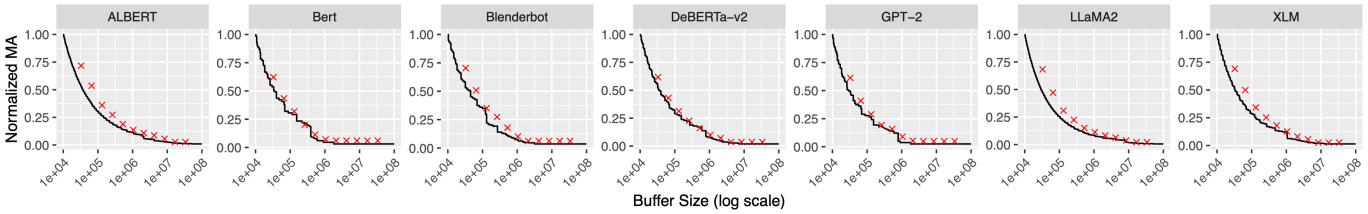


Fig. 9. Normalized memory access validation against DAT. The line indicates the dataflow our optimized, the points indicate the dataflow DAT optimized.

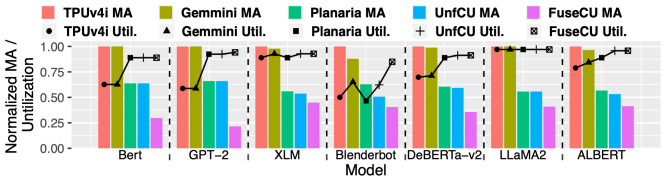


Fig. 10. Comparison of normalized memory access and utilization.

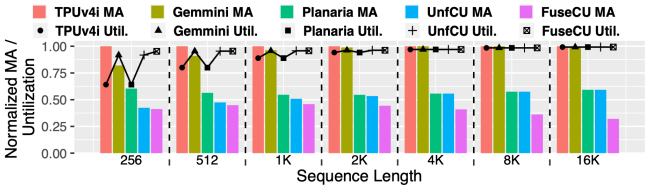


Fig. 11. Comparison of LLaMA2 under difference sequence length.

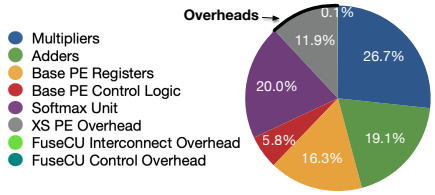


Fig. 12. FuseCU area breakdown and its overheads.

Workloads. We study seven recent open-source attention-based models. The model sizes and input sequence lengths are shown in Table II. The batch size is set as 16. We additionally evaluate LLaMA2 with sequence lengths from 256 to 16K.

Comparisons. We compare our principles optimized dataflow with those from the SOTA dataflow explorer, DAT [15]. DAT supports tensor operator fusion and utilizes mixed-integer programming and genetic algorithms to obtain optimal intra- and inter-dataflow. It has been reported to achieve better dataflow than many other approaches [10], [11], [22].

We apply FuseCU to the TPUv4i [5], and compare it with TPUv4i [5], Gemmini [16], and Planaria [17]. Gemmini supports different stationary in PE, and Planaria utilizes additional hardware to support connection between PE arrays. We also implement a design identical to FuseCU but without tensor fusion, referred to as UnfCU. We summarize these architecture attributes in Table III. All designs undergo our optimization process to select the best dataflow within their supported spaces for fair comparisons.

B. Validating the Optimality of the Four Principles

Fig. 9 shows memory access compared to DAT [15] under different buffer sizes. Two approaches achieve similar memory access, demonstrating that the principles we proposed can yield optimized dataflow. In some cases, our dataflow outperform DAT because DAT uses genetic algorithm that does not guarantee global optimization.

C. Evaluation on FuseCU Architecture

Memory Access. Fig. 10 (bar chart) presents the memory access results normalized to TPUv4i. FuseCU achieves an average reduction in memory access of 63.6%, 62.4%, and 38.7% over TPUv4i, Gemmini, and Planaria, respectively.

This improvement primarily stems from FuseCU's support for all profitable fusion dataflow, enabling it to identify superior solutions. Although still substantial without tensor operator fusion, UnfCU's memory reduction over TPUv4i, Gemmini, and Planaria decreases to 42.6%, 41.0%, and 4.5%, respectively. UnfCU's benefits result from its comprehensive support for the optimal intra-operator dataflow.

Performance. Fig. 10 (line chart) shows performance results, normalized to the target accelerator's peak FLOPs, indicating utilization. FuseCU delivers average performance gains of 1.33×, 1.25×, and 1.14× over TPUv4i, Gemmini, and Planaria, respectively.

This enhancement is driven by flexible stationary dataflow, adaptable tiling, and operator fusion. Models with smaller dimensions benefit from flexible tiling, achieving high utilization with non-square tiling. Flexible stationary strategies enhance utilization when the single stationary is suboptimal. Additionally, fusion further boosts utilization by consolidating small MMs into larger computations.

Sensitivity to Sequence Length. Fig. 11 presents the normalized memory access and utilization across different sequence lengths. FuseCU demonstrates robust performance for both short and long sequences by supporting efficient intra- and inter-operator dataflow, with greater memory access reduction observed for longer sequences.

Area Overheads. Fig. 12 presents the area breakdown across various hardware components within FuseCU when synthesized at 28 nm. This breakdown includes components introduced to support efficient fusion, specifically the XS PE logic, FuseCU resize interconnects, and control units for diverse compute patterns. Other essential elements, such as multipliers, adders, accumulators, base PE registers, control logic, and softmax unit, remain unchanged from a standard systolic array and are therefore not considered overheads.

FuseCU exhibits an area increase of 12.0% over the TPUv4i design. Notably, the FuseCU interconnects and control logic contribute less than 0.1% to this overhead—significantly lower than the 12.6% incurred by Planaria [17], which supports more flexible interconnects.

VI. Conclusion

This paper proposes principle-based dataflow optimization methodology and derives profitable tensor-wise operator fusion. These principles guide the design of FuseCU, the first architecture to unlock the full potential of tensor operator fusion in PEs. Experimental results show that FuseCU significantly reduces memory access and improves hardware utilization with minimal overhead.

REFERENCES

[1] Q. Huang, M. Kang, G. Dinh, T. Norell, A. Kalaiah, J. Demmel, J. Wawrzynek, and Y. S. Shao, "CoSA: Scheduling by Constrained Optimization for Spatial Accelerators," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2021, pp. 554–566.

[2] H. Kwon, P. Chatarasi, V. Sarkar, T. Krishna, M. Pellauer, and A. Parashar, "MAESTRO: A Data-Centric Approach to Understand Reuse, Performance, and Hardware Cost of DNN Mappings," *IEEE Micro*, vol. 40, no. 3, pp. 20–29, May 2020.

[3] S. Zheng, X. Zhang, L. Liu, S. Wei, and S. Yin, "Atomic Dataflow based Graph-Level Workload Orchestration for Scalable DNN Accelerators," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Apr. 2022, pp. 475–489.

[4] X. Yang, M. Gao, Q. Liu, J. Setter, J. Pu, A. Nayak, S. Bell, K. Cao, H. Ha, P. Raina, C. Kozyrakis, and M. Horowitz, "Interstellar: Using Halide's Scheduling Language to Analyze DNN Accelerators," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 369–383.

[5] N. P. Jouppi, D. Hyun Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, T. Norrie, N. Patil, S. Prasad, C. Young, Z. Zhou, and D. Patterson, "Ten Lessons From Three Generations Shaped Google's TPUv4i : Industrial Product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. Valencia, Spain: IEEE, Jun. 2021, pp. 1–14.

[6] A. Parashar, P. Raina, Y. S. Shao, Y.-H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2019, pp. 304–315.

[7] S.-C. Kao and T. Krishna, "GAMMA: Automating the HW Mapping of DNN Models on Accelerators via Genetic Algorithm," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Nov. 2020, pp. 1–9.

[8] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–12.

[9] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[10] M. Gao, X. Yang, J. Pu, M. Horowitz, and C. Kozyrakis, "TANGRAM: Optimized Coarse-Grained Dataflow for Scalable NN Accelerators," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, Apr. 2019, pp. 807–820.

[11] S.-C. Kao, S. Subramanian, G. Agrawal, A. Yazdanbakhsh, and T. Krishna, "FLAT: An Optimized Dataflow for Mitigating Attention Bottlenecks," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, Jan. 2023, pp. 295–310.

[12] S. Zheng, S. Chen, P. Song, R. Chen, X. Li, S. Yan, D. Lin, J. Leng, and Y. Liang, "Chimera: An Analytical Optimizing Framework for Effective Compute-intensive Operators Fusion," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Feb. 2023, pp. 1113–1126.

[13] J. Cai, Y. Wei, Z. Wu, S. Peng, and K. Ma, "Inter-layer Scheduling Space Definition and Exploration for Tiled Accelerators," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, Jun. 2023, pp. 1–17.

[14] S. Zheng, S. Chen, S. Gao, L. Jia, G. Sun, R. Wang, and Y. Liang, "TileFlow: A Framework for Modeling Fusion Dataflow via Tree-based Analysis," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, Dec. 2023, pp. 1271–1288.

[15] L. Xu, Z. Mo, Q. Wang, J. Jiang, and N. Jing, "Enabling Multiple Tensor-wise Operator Fusion for Transformer Models on Spatial Accelerators," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 1–6.

[16] H. Genc, S. Kim, A. Amid, A. Haj-Ali, V. Iyer, P. Prakash, J. Zhao, D. Grubb, H. Liew, H. Mao, A. Ou, C. Schmidt, S. Steffl, J. Wright, I. Stoica, J. Ragan-Kelley, K. Asanovic, B. Nikolic, and Y. S. Shao, "Gemmini: Enabling Systematic Deep-Learning Architecture Evaluation via Full-Stack Integration," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. San Francisco, CA, USA: IEEE, Dec. 2021, pp. 769–774.

[17] S. Ghodrati, B. H. Ahn, J. Kyung Kim, S. Kinzer, B. R. Yatham, N. Alla, H. Sharma, M. Alian, E. Ebrahimi, N. S. Kim, C. Young, and H. Esmaeilzadeh, "Planaria: Dynamic Architecture Fission for Spatial Multi-Tenant Acceleration of Deep Neural Networks," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Athens, Greece: IEEE, Oct. 2020, pp. 681–697.

[18] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "FlashAttention: Fast and memory-efficient exact attention with IO-Awareness," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 16 344–16 359.

[19] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[20] H. Kwon, P. Chatarasi, M. Pellauer, A. Parashar, V. Sarkar, and T. Krishna, "Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, Oct. 2019, pp. 754–768.

[21] L. Lu, N. Guan, Y. Wang, L. Jia, Z. Luo, J. Yin, J. Cong, and Y. Liang, "TENET: A framework for modeling tensor dataflow based on relation-centric notation," in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21. Virtual Event, Spain: IEEE Press, Nov. 2021, pp. 720–733.

[22] P. Chatarasi, H. Kwon, A. Parashar, M. Pellauer, T. Krishna, and V. Sarkar, "Marvel: A Data-Centric Approach for Mapping Deep Learning Operators on Spatial Accelerators," *ACM Transactions on Architecture and Code Optimization*, vol. 19, no. 1, pp. 1–26, Mar. 2022.

Mr.TPL: A Method for Multi-Pin Net Router in Triple Patterning Lithography

Chengkai Wang[1], Weiqing Ji[1*], Mingyang Kou[1], Zhiyang Chen[2], Fei Li[3], Nengyong Zhu[3] and Hailong Yao[1, 4*]
[1]*University of Science and Technology Beijing*, Beijing, China
[2]*Tsinghua University*, Beijing, China
[3]*Empyrean Technology Co., Ltd.*, Beijing, China
[4]*Key Laboratory of Advanced Materials and Devices for Post-Moore Chips, Ministry of Education*, Beijing, China
chengkaiwang@xs.ustb.edu.cn, weiqingji@ustb.edu.cn, kmy@ustb.edu.cn,
chenzhiy21@mails.tsinghua.edu.cn, lifei2@empyrean.com.cn, Zhuny@empyrean.com.cn, hailongyao@ustb.edu.cn

***Abstract*—Triple patterning lithography (TPL) has been recognized as one of the most promising solutions to print critical features in advanced technology nodes. A critical challenge within TPL is the effective assignment of the layout to masks. Recently, various layout decomposition methods and TPL-aware routing methods have been proposed to consider TPL. However, these methods typically result in numerous conflicts and stitches, and are mainly designed for 2-pin nets. This paper proposes a multi-pin net routing method in triple patterning lithography, called Mr.TPL. Experimental results demonstrate that Mr.TPL reduces color conflicts by 81.17%, decreases stitches by 76.89%, and achieves up to 5.4× speed improvement compared to the state-of-the-art TPL-aware routing method.**

***Index Terms*—triple patterning aware routing, detail routing, multiple patterning lithography**

I. Introduction

Triple patterning lithography has been recognized as one of the most promising solutions to print critical features in advanced technology nodes [1]. The primary challenge of TPL lies in the efficient decomposition of design layouts, ensuring that patterns are correctly assigned to different masks.

In recent years, various layout decomposition methods [2]–[4] and TPL-aware routing methods [5]–[7] have been proposed to consider TPL. Li et al. developed a layout decomposition method, called OpenMPL [2], which does not change the layout but assigns features that are close to each other to different masks. Zhang et al. proposed a dedicated low-rank SDP algorithm for MPL decomposition with augmented Lagrangian relaxation and Riemannian optimization [3]. However, since the layout patterns remain unchanged, existing layout decomposition methods inevitably lead to unsolvable color conflict issues. As shown in Fig 1(a), the four patterns are too close to each other, making it impossible to assign three colors in a rule-compliant manner, thus introducing an unresolvable color conflict. Fig 1(b) is a reasonable solution.

Ma et al. introduced a TPL aware routing method that modifies the routing grid, splits each vertice into 12 vertices to represent different masks and directions, and then applies routing [5]. Lin et al. proposed a TPL aware routing method, which uses a token graph-embedded conflict graph [6]. On the one hand, most TPL-aware routing methods have a very high time complexity, typically running 3-10 times slower. On the other hand, these methods are primarily designed for 2-pin connections, resulting in an excessive number of stitches when applied to multi-pin nets, as shown in Fig 1(c). This occurs because 2-pin methods cannot dynamically adjust the already-colored paths when connecting multiple pins. Fig 1(d) is a reasonable solution.

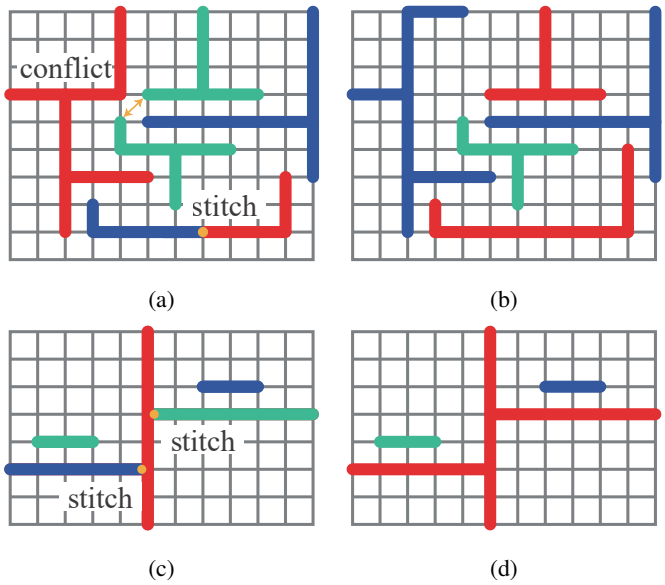


Fig. 1: Examples of TPL solution. (a) The routing solution with an unsolvable color conflict and a stitch. (b)(d) The routing solution without color conflicts or stitches. (c) Stitches caused by 2-pin TPL routing method.

To address the above issue, we propose a novel detailed routing method, called Mr.TPL, that considers TPL during the detailed routing phase, specifically targeting multi-pin nets. Our major contributions are as follows:

- We propose a search algorithm based on detail routing that better handles the TPL issues in multi-pin nets. To the best of our knowledge, this is the first work on the routing problem of multi-pin nets in TPL.

*Corresponding author: Weiqing Ji and Hailong Yao. This work was supported in part by the Key Program of National Natural Science Foundation of China (No. 62034005) and the China Postdoctoral Science Foundation under Grant Number 2024M750183.

- We propose a detailed routing method that considering TPL in the routing stage of multi-pin nets, which can greatly reduce the number of color conflicts and stitches.
- We develop a set-based color state merging and vertice coloring method that allows routing path to be in triple color candidate states at the same time during the search process, rather than a determined single color mask.
- Compared with the state-of-the-art TPL-aware routing method [5], Mr.TPL reduces color conflicts by 81.17% and stitches by 76.89%, while achieving a speedup of up to 5.4× on ISPD benchmarks.

The rest of this paper is organized as follows. Section 2 provide the preliminary and problem formulation of the TPL-aware detail routing. Section 3 provides a overall workflow of the proposed method. Section 4 elaborates on the various stages of the method. Section 5 presents experimental results, followed by Section 6, which offers conclusions based on the findings.

II. Preliminary and Problem Formulation

In this section, we first introduce the TPL aware routing. Then, we formulate the TPL aware routing problem for multi-pin nets.

A. TPL-aware Routing

Triple Patterning Lithography (TPL) is a photolithography technique used in semiconductor manufacturing to overcome the limitations of traditional lithography at advanced technology nodes [1]. By splitting a design layer into three separate masks, TPL enables the printing of smaller features, improving resolution. However, TPL introduces significant challenges in mask assignment and routing, as conflicts between features must be avoided to ensure manufacturability. TPL-aware routing integrates mask assignment considerations into the routing stage, addressing mask assignment earlier in the design process.

Fig 1 shows examples of routing solution considering TPL. As shown in Fig 1(a), when the distance between patterns on a layout falls below a predefined threshold, these patterns must be assigned to separate masks to avoid conflicts. If such conflicts arise, one way to resolve them is by assigning the layout to triple masks, introducing a stitch at the boundary where the color changes. Conflicts can lead to printing failures in the metal layout, while stitches may reduce the overall yield. The primary goal of TPL is to minimize both the number of conflicts and stitches. Fig 1(b) shows the routing solution without conflicts or stitches.

B. Problem Formulation

The TPL aware routing problem for multi-pin nets can be formulated as follows.

Given: (1) Layout, including the distribution of pre-placed standard cells, macros, obstacles, and ports. (2) The netlist, which describes the connections between components in the layout. (3) Design rules, including the design rules mentioned in ISPD contests [8] and different mask spacing D_{color}.

Goal: A routed and colored layout that minimizes the weighted sum of conflict cost and stitch cost.

III. Overview

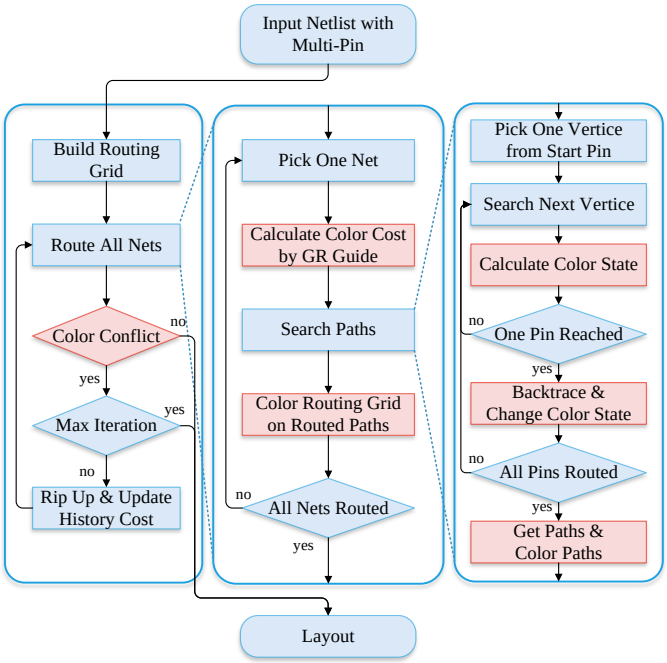


Fig. 2: Overall flow of Mr.TPL

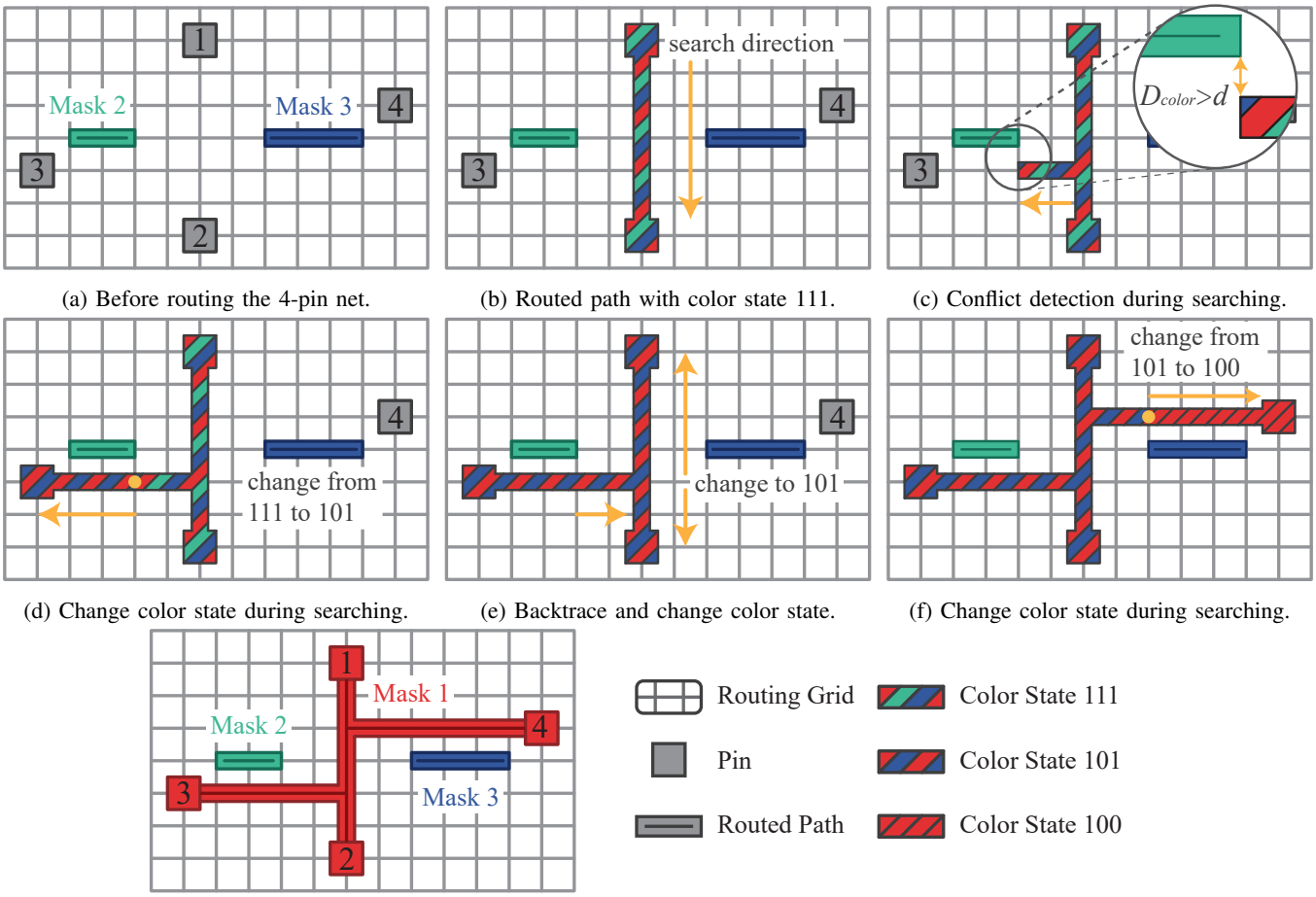
Fig 2 illustrates the overall flow of Mr.TPL. Before routing begins, the necessary data must be fully loaded including layout information, netlist, and global routing results. We also construct the routing grid at this stage. Once the data is ready, the routing process commences. After routing is completed, conflict detection is performed. If conflicts are identified and the maximum iteration limit has not been reached, the process proceeds with rip-up, reroute and updates the history cost accordingly.

The central column details the routing process. At the start of routing, a net is selected for routing. The color cost is calculated with routing regions based on global routing (GR) guide. The path-searching phase follows, during which a colored path is identified. This path is then used to mark the routed vertices and their associated colors on the routing grid. The process iterates until all nets are successfully routed.

The third column describes the detailed steps of the search path process. A pin is randomly selected from the net as the starting point for path searching. Also a vertice is selected within the coverage of the start pin. During searching to the next vertice, the color state of the vertice is calculated. This step is repeated until the next pin is reached. Subsequently, a backtrace phase and color state change is performed. The process repeats until all pins are routed. Finally, the paths are generated and colored.

IV. Method

In this section, we will introduce details of Mr.TPL. First, we will introduce the color state and how the routing grid is constructed. Then, we will explain how the cost is calculated when color is introduced. Next, we will describe how the color state of a vertice is determined during the search process



(a) Before routing the 4-pin net. (b) Routed path with color state 111. (c) Conflict detection during searching.

(d) Change color state during searching. (e) Backtrace and change color state. (f) Change color state during searching.

(g) Backtrace and final layout.

Fig. 3: Example of Mr.TPL.

and how we handle the adding of stitches when necessary. Finally, we will detail our approach to assigning colors to vertices during the backtracing stage. Fig 3 shows an example of Mr.TPL.

A. Color State

To tackle the challenge of color representation in the detailed routing process, we introduce a property called *color state* for each wire segment, which represents a part of the wire. Now we have the definition here.

Definition 1: In Mr.TPL, a ***color state*** refers to the preparatory assignment of different colors to the routing segments on the same metal layer, ensuring compliance with design rules while minimizing color conflicts and associated manufacturing costs.

This property plays a critical role in managing the triple patterning process and ensuring the correct application of masks. The color state is represented as a 3-bit unsigned integer, effectively encapsulating the necessary information. Each bit corresponds to one of the three masks—red, green, and blue. For example, a binary value of 100 represents red, 010 represents green, and 001 represents blue.

During the routing search process, a wire segment's color state can simultaneously include multiple mask options. When the color state includes three colors, it implies that assigning any of these colors will incur the same cost. For instance, a color state of 110 indicates that the wire segment can be assigned to either the red or green mask, but not the blue.

At the end of the backtracing phase, a single mask is selected for each wire segment. During this phase, the color state of each segment gradually converges to a single color, ensuring an optimal routing solution. This approach allows for flexible and efficient color management throughout the routing process, significantly enhancing the results in triple patterning lithography.

TABLE I: color state description

Encode	Description
000	none color is allowed
100	only red is allowed
010	only green is allowed
001	only blue is allowed
110	red and green are allowed
101	red and blue are allowed
011	green and blue are allowed
111	all colors are allowed

B. Graph Construction and Cost Calculation

In the initialization phase of triple patterning detailed routing, it is essential to construct the graphical representation of the routing grid. We model it as an undirected graph $G = (V, E)$, where V denotes the set of vertices representing the intersections or corners within the vertice, and E represents the set of edges indicating the connectivity between vertices. To incorporate triple patterning manufacturing considerations, we augment each edge $e \in E$ with an additional attribute, referred to as the *color state*, denoted by $c(e)$. The color state $c(e)$ takes values from a predefined finite set $C = \{c_{100}, c_{010}, \ldots, c_{111}\}$, which corresponds to the number of color states used in the process.

The calculation of routing cost is a critical aspect of the detailed routing process. The routing cost for an edge $e \in E$ is influenced by multiple factors, including traditional routing costs, stitch costs, and color conflicts. The overall routing cost $\text{Cost}(e)$ is defined as follows:

$$\text{Cost}(e) = \alpha \cdot \text{Cost}_{\text{trad}}(e) + \beta \cdot \text{Cost}_{\text{stitch}}(e) + \gamma \cdot \text{Cost}_{\text{color}}(e) \quad (1)$$

where:

- $\text{Cost}_{\text{trad}}(e)$ represents the traditional routing cost associated with edge e. The detailed computation is explained in ISPD contest [8], [9].
- $\text{Cost}_{\text{stitch}}(e)$ accounts for the cost associated with stitches, necessary when different segments of a wire are patterned using different masks and must be connected seamlessly.
- $\text{Cost}_{\text{color}}(e)$ reflects the cost associated with color conflicts, which penalizes cases where adjacent edges share the same color state.

In this equation, α, β, and γ are weighting factors that balance the traditional routing costs, stitch costs, and color conflict costs, respectively.

This comprehensive cost function is designed to guide the routing process in an TPL environment, optimize both the physical and manufacturability aspects of circuit design, thereby improving overall performance and reliability.

Algorithm 1: Multi-pin Net Routing

```
   Input: The pin list of the net pins, the routing grid G
   Output: Path result resPaths
1  startPin = pins[0];
2  solQueue = {∅};
3  resPaths = {∅};
4  foreach vertice in G.get_covered_vertices(startPin)
     do
5      vertice.cost = 0;
6      vertice.colorState = COLORSTATE(111);
7      solQueue.push(vertice);
8  end
9  while not all pins routed do
10     dstVertice = color_state_searching(solQueue, G);
11     path = backtrace(solQueue, dstVertice);
12     resPaths.push(path);
13 end
14 return resPaths;
```

C. Multi-pin Net Routing

At this stage, we search for the minimal cost path on the layout, focusing on connecting multiple pins within the net. This method incorporates color conflict costs during the path search, unlike approaches that handle color decomposition after path determination. By considering colors during pathfinding, the solution space is expanded, leading to more optimal results. Algorithm 1 provides the detailed steps.

The process begins with the initial steps (lines 1–3), where the starting pin is determined, typically the first pin in the net's pin-list. An empty priority queue is initialized, using the vertice cost as the priority criterion, alongside a path list to store the routing results. Lines 4-8 involve traversing all vertices of the starting pin, setting their cost to zero, and initializing their color state with 111. These vertices are then added to the solution queue.

Line 9 initiates the search for multi-pin nets by verifying whether any pins remain unrouted. If all pins are connected, the loop terminates. Line 10 performs color state searching, starting from the current vertices in the priority queue (solQueue) and continuing until an unconnected pin is found. Line 11 executes backtracing to finalize the path with its corresponding color state, as discussed in subsequent sections. Finally, Line 12 appends the completed path to the result list.

Algorithm 2: Color State Searching

```
   Input: The priority queue of vertices solQueue, the
          routing grid G
   Output: The vertice of reached pin dstVertice
1  dstVertice = NULL;
2  while solQueue ≠ ∅ do
3      vertice = solQueue.front();
4      if vertice is covered with an unreached pin then
5          dstVertice = vertice;
6          break;
7      end
8      for dir ∈ directions {F,B,R,L,U,D} do
9          cost = [[R,0],[G,0],[B,0]];
10         for color ∈ colors {R,G,B} do
11             cost[color][0] = color;
12             cost[color][1] = G.get_color_cost(vertice,
                dir, color);
13             if dir ∉ {U,D} &&
                color ∉ vertice.colorState then
14                 cost[color][1] += stitchCost;
15             end
16         end
17         minCost, colorState = get_lowest(cost);
18         newCost = vertice.cost + minCost;
19         solQueue.update(vertice, dir, newCost,
            colorState);
20     end
21 end
22 return dstVertice
```

D. Color State Searching

The color state searching process is a key step in routing that dynamically evaluates and updates the routing grid to minimize overall cost while considering color conflicts and stitches. This process involves iterative calculations to ensure that paths are

optimized for manufacturability within the constraints of triple patterning lithography. The algorithm takes as input the priority queue and the routing grid, iteratively searching for connections between pins. Algorithm 2 provides a detailed description of this process.

Lines 1–3 initialize the necessary data structures and define the conditions for terminating the search. Lines 4–7 check whether the current vertice is covered by an unreached pin, indicating that a new pin has been located, allowing the search for that pin to end.

Line 8 initiates the search for the next vertice in all possible directions. Line 9 defines a 2-D cost array for the current direction, structured as a 3×2 matrix. The first dimension corresponds to the three possible masks, while the second dimension's first variable stores mask information, and the second variable represents the associated color cost. Lines 10–16 handle the computation and aggregation of color costs: Lines 11–12 calculate the current cost for each mask, while Lines 13–15 evaluate whether a stitch is required and add the associated cost if present. Lines 17–19 then sort the computed color costs to identify the lowest-cost option. This minimum cost is added to vertice. cost and the updated vertice is reinserted into solQueue to continue the search.

Algorithm 3: Backtrace

Input: The priority queue of vertices $solQueue$, the vertice of reached pin $dstVertice$
Output: Path result

```
1  vertice = dstVertice;
2  while vertice ≠ NULL and vertice.cost ≠ 0 do
3      if vertice.verSetPtr = NULL then
4          vertice.verSetPtr =
             make_verSet(vertice.colorState);
5          vertice.verSetPtr.segSetPtr =
             make_segSet(vertice.colorState);
6      end
7      if vertice.prev has common color with vertice
         then
8          if vertice.prev.verSetPtr = NULL then
9              vertice.prev.verSetPtr =
                 vertice.verSetPtr;
10         end
11         else
12             tState = get_same_color(vertice.prev,
                 vertice);
13             vertice.verSetPtr.segSetPtr.
                 change_state(tState);
14             vertice.prev.verSetPtr.segSetPtr =
                 vertice.verSetPtr.segSetPtr;
15         end
16     end
17     vertice.cost = 0;
18     solQueue.update(vertice);
19     vertice = vertice.prev;
20 end
21 return get_path(dstVertice);
```

E. Backtrace

In this section, we introduce the backtrace algorithm, which is designed to optimize the tracing and cost evaluation of vertices along routing paths. This algorithm takes as input the priority queue of the solution and the destination vertice and outputs the finalized routing path. The core concept of the algorithm is to trace backward from the destination vertice to the starting vertice, progressively updating both the cost and color state of each vertice to ensure compliance with design rules and optimal path quality. Before describing the algorithm, it is helpful to introduce two key definitions:

Definition 2: The vertice color-set (***verSet***) is a collection of vertices. These vertices are adjacent on the layout, are searched consecutively, and share the same color state.

Definition 3: The segment color-set (***segSet***) is a collection of verSets. All versets in this set will have the same color state. Two connected vertices belong to different segSets only if stitch is introduced between them. Otherwise, adjacent vertices must belong to same segSet.

Algorithm 3 begins by setting the current vertice to the destination vertice of the reached pin (line 1). The loop in line 2 continues until the current vertice is either null or its cost is zero. In each iteration (lines 3-6), if the verSetPtr, which points to the verSet of the vertice, of current vertice is null, a new verSet and segSet are created based on the color state of the vertice. In lines 7-16, if the predecessor vertice shares the same color state with the current vertice, the algorithm checks if the verSetPtr of predecessor is null. If it is, the verSetPtr of predecessor is set to the verSet of current vertice. Otherwise, the algorithm computes the shared color state of both vertices and modifies the current segSet of vertice to reflect this. The segSetPtr of predecessor is then set to the current segSet of vertice. Throughout each iteration, the update function is called to update the solution of current vertice. Than, the current vertice is updated to its predecessor, and the loop continues. When the loop finishes, the final color of mask is determined by the dstVertice, which is the same as the backtracing process in Dijkstra's algorithm.

V. Experiment Results

We have implemented a triple patterning lithography detailed router in C++ and tested all benchmarks on a Gold 2.30 GHz Intel Xeon Linux Server with 16 cores and 512 GB memory. Mr.TPL is benchmarked against one of the most advanced detailed routing methods for triple patterning lithography [5], as well as the most advanced techniques in multiple patterning lithography layout decomposition [2].

Mr.TPL is designed to be integrated into widely used detailed routers. To ensure a thorough comparison, we incorporated our algorithm into Dr.CU(2.0) [10], one of the most advanced detailed routing systems available. Since the triple patterning lithography routing algorithm described in the [5] is not publicly available, we replicated it within the Dr.CU(2.0) framework to provide a fair comparison. For our experiments, we used datasets from both the ISPD 2018 [8] and ISPD 2019 contests [9], applying our multi-mask routing algorithm.

TABLE II: experiment results compared with [5] on ISPD 2018 contest

case	conflict			stitch			cost			runtime (s)		
	[5]	ours	imp.	[5]	ours	imp.	[5]	ours	imp.	[5]	ours	speedup
test1	0	**0**	$zero$[a]	3	**0**	100.0%	2.9545E+05	**2.9203E+05**	1.15%	59.93	**14.98**	4.00×
test2	0	**0**	$zero$	20	**4**	80.00%	4.7127E+06	**4.6736E+06**	0.83%	605.34	**156.76**	3.86×
test3	0	**0**	$zero$	86	**16**	81.40%	5.3864E+06	**5.2565E+06**	2.41%	1932.20	**518.25**	3.73×
test4[b]	-	**4**	-	-	**615**	-	-	**1.5055E+07**	-	>24h	**2796.72**	-
test5	2	**0**	100.0%	99	**23**	76.77%	1.6376E+07	**1.6361E+07**	0.10%	14188.33	**1110.10**	12.78×
test6	17	**1**	94.12%	185	**30**	83.78%	2.1340E+07	**2.1314E+07**	0.12%	4097.95	**886.12**	4.62×
test7	21	**3**	85.71%	354	**62**	82.49%	3.8536E+07	**3.8489E+07**	0.12%	14944.13	**2272.81**	6.58×
test8	42	**0**	100.0%	368	**63**	82.88%	3.8715E+07	**3.8674E+07**	0.11%	12584.58	**2143.91**	5.87×
test9	20	**3**	85.00%	335	**53**	84.18%	3.3054E+07	**3.3008E+07**	0.14%	5385.06	**1335.92**	4.03×
test10	352	**274**	22.16%	2151	**1710**	20.50%	**4.3454E+07**	4.3643E+07	-0.44%	20931.53	**6498.20**	3.22×
avg.	50.44	**31.22**	81.17%	400.11	**217.89**	76.89%	2.2430E+07	**2.2412E+07**	0.51%	8303.23	**1659.68**	5.41×

[a] Color conflicts are all 0, so no comparison is made.
[b] The running time of the program exceeds 24h, and no data can be obtained.

TABLE III: experiment results compared with OpenMPL [2] on ISPD 2019 contest

case	conflict			stitch		
	[2]	ours	imp.	[2]	ours	imp.
test1	358	**0**	100.00%	279	**7**	97.49%
test2	-[a]	**15**	-	-	**388**	-
test3	-	**2**	-	-	**13**	-
test4	-	**109**	-	-	**1369**	-
test5	595	**2**	99.66%	303	**6**	98.02%
test6	297	**11**	96.30%	**558**	883	58.24%
test7	-	**83**	-	-	**399**	-
test8	5782	**1**	99.98%	4561	**150**	96.71%
test9	8966	**130**	98.55%	6852	**234**	96.58%
test10	10186	**260**	97.45%	9433	**498**	94.72%
avg.	4364	**61.30**	98.66%	3664.33	**394.70**	70.88%

[a] OpenMPL [2] does not provide the relevant data, the same below.

A. Compared with the TPL-aware Routing Method

The comparison results are presented in Table 2. The *conflict* column sums the same-color and different-color conflicts. Our approach significantly reduces color conflicts, with an average decrease of 81.17%. The *stitch* column indicates the number of stitches generated during multi-mask routing, where our method achieves an average reduction of 76.89%. The *cost* column reflects the detailed routing cost score as calculated in the ISPD 2018 contest, detailed explanations can be found in the contest documentation. Our method also shows an improvement in this metric, with a 0.51% reduction in cost. The final column compares the runtime of the algorithms. As anticipated from our earlier complexity analysis, our approach demonstrates a significant advantage in computational efficiency, achieving a speedup of up to 5.4×. Notably, in the fourth example, *ispd18test4*, the DAC 2012 method failed to produce valid results within 24h, whereas our algorithm successfully completed the task, underscoring the superiority of our approach.

B. Compared with the Layout Decomposition Methods

We conducted a second experiment to compare our algorithm with the most common multiple patterning lithography layout decomposition algorithm, OpenMPL. Using the dataset from the ISPD 2019 contest, we ensured a fair comparison by utilizing the data presented in the OpenMPL paper [2]. Additionally, the input routed layout is generated by Dr.CU(2.0) [10]. The results are summarized in Table 3.

The *conflict* column represents the number of color conflicts, while the *stitch* column indicates the number of stitches. Our method demonstrates a significant reduction in both metrics, with color conflicts and stitches decreased by 98.66% and 70.88%, respectively. These results strongly highlight the advantages of considering TPL during the detailed routing stage.

VI. Conclusions

This paper investigates a triple patterning lithography aware detailed routing algorithm Mr.TPL. We propose a unified routing grid construction model, search, and coloring algorithm that effectively addresses TPL challenges in multi-pin nets. Our approach was compared against the state-of-the-art TPL-aware detailed routing and TPL decomposition algorithms. The experimental results demonstrate significant improvements in color conflict reduction, stitch count, cost score, and runtime, which proves the great effectiveness of our method.

References

[1] D. Z. Pan, L. Liebmann, B. Yu, X. Xu, and Y. Lin. Pushing multiple patterning in sub-10nm: Are we ready? In *Proc. of DAC*, pages 1–6, 2015.

[2] W. Li and *et al.* Openmpl: An open-source layout decomposer. *IEEE TCAD*, 40(11):2331–2344, 2021.

[3] Y. Zhang and *et al.* Lrsdp: Low-rank sdp for triple patterning lithography layout decomposition. In *Proc. of DAC*, pages 1–6, 2023.

[4] H. Liu and *et al.* Layout decomposition via boolean satisfiability. In *Proc. of DAC*, pages 1–6, 2023.

[5] Q. Ma, H. Zhang, and M. D. F. Wong. Triple patterning aware routing and its comparison with double patterning aware routing in 14nm technology. In *Proc. of DAC*, pages 591–596, 2012.

[6] Y. Lin, B. Yu, D. Z. Pan, and Y. Li. Triad: a triple patterning lithography aware detailed router. In *Proc. of ICCAD*, ICCAD '12, page 123–129, New York, NY, USA, 2012. Association for Computing Machinery.

[7] P. Hsu and Y. Chang. Non-stitch triple patterning-aware routing based on conflict graph pre-coloring. In *Proc. of ASP-DAC*, pages 390–395, 2015.

[8] S. Mantik, G. Posser, W. Chow, Y. Ding, and W. Liu. Ispd 2018 initial detailed routing contest and benchmarks. 2018.

[9] W. Liu and *et al.* Ispd 2019 initial detailed routing contest and benchmark with advanced routing rules. 2019.

[10] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. Young. Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction. In *Proc. of ICCAD*, pages 1–7, 2019.

Synergistic Die-Level Router for Multi-FPGA System with Time-Division Multiplexing Optimization

Jiarui Wang[1,2], Yanjing Liu[2,3], Yibo Lin[2,4,5*]

[1]School of Computer Science, Peking University, Beijing, China [2]School of Integrated Circuits, Peking University, Bejing, China

[3]School of Software & Microelectronics, Peking University, Bejing, China [4]Institute of EDA, Peking University, Wuxi, China

[5]Beijing Advanced Innovation Center for Integrated Circuits, Beijing, China

jiaruiwang@pku.edu.cn, liuyanjing@stu.pku.edu.cn, yibolin@pku.edu.cn

***Abstract*—Modern multi-FPGA systems featuring multi-die devices connected through time-division multiplexing (TDM) techniques have become increasingly common as the scale of designs increases rapidly. FPGA designs are meticulously partitioned at die-level for prototyping in modern emulation systems. Conventional FPGA-level routers often result in a large critical connection delay that impacts the whole design's frequency. Additionally, the excessive use of super long lines (SLLs) between neighboring dies leads to substantial routing congestion, causing the failure of the routing progress. To tackle these issues, we propose an effective and efficient die-level router for multi-FPGA systems, optimizing routing topology and the TDM ratio. Experimental results on the benchmarks from the die-level routing contest 2023 demonstrate 7.6% better critical connection delay with a 5.761× speed-up compared to the state-of-the-art router.**

I. Introduction

Multi-FPGA systems with multiple dies within each FPGA device are widely applied to implement large-scale circuit designs as the scale of designs increases rapidly. As shown in Fig. 1(a), a single multi-die FPGA device consists of several dies, each interconnected by super long lines (SLLs) [1, 2]. To address the challenge of limited I/O pins in the FPGA devices, the time-division multiplexing (TDM) technique [3] has been widely applied to enable efficient communication between different FPGAs. Fig. 1(b) and (c) illustrate the TDM I/O structure driven by the TDM clock, whose frequency is much higher than that of the system clock. Each physical TDM wire is assigned a TDM ratio, which specifies the number of nets it transmits within one system clock period.

There are two primary approaches to deploying a design to multi-FPGA systems: FPGA-level and die-level flows. FPGA-level flows partition a design into different FPGAs and let FPGA CAD tools like Vivado [4] finish the physical implementation of each FPGA. In contrast, die-level flows perform more fine-grained partitioning, dividing the design into dies (a.k.a SLRs in Xilinx FPGA) of FPGAs and only letting FPGA CAD tools finish the physical implementation of each die. FPGA-level flow can quickly generate the partitioning and FPGA-level routing topology, but it can lead to congestion during physical implementation for each FPGA device as it cannot precisely estimate the congestion level between and inside each die. Thus, fine-grained die-level flow is commonly used in many emulation systems like Synopsys ZeBu [5] and S2C OmniArk [6] to precisely estimate the design performance and control congestion at the early stage of the design process.

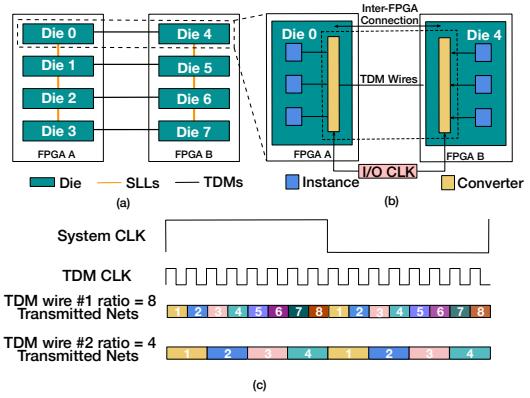


Fig. 1: (a) A multi-FPGA system with 2 FPGAs and 8 dies, with each die-to-die edge having several physical wires, for example, 100 wires. (b) The TDM edge between two dies on different FPGAs. (c) The waveforms of the system clock and the TDM clock.

System routing is an essential step in the multi-FPGA system design flow to generate inter-FPGA and inter-die routing topologies and TDM ratios in FPGA-level flow and die-level flow, respectively. Compared to FPGA-level routing, die-level routing provides more precise control and requires a more synergistic routing algorithm. A die-level router must simultaneously consider the different timing behaviors of SLL and TDM edges. It shall also synergistically avoid violating complex design rules or overlapping on SLL edges, as well as increasing design frequency by minimizing the maximum delay of all net connections (critical connection delay).

Prior academic works on multi-FPGA system routing mainly focus on FPGA-level [7] while ignoring die-level routing. They usually consist of two steps: initial routing to generate the routing topology, and TDM ratio assignment to determine TDM ratios. [8] and [9] propose SMT-based initial routing algorithms by finding minimum Steiner trees while dynamically adjusting routing costs. [10] applies maze routing [11, 12] and spanning tree algorithm [13] for nets with different criticality. However, as routing resources within different dies differ, FPGA-level initial routing cannot precisely estimate the design's frequency and can lead to insufficient SLL resources when the design's scale increases. To do the

*Corresponding author

TDM ratio assignment, many academic FPGA-level routers apply criticality-based algorithms [8, 10, 14] or Lagrangian-relaxation-based algorithms [15, 16]. Besides those works, [17] also considers the TDM capacity and direction rule during the TDM ratio assignment process. However, they ignore the different timing behavior between SLL and TDM wires, leading to an imprecise timing estimation.

Recently, [18] considers die-level routing by combining the minimum Steiner tree algorithm and maze routing algorithm for the initial routing and applying dynamic programming to assign the TDM ratio for each physical TDM wire. Their initial routing algorithm focuses on minimizing the total usage of SLL and TDM edges, which can lead to a large delay on critical connections. Their dynamic programming does not scale with design sizes either.

In this paper, we propose a synergistic die-level router for multi-FPGA systems to minimize the critical connection delay. The major contributions of this paper are summarized as follows.

- We propose a synergistic die-level router for multi-FPGA systems, optimizing both routing topologies and TDM ratios at die-level.
- We propose a balanced initial routing algorithm that synergistically optimizes connection delay and demands on SLL and TDM edges.
- We propose a multi-threaded Lagrangian-relaxation-based initial TDM ratio assignment algorithm with a margin-aware TDM ratio legalization and TDM wire assignment algorithm to assign TDM ratios for nets and physical TDM wires synergistically.

Compared to the state-of-the-art (SOTA) die-level multi-FPGA system routers, we achieve a 7.6% less critical connection delay with 5.761× speed-up on die-level multi-FPGA system routing contest 2023 [19] benchmarks (a problem of the 2023 Integrated Circuit EDA Elite Challenge organized by an industrial vendor [20]).

The rest of this paper is organized as follows. Section II describes a die-level multi-FPGA system and the problem formulation of the die-level multi-FPGA system routing problem. Section III demonstrates the algorithm flow of our router. Section IV validates our routing algorithm with experimental results. Section V concludes the paper.

II. Preliminaries

In this section, we introduce the background of the die-level multi-FPGA system routing problem and demonstrate our design rule constraints. The symbols and their descriptions used in this paper are listed in TABLE I.

A. Background

Modern FPGA architectures [1] usually have multiple dies. As shown in Fig. 1(a), FPGA A and FPGA B are two FPGAs, each with 4 dies. Neighboring dies are connected by SLL edges, and each SLL edge consists of several physical SLL wires. We define the number of physical SLL wires as the capacity of an SLL edge e, denoted as cap_e, which is the maximum number of nets it can route. The capacity of each SLL edge can be up to 10^4.

Multi-FPGA systems have been widely applied for large designs in recent years. As shown in Fig. 1(a), FPGA A and FPGA B form a multi-FPGA system consisting of two FPGAs. TDM edges $\mathbb{E}_{\mathrm{TDM}}$ connect different dies across

TABLE I: List of symbols.

Symbol	Description
$\mathbb{V}$	The set of FPGA dies.
$\mathbb{E}_{\mathrm{TDM}}, \mathbb{E}_{\mathrm{SLL}}$	The set of TDM edges and SLL edges.
$\mathbb{N}$	All the nets in the netlist.
$\mathbb{N}_e$	All the nets using edge e.
$\mathbb{C}$	All the connections.
S_{ce}	Binary variable representing whether edge e is used to route connection c.
n_c	The net of the connection c.
r_{ne}	The TDM ratio of net n on TDM edge e.
cap_e	The number of physical wires edge e contains.
demand_e	The number of nets using e.
d	the critical connection delay.
d_c	The delay of the connection c.
d_{SLL_c}	The delay of connection c contributed by the SLL edges.
d_{SLL}	The delay of the SLL edges.
p	TDM step.

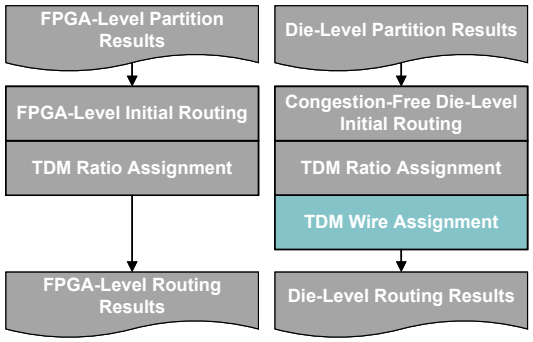


Fig. 2: The Difference between FPGA-level routing and die-level routing. (a) A typical flow of ICCAD 2019 contest [7] FPGA-level routers. (b) A typical flow of die-level routers.

different FPGAs. Each TDM edge e_{TDM} is formed by several physical TDM wires, and each physical TDM wire has a TDM ratio. The capacity of a TDM edge e is the number of physical wires it has, which is also denoted as cap_e. All the physical TDM wires within the same TDM edge share the same TDM clock but may have different TDM ratios. As shown in Fig. 1(c), the two physical wires have TDM ratios of 8 and 4, respectively.

As shown in Fig. 2, die-level routing is completely different from FPGA-level routing. Conventional FPGA-level multi-FPGA system routers (e.g. ICCAD 2019 contest routers [7]) take FPGA-level partitioning results as their inputs and assign TDM ratio between each FPGA-to-FPGA pair, which is shown in Fig. 2(a). As illustrated in Fig. 2(b), die-level routers take die-level partitioning results as inputs and output an overlapping-free routing solution with minimum critical connection delay at die-level. They shall also do TDM wire assignment to assign the TDM ratio on each physical TDM wire and assign each cross-FPGA net to physical TDM wires on the die-to-die TDM edges it passes, which is not considered by most FPGA-level routers.

B. Design rules

Unlike conventional FPGA-level routing, die-level routing deals with much more complex design rules, which are listed below.

Connectivity rule. Each connection of a net shall be routed by SLL edges or TDM edges without loops.

SLL capacity and delay rule. Each physical SLL wire can only be used to route at most one net. Therefore, the

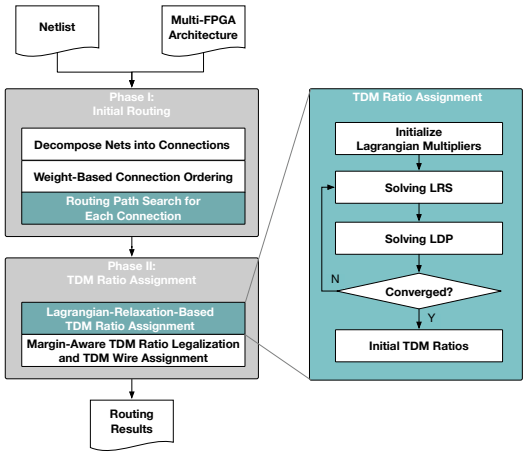


Fig. 3: The overall algorithm flow of our die-level router.

number of nets using an SLL edge e shall never exceed cap_e. The delay of all the SLL edges $\mathbb{E}_{\text{SLL}}$ is a constant d_{SLL}.

TDM wire ratio and delay rule. A physical TDM wire belonging to TDM edge e can route any number of nets. The number of nets routed by a wire is defined as its demand. The TDM ratio r of a TDM wire shall be not less than its demand, and it shall also be multiple of the TDM ratio step p. All the TDM ratios of nets using the same physical TDM wire with TDM ratio r are the same as r, and the delay of a TDM wire is defined as $d_0 + d_1 r$, where d_0 and d_1 are constants.

TDM capacity rule and TDM direction rule. For a TDM edge e, the demand for physical TDM wires shall never exceed cap_e. Moreover, although TDM edges are bi-directional, a physical TDM wire can only be used to pass signals in the same direction.

C. Die-level routing problem for multi-FPGA system

We formally define our die-level multi-FPGA system routing problem as follows.

Problem. *Given a die-level multi-FPGA system and netlist $\mathbb{N}$, find the routing paths for each net with the minimum critical connection delay following the design rules listed in Section II-B:*

$$\min \max_{c \in \mathbb{C}} \sum_{e \in \mathbb{E}_{\text{TDM}}} S_{ce}(d_0 + d_1 r_{n_c e}) + \sum_{e \in \mathbb{E}_{\text{SLL}}} S_{ce} d_{\text{SLL}}. \quad (1)$$

III. Algorithm

In this section, we introduce the algorithm of our die-level router for multi-FPGA systems.

A. Overall flow of our algorithm

The overall flow of our die-level router is shown in Fig. 3. Our router takes the netlist and the multi-FPGA system as input and outputs die-level routing paths for each net and TDM ratios for each net and TDM wire. Our router synergistically optimizes the usage of SLL edges and TDM edges by two phases: (I) initial routing to generate routing paths for each die-crossing net, and (II) TDM ratio assignment to assign TDM ratios on TDM wires to minimize the critical connection delay.

Our router first generates die-level routing paths for each net at the initial routing phase (Section III-B). We decompose each net into die-to-die connections and find a routing result while balancing the connection delay and demand on SLL and TDM edges.

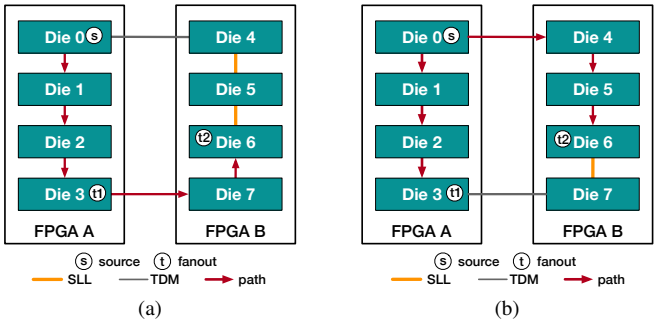


Fig. 4: (a) The minimum Steiner tree for the net connecting die 0 to die 3 and die 6 with the larger connection delay to die 5. (b) The shortest path tree for the same net with a smaller connection delay but uses more edges to route.

To further optimize the critical connection delay, our router assigns TDM ratios on each TDM edge based on our initial routing solution. We first use the Lagrangian-relaxation-based method to assign the initial TDM ratio (Section III-C). Then, we legalize the TDM ratio and assign the TDM ratio for each physical TDM wire (Section III-D).

B. Delay-demand-balanced initial routing

Our initial routing phase aims to find a routing path for each net without congestion. Some prior works [8] apply the minimum Steiner tree algorithm to get initial routing results. Those algorithms aim to minimize the total routing cost of the input design. However, as shown in Fig. 4(a), such algorithms can lead to a large connection delay when routing multi-fanout nets. An alternate idea is to find each net a shortest-path tree to minimize the connection delay. Such an idea can decrease the delay for each connection, but as shown in Fig. 4(b), it increases the usage of routing edges and can cause routing congestion on the SLL edges or lead to large TDM ratios on TDM edges. Thus, our initial routing algorithms synergistically balance the delay of each connection and the demand of each SLL and TDM edge.

To find delay-demand-balanced initial routing results for each net, our initial router decomposes each net into connections and finds each connection a routing path. The routing order of those connections is decided by sorting the routing weight by the Floyd-Warshall [21] algorithm by the decreasing order. Using the cost function listed in the following paragraphs, we then find a routing path for each connection following the negotiation-based path-finding algorithms [22]. The routing cost for each SLL and TDM edge is impacted by both edge delay and demand of each edge to synergistically consider connection delay and edge demand. As there may be congestion in the routing result since there are capacity constraints on those SLL edges, we increase the cost of those congested SLL edges and reroute those congested nets iteratively to find an overlapping-free routing result for each net.

Weight estimation and connection ordering. We assign a routing weight to each edge to calculate the routing weight of each pair of dies. Based on the number of nets in the design, SLL and TDM edges have different weights. When the number of nets on each die is less than half of the capacity of SLL edges, TDM edges are assigned a higher

weight of $\|\mathbb{V}\| + 1$, while SLL edges are given a lower weight of 1. Such a weight assignment encourages the router to explore more SLL and TDM edges for less delay. Conversely, if the number of dies exceeds this threshold, SLL edges receive a $\|\mathbb{V}\| + 1$ weight, and TDM edges are weighted at 1, which helps the router avoid congestion on SLL edges.

The routing weight of each connection is the sum of the edge weight of the shortest path between two dies, which is collected by the Floyd-Warshall algorithm [21]. Connections with higher routing weights are routed first, and connections belonging to the net with fewer fanouts have higher priority when the routing weight is the same.

Calculating routing cost. Our router iteratively applies the negotiation-based path-finding algorithms [22] to find a routing path for each connection, and we need to precisely estimate the routing cost of each SLL edge and each TDM edge to balance usage of routing edges and delay of connections. SLL and TDM edges have different routing cost functions as their timing behaviors differ. To reduce usage of the routing edges, when routing a connection of a net n, the cost of a routing edge is also impacted by whether the edge is used to route other connections of n. For SLL edges, we define their routing costs as μw_e, where w_e is the estimated edge weight, and μ is impacted by whether the SLL edge has been used to route the net the connection belongs to, which will be discussed in the following paragraph. For TDM edges, their routing costs are defined as follows:

$$\text{cost}_e = \mu(d_0 + p + \frac{\text{demand}_e}{\text{cap}_e}). \tag{2}$$

The routing cost of TDM edges increases as their demand increases, which can balance the usage of TDM edges to minimize the critical connection delay.

The parameter μ is used to balance the total usage of routing edges and the delay of each connection. When we explore edge e when routing a connection c of a net n, μ is impacted by whether e has already been used to route n. If e is not used to route n, μ is set as 1. If e is used to route other connections of n, then μ is set as a real number between 0 and 1, and we set it as $\frac{1}{2}$ in practice.

C. Lagrangian-relaxation-based initial TDM ratio assignment

It is difficult to directly assign the TDM ratio of each TDM wire and assign nets to them. Therefore, we relax the TDM wire ratio and delay rule, and assign each net a TDM ratio as a real number on each TDM edge it passes. Those initially assigned TDM ratios are legalized by the algorithms described in the next section.

The formulation of our TDM ratio assignment problem is listed as follows:

$$\begin{aligned} \min_{d, r_{ne}} \quad & d \\ s.t. \quad & \sum_{e \in \mathbb{E}_{\text{TDM}}} S_{ce} d_{ce} + d_{\text{SLL}_c} \le d, \quad \forall c \in \mathbb{C}, \\ & \sum_{n \in \mathbb{N}_e} \frac{1}{r_{ne}} \le \text{cap}_e - 1, \qquad \forall e \in \mathbb{E}_{\text{TDM}}, \end{aligned} \tag{3}$$

where

$$d_{ce} = d_0 + d_1 r_{n_c e}. \tag{4}$$

Note that to avoid a physical TDM wire being used bidirectionally, we restrict usage of each TDM edge e to no more than $\text{cap}_e - 1$, not cap_e. The usage of TDM edges is fulfilled in our TDM ratio legalization phase.

As the primal problem (PP) is difficult to solve, we apply Lagrangian relaxation (LR) to PP and get the Lagrangian relaxation subproblem (LRS) by adding Lagrangian multipliers (LMs) λ to the first set of constraints in (3), such a method is widely used in many FPGA-level TDM ratio assignment algorithms [15, 16]. LRS is much easier to solve than PP, and its optimal solution of LRS is a lower bound on PP. We list the formulation of LRS as follows:

$$\begin{aligned} \min_{d, r} \quad & L_\lambda(d, r) \\ s.t. \quad & \sum_{n \in \mathbb{N}_e} \frac{1}{r_{ne}} \le \text{cap}_e - 1, \quad \forall e \in \mathbb{E}_{\text{TDM}}. \end{aligned} \tag{5}$$

where

$$L_\lambda(d, r) = d + \sum_{c \in \mathbb{C}} \lambda_c \Big(\sum_{e \in \mathbb{E}_{\text{TDM}}} S_{ce}(d_0 + d_1 r_{n_c e}) + d_{\text{SLL}_c} - d \Big). \tag{6}$$

The optimal solution of LRS is defined as $LRS^*(\lambda)$. The maximization of $LRS^*(\lambda)$ provides the maximum lower bound of PP. We aim to maximize $LRS^*(\lambda)$ in our TDM ratio assignment phase, which is known as solving the Lagrangian dual problem (LDP). The formulation of LDP is defined as follows:

$$\begin{aligned} \max_{\lambda} \quad & LRS^*(\lambda) \\ s.t. \quad & \lambda_c \ge 0, \qquad \forall c \in \mathbb{C}. \end{aligned} \tag{7}$$

We list how we assign the initial TDM ratio and solve LDP in Algorithm 1. We init λ_c as $\frac{1}{\|\mathbb{C}\|}$ (line 1) at the beginning and iteratively update λ to maximize $LRS^*(\lambda)$ until the difference between $LRS^*(\lambda)$ and d is less than the given threshold ϵ or the number of iterations reaches the maximum iteration number (lines 2-7). In each iteration, we first solve LRS to assign TDM ratios for each net on each TDM edge it passes (line 4). Then, we calculate $LRS^*(\lambda)$ and d and update the LMs λ (lines 5-6).

Algorithm 1: Initial TDM Ratio Assignment

Input: Initial routing result, Maximum iterations $MaxIter$, Threshold ϵ
Output: Initial TDM ratios r_{ne}

1 Initialize Lagrangian Multipliers λ.
2 $Iter \leftarrow 0$.
3 **while** $\frac{d - LRS^*(\lambda)}{LRS^*(\lambda)} \ge \epsilon$ ***And*** $Iter < MaxIter$ **do**
4 Calculate r_{ne} by solving LRS.
5 Calculate $LRS^*(\lambda)$ and d.
6 Update the Lagrangian Multipliers λ.
7 $Iter \leftarrow Iter + 1$.

Finding the optimal solution of LRS. To find the optimal solution of LRS, we first apply the Karush-Kuhn-Tucker (KKT) condition:

$$\frac{\partial L_\lambda}{\partial d} = 1 - \sum_{c \in \mathbb{C}} \lambda_c = 0. \tag{8}$$

Therefore, we only need to solve the following optimization problem for each TDM edge e to minimize $L_\lambda(d, r)$:

$$\min_{r} \sum_{n \in \mathbb{N}_e} \eta_{ne} r_{ne} + \sum_{c \in \mathbb{C}} \lambda_c (d_{\text{SLL}_c} + \sum_{e \in \mathbb{E}_{\text{TDM}}} d_0 S_{ce}), \tag{9}$$

where

$$\eta_{ne} = d_1 \sum_{c \in \mathbb{C}_n} \lambda_c S_{ce}. \quad (10)$$

By applying Cauchy-Schwarz Inequality, we have:

$$(\sum_{n \in \mathbb{N}_e} \sqrt{\eta_{ne} r_{ne}}^2)(\sum_{n \in \mathbb{N}_e} \frac{1}{\sqrt{r_{ne}}^2}) \geq (\sum_{n \in \mathbb{N}_e} \sqrt{\eta_{ne}})^2. \quad (11)$$

To fully use the capacity of each TDM edge, the optimal solution of LRS must follow that $\sum_{n \in \mathbb{N}_e} \frac{1}{r_{ne}} = \text{cap}(e) - 1$. Therefore, the equality in (11) holds if and only if

$$r_{ne} = \frac{\sum_{n' \in \mathbb{N}_e} \sqrt{\eta_{n'}}}{\sqrt{\eta_n}(\text{cap}_e - 1)}. \quad (12)$$

Afterwards, we obtain the optimal solution r_{ne} for LRS.

Iteratively update solution of LDP. The target of LDP is to find λ that maximize $LRS^*(\lambda)$. Based on the fact that PP is a convex problem, which has only one optimal solution, we apply methods in [15] to update λ in each iteration by the following equation to obtain the solution of LDP quickly:

$$\lambda_c'^{i+1} = \lambda_c^i (\frac{d_c}{d})^{K_c^i}. \quad (13)$$

K_c^i is the acceleration factor, and it is updated every iteration by the method in [15]. After that, we normalize $\lambda_c'^{i+1}$ to satisify the KKT condition listed in (8).

Multi-threaded acceleration. Solving LDP usually takes many iterations, which is very time-consuming when the netlist is very large. We apply a multi-threaded technique for large-scale netlists to efficiently obtain initial TDM ratio assignment results and accelerate the process without losing quality. At the beginning of each iteration, we first calculate η_{ne} and use them to calculate r_{ne}. This progress is individual for each TDM edge and we can assign each TDM edge an independent thread to accelerate the process. Next, we need to calculate the results of LRS and LDP. To calculate them, we need to obtain the delay of each connection, which can be calculated parallelly for each connection. We assign each connection a thread to calculate its delay. To avoid the read-write conflict, we use the reduction operation [23] to parallelly calculate $LRS^*(\lambda)$ and d. We then need to update LMs λ. We first parallelly calculate $\lambda_c'^{i+1}$ and use the reduction operation to get the sum of $\lambda_c'^{i+1}$. Finally, we update λ_c for each connection individually.

D. TDM ratio legalization and margin-aware TDM wire assignment

Our initial TDM ratio assignment process assigns each net a TDM ratio on each TDM edge it passes. We then need to legalize those TDM ratios to multiples of TDM step p. The legalized TDM ratios can be used for TDM wire assignment to decide the TDM ratio of each TDM wire and assign nets to TDM wires. Such a process increases the TDM ratio of each net, leaving a margin between demand and capacity on each TDM edge. To further optimize the critical connection delay, we propose a TDM ratio refinement algorithm to resolve those margins. Our legalization and assignment process is individual for each TDM edge, and we parallelly process each edge on large designs for less runtime.

Since a physical TDM wire can only pass connections in one direction, we first assign the number of physical wires for two directions of a TDM edge. For each bi-directional TDM edge e, we first calculate the sum of $\frac{1}{r_{ne}}$ for two directions and round them up to the nearest integer as the number of physical TDM wires for each direction, which are nominated as $\text{cap}_{\overrightarrow{e}}$ and $\text{cap}_{\overleftarrow{e}}$ for two directions. Note that we restrict the sum of $\frac{1}{r_{ne}}$ less than $\text{cap}_e - 1$ in the initial assignment stage. Therefore, the sum of physical wires assigned to two directions is exact cap_e.

Algorithm 2: TDM Ratio refinement

Input: TDM Ratios $r_{n\overrightarrow{e}}$ on the directed TDM edge $\overrightarrow{e}$

Output: Refine TDM Ratios $r_{n\overrightarrow{e}}$

1 Initialize an empty priority queue q.
2 Push all the nets using $\overrightarrow{e}$ and their criticality in to q.
3 **while** $\text{cap}_{\overrightarrow{e}} - \sum \frac{1}{r_{n\overrightarrow{e}}} \geq \epsilon$ **do**
4 Pop out the most critical net n from q.
5 Decrease $r_{n\overrightarrow{e}}$ and the criticality of n.
6 Add n back to q.

To legalize TDM ratios on a TDM edge e, we first increase all the r_{ne} to the nearest multiple of TDM step p. After that, there will be a margin between demand and capacity on each TDM edge. We list how we refine TDM ratios on a directed TDM edge $\overrightarrow{e}$ in Algorithm 2. We define the criticality of a net on a TDM edge as the largest connection delay of the connections of the net passing the TDM edge. We use a priority queue to manage all the nets passing the TDM edge (line 2) and pop out the most critical net repeatedly until the margin is less than a certain threshold (lines 3-6). If the TDM ratio for the most critical net on the TDM edge is larger than TDM step p, we decrease its ratio by p and its criticality by $d_1 p$ (line 5), and push the net back into the priority queue (line 6).

After the TDM ratios are refined, we use a greedy-based method to assign the TDM ratio of physical TDM wires and assign nets to them. For each TDM wire, we assign the smallest r_{ne} of the remaining nets as its TDM ratio and then assign nets from the same direction with the top r_{ne}-smallest to use the TDM wire. After that, there will be remaining demands to assign or remaining wires not assigned a ratio. We then increase the TDM ratio of the wires with small criticality nets to assign the remaining demands or assign nets with large criticality to empty wires to resolve the remaining capacity. Finally, each net will be assigned to a TDM wire, and the TDM ratio of the wire is assigned to nets using it as their ratio on the TDM edge.

IV. Experimental Results

We implement our algorithm in C++ and use OpenMP [24] for multi-threading. Our experiments are conducted on a Linux machine with an Intel Xeon 4210-R 10-Core Processor (2.40 GHz) and 320 GB RAM. For designs with more than 200,000 nets, we use 10 threads to run phase II of our router. We use a single thread to avoid the time of multi-thread scheduling for other designs.

TABLE II: The statistics of the contest [19] benchmarks.

Design	#FPGAs	#Dies	SLL		TDM		#Nets	#Conns
			#Edges	#Wires	#Edges	#Wires		
Case #1	2	8	6	122K	2	400	5	5
Case #2	2	8	6	122K	2	400	86	155
Case #3	2	8	6	122K	2	20	84	154
Case #4	2	8	6	122K	2	40	449	577
Case #5	3	12	9	183K	3	440	5K	5K
Case #6	3	12	9	183K	14	10K	145K	281K
Case #7	4	16	12	244K	15	9K	76K	118K
Case #8	4	16	12	244K	15	7K	86K	146K
Case #9	4	16	12	244K	21	142K	871K	183K
Case #10	5	20	15	305K	19	75K	3324K	7279K

TABLE III: Critical connection delay (Delay), the number of overlapping on SLL edges (#CONF), and routing runtime (Runtime, s) on the die-level routing contest benchmarks [19] between the contest winners, [18], adapted [9] and our router.

Router \ Design		Case #1	Case #2	Case #3	Case #4	Case #5	Case #6	Case #7	Case #8	Case #9	Case #10	Norm.
1st	Delay	**6.5**	**7.5**	**11.5**	19.5	135.5	213	84.5	124	150	4657.5	1.098
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.03	0.17	6.73	2.43	3.11	50.81	356.70	1.557
2nd	Delay	**6.5**	**7.5**	14.5	19.5	136	260	102.5	145.5	196.5	4745	1.238
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.02	0.10	3.70	1.49	1.90	612.19	784.55	4.468
3rd	Delay	**6.5**	**7.5**	**11.5**	**18.5**	132.5	287	76	123	177	5700	1.171
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.03	0.87	263.14	85.56	303.77	2274.93	2115.07	34.362
[18][1]	Delay	**6.5**	**7.5**	**11.5**	**18.5**	**131**	211.5	78.5	113.5	154.5	4739.5	1.076
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.01	0.01	0.02	0.30	18.76	6.04	5.94	363.28	2169.83	5.761
Adapted [9]	Delay	**6.5**	12	28	32	185	FAIL[2]	270	505	FAIL	FAIL	2.353
	#CONF	0	0	0	0	0	11052	0	0	676430	4920599	-
	Runtime	0.08	0.06	0.06	0.07	0.14	2.96	1.34	1.59	22.69	106.38	2.323
Ours	Delay	**6.5**	**7.5**	**11.5**	**18.5**	136	**163**	**74**	**107.5**	**122.5**	**4207.5**	1.000
	#CONF	0	0	0	0	0	0	0	0	0	0	-
	Runtime	0.01	0.02	0.02	0.02	0.10	5.08	2.09	2.88	21.01	84.50	1.000

[1] We obtained the latest binary from the authors of [18]. The results on our machine are slightly different from the original paper, which have been verified by the authors [18].

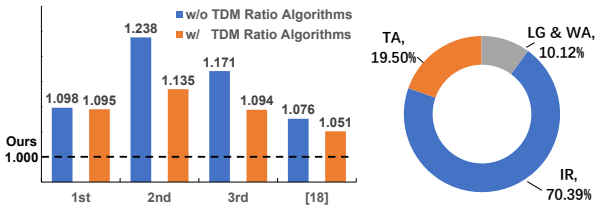
[2] 'FAIL' means the routing result is illegal as there are overlapping on the SLL edges.

We collect 10 cases from the die-level routing contest 2023 [19] provided by an industrial vendor [20], and the cases are available at [25]. The statistics of the contest benchmarks are listed in TABLE II. The number of FPGAs in the multi-FPGA system varies from 2 to 5, and the number of dies in the multi-FPGA system ranges from 8 to 20. The number of nets and connections can be up to 10^6.

We collect the binary files of the top-3 winners from the die-level routing contest [19] and the latest binary file of [18] and run their binary files on our machine by 10 threads. We list the results of those four routers and our routers in TABLE III. Compared to the contest winners, our router has a 9.8%, 23.8%, and 17.1% less critical connection delay with a 1.557×, 4.468×, and 34.362× speedup. Also, our router has a 7.6% less critical connection delay and a 5.761× speedup compared with [18].

We also collect the binary file of [9], which is the SOTA FPGA-level multi-FPGA routing to the best of our knowledge. We run their binary files by faking each die as an FPGA, with each edge as a fake FPGA-to-FPGA connection while regarding each net as a net group. We use our TDM ratio legalization and TDM wire assignment algorithm to assign the TDM ratio of their result. The adapted result is also listed in TABLE III. The adapted router fails to deal with 3 of the 10 cases and has much more critical connection delay in other cases. Note that comparing the results of adapted [9] with ours is not fair since the die-level routing problem is entirely new compared to the FPGA-level routing problem from ICCAD 2019 contest [7]. The main purpose of this experiment is to show that these two problems are quite different, and thus, simple adaption is not effective.

To validate the effectiveness of our initial routing algorithm and our TDM ratio algorithms (containing initial TDM ratio assignment, TDM ratio legalization, and TDM wire assignment), we collect the routing topology generated by the router of the contest [19] winners and [18] and use our TDM ratio algorithms to refine the results. The results of the ratio of the critical connection delay are listed in Fig. 5(a). As Fig. 5(a) shows, the results assigned by our TDM ratio algorithms can refine the critical connection delay of the routing results of the four baselines by 0.3%, 10.3%, 7.7%, and 2.5%, which shows the effectiveness of our TDM ratio algorithms. The critical connection delay of the refined



(a) (b)

Fig. 5: (a) The ratio of critical connection comparison between the results of the contest [19] winners and [18] without our TDM algorithms, the results of the contest winners and [18] refined by our TDM ratio algorithms, and the results of our router. (b) Runtime breakdown of the routing algorithms on design Case #10.

results are 9.5%, 13.5%, 9.4%, and 5.1% larger than the results of our router, demonstrating the effectiveness of our initial routing algorithm.

The runtime breakdown of our routing algorithms, including initial routing (IR), initial TDM ratio assignment (TA), TDM ratio legalization and TDM wire assignment (LG & WA), is listed in Fig. 5(b). The initial routing takes 70.39% of the total routing algorithm runtime, as we need to route each connection sequentially. Assigning the TDM ratio by Lagrangian relaxation takes 19.50% of the runtime while legalizing the TDM ratio and assigning the TDM ratio to each wire takes 10.12% of the runtime, as they are multi-threaded accelerated.

V. Conclusion

In this paper, we propose a synergistic die-level multi-FPGA system router to generate die-level routing results for multi-FPGA systems. We propose a connection-based path-searching algorithm to balance connection delay and edge demand synergistically. We propose a Lagrangian-relaxation-based initial TDM ratio assignment algorithm to effectively reduce the critical connection delay with a margin-aware TDM ratio legalization and TDM wire assignment method. Compared to the SOTA, we can achieve a 7.6% less critical connection delay with a 5.761× speed-up.

Acknowledge

This project is supported in part by the Natural Science Foundation of Beijing, China (Grant No. Z230002), and the 111 Project (B18001).

REFERENCES

[1] R. Raikar and D. Stroobandt, "Multi-die heterogeneous FPGAs: How balanced should netlist partitioning be?" ser. SLIP '22, 2023.

[2] C. Ravishankar, D. Gaitonde, and T. Bauer, "Placement strategies for 2.5D FPGA fabric architectures," in *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, 2018, pp. 16–164.

[3] J. Babb, R. Tessier, M. Dahl, S. Hanono, D. Hoki, and A. Agarwal, "Logic emulation with virtual wires," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 6, pp. 609–626, 1997.

[4] AMD, "Vivado overview," 2024, https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html.

[5] Synopsys, "Emulation systems — system verification," 2024, https://www.synopsys.com/verification/emulation.

[6] X. Zhang, "How do logic simulation, emulation, and FPGA prototyping work?" 2023, https://www.s2cinc.com/resources/lit/en/wp/s2c-how-do-logic-simulation-emulation-and-fpga-prototyping-work.pdf.

[7] Y.-H. Su, R. Sun, and P.-H. Ho, "2019 CAD contest: System-level FPGA routing with timing division multiplexing technique," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–2.

[8] P. Zou, Z. Lin, X. Shi, Y. Wu, J. Chen, J. Yu, and Y.-W. Chang, "Time-division multiplexing based system-level FPGA routing for logic verification," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[9] W.-K. Liu, M.-H. Chen, C.-M. Chang, C.-C. Chang, and Y.-W. Chang, "Time-division multiplexing based system-level FPGA routing," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–6.

[10] D. Zheng, X. Zhang, C.-W. Pui, and E. F. Young, "Multi-FPGA co-optimization: Hybrid routing and competitive-based time division multiplexing assignment," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 176–182. [Online]. Available: https://doi.org/10.1145/3394885.3431565

[11] C. Y. Lee, "An algorithm for path connections and its applications," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 346–365, 1961.

[12] H. Li, G. Chen, B. Jiang, J. Chen, and E. F. Y. Young, "Dr. cu 2.0: A scalable detailed routing framework with correct-by-construction design rule satisfaction," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2019, pp. 1–7.

[13] K. Mehlhorn, "A faster approximation algorithm for the steiner problem in graphs," *Inf. Process. Lett.*, vol. 27, no. 3, p. 125–128, mar 1988. [Online]. Available: https://doi.org/10.1016/0020-0190(88)90066-X

[14] Z. Zhuang, X. Huang, G. Liu, W. Guo, W. Qian, and W.-H. Liu, "ALIFRouter: A practical architecture-level inter-FPGA router for logic verification," in *Proc. DATE*, 2021, pp. 1570–1573.

[15] T.-W. Lin, W.-C. Tai, Y.-C. Lin, and I. H.-R. Jiang, "Routing topology and time-division multiplexing co-optimization for multi-FPGA systems," in *Proceedings of the 57th ACM/EDAC/IEEE Design Automation Conference*, ser. DAC '20. IEEE Press, 2020.

[16] C.-W. Pui and E. F. Y. Young, "Lagrangian relaxation-based time-division multiplexing optimization for multi-FPGA systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 25, no. 2, feb 2020. [Online]. Available: https://doi.org/10.1145/3377551

[17] C.-W. Pui, G. Wu, F. Y. C. Mang, and E. F. Y. Young, "An analytical approach for time-division multiplexing optimization in multi-FPGA based systems," in *2019 ACM/IEEE International Workshop on System Level Interconnect Prediction (SLIP)*, 2019, pp. 1–8.

[18] C. Huang, P. Chu, S. Bi, R. Sun, and H. You, "System routing and tdm assignment optimization in multi-2.5d FPGA-based prototyping systems," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*, 2024, pp. 324–331.

[19] S2C, "FPGA die-level system routing algorithm design," 2023, https://edaicisc.oss-cn-shanghai.aliyuncs.com/file/cacheFile/2023/10/23/0e6d23494cd24c7c94816d5bd0acf89c.pdf.

[20] ——, "S2C prototyping: FPGA ASIC SoC IP verification, validation, emulation," 2024, https://www.s2cinc.com.

[21] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, jun 1962. [Online]. Available: https://doi.org/10.1145/367766.368168

[22] L. McMurchie and C. Ebeling, "Pathfinder: A negotiation-based performance-driven router for FPGAs," in *Third International ACM Symposium on Field-Programmable Gate Arrays*, 1995, pp. 111–117.

[23] J. Ciesko, S. Mateo, X. Teruel, X. Martorell, E. Ayguadé, J. Labarta, A. Duran, B. R. de Supinski, S. Olivier, K. Li, and A. E. Eichenberger, "Towards task-parallel reductions in OpenMP," in *OpenMP: Heterogenous Execution and Data Movements*. Cham: Springer International Publishing, 2015, pp. 189–201.

[24] "OpenMP," http://www.openmp.org/.

[25] "2023 Integrated Circuit EDA Elite Challange contest problems," 2023, https://eda.icisc.cn/en/download/index?periodId=c87b85c3751c43ceb3ca482ee4cc313d&type=2.

RUPlace: Optimizing Routability via Unified Placement and Routing Formulation

Yifan Chen[1], Jing Mai[1], Zuodong Zhang[1,2], Yibo Lin[1,2,3*]
[1]School of Integrated Circuits, Peking University, Beijing, China
[2]Institute of Electronic Design Automation, Peking University, Wuxi, China
[3]Beijing Advanced Innovation Center for Integrated Circuits
Email: {chenyifan2019, jingmai, zuodongzhang, yibolin}@pku.edu.cn

***Abstract*—Placement plays a critical role in VLSI physical design, particularly in optimizing routability. With continuous advancements in semiconductor manufacturing technology, increased integration, and growing design complexity, managing routing congestion during placement has become increasingly challenging. Despite the widespread techniques to improve routability, these methods often lack theoretical guidance or sever the intrinsic connection between placement and routing optimization. In this paper, we propose `RUPlace`, an ADMM-based placer for unified optimization of placement and routing. Leveraging Wasserstein distance and bilevel optimization, our approach provides a unified framework for congestion optimization by alternately running global routing and incremental placement. Furthermore, we introduce a simple yet effective model for cell inflation-based global placement, where convex programming is employed to determine the optimal inflation ratio. Experimental results on a diverse set of open-source industrial benchmarks from `CircuitNet` and `Chipyard` demonstrate that our method achieves superior congestion reduction compared to widely used tools such as `OpenROAD`, `Xplace 2.0`, and `DREAMPlace 4.1`, while maintaining competitive wirelength and runtime.**

I. Introduction

Routability optimization in Placement plays a significant role in determining the overall performance and manufacturability of a chip. As semiconductor technology advances, increased complexity and high integration make congestion management more challenging. Routability placement usually consists of two components: routing congestion evaluation and congestion optimization.

Routing congestion evaluation is essential in routability-driven placement. Global routing [1], [2], [3], assigns temporary routing paths to estimate congestion, offering high accuracy. Statistical methods [4] predict congestion using probabilistic models, trading some accuracy for faster computation. Machine learning [5] approaches use data-driven models, though they may struggle with generalization.

For congestion optimization, there are three common approaches to addressing routing congestion: node inflation [1], [3], [6], [7], [8] based on tile congestion, where cell sizes are increased to spread them out; force-based placement [1], [8], ntuplace4h, which shifts cells to less congested areas; and machine learning-based methods [9], [10], which use data-driven models to predict and mitigate congestion.

However, node inflation methods often rely on heuristic approaches to adjust nodes based on congestion maps generated by global routing [1], [3] or statistical models [6], [7], [8]. These approaches depend heavily on human experience, lack theoretical modeling, and fail to capture the underlying connections between placement and routing outcomes. Force-based methods attempt to integrate routing considerations into placement modeling, such as introducing density constraints for route utilization [11] or incorporating congestion-weighted wirelength models [8]. However, to make such models tractable for optimization, they typically rely on simple statistical models to estimate routing outcomes, which limits their ability to embed precise routing predictions into placement. Machine learning-based methods [9], [10], on the other hand, aim to predict accurate routing results using neural networks and integrate them into placement optimization. However, these methods treat routing as a black box, disregarding the structural properties of the routing problem itself and lacking theoretical guidance.

To overcome these limitations, in this work, we propose `RUPlace`, a novel unified framework that seamlessly integrates mathematical models of both placement and routing. We formulate a unified placement and routing optimization problem that simultaneously minimizes congestion while maintaining placement quality. Rather than treating routing as a black box, this unified formulation explicitly models routing constraints and objectives alongside placement considerations. We enhance an alternating direction method of multipliers (ADMM) framework with Wasserstein distance metrics and bilevel optimization techniques to solve the problem. We further develop a principled approach to node inflation by reformulating traditional heuristic methods into a rigorous convex programming model, providing a mathematically sound and structured mechanism for congestion reduction. The key contributions of this work are summarized as follows:

- We propose a unified placement and routing formulation for routability optimization, which enables fine-grained congestion minimization.
- We propose an ADMM-based framework to solve the unified formulation leveraging Wasserstein distance and bilevel programming.
- We introduce a convex programming-based node inflation method, incorporating modularity-based clustering to automatically determine the relationship between the inflation ratio and tile congestion.

Experimental results on open-source industrial benchmarks demonstrate that our proposed algorithm can significantly reduce congestion in terms of both occurrence frequency and severity levels. Compared to the widely-used open-source EDA tool `OpenROAD` [12], our algorithm achieves 4.74× (3.47×) smaller horizontal (vertical) congestion with 7% better wirelength and a 3.67× speedup in runtime, demonstrating that our proposed algorithm significantly improves routability while achieving better wirelength with reduced runtime.

The rest of the paper is organized as follows. Sec. II introduces the basic background and problem formulation; Sec. VI explains the details of the proposed algorithm; Sec. IV validates the algorithm with experimental results; Sec. V concludes the paper.

II. Preliminaries

In this section, we provide an overview of key concepts which form the basis of our approach.

A. Analytical Placement

Analytical placement typically involves three stages: global placement (GP), legalization (LG), and detailed placement (DP). Global placement spreads instances across the layout, legalization resolves overlaps, and detailed placement refines the solution. Given the importance of GP to the overall placement quality, we focus on the GP stage. Global placement minimizes wirelength under density constraints, as in `ePlace` [13] and `NTUPlace` [11]:

$$\mathcal{BP}: \min_x \quad WL(x), \tag{1a}$$

$$\text{s.t.} \quad d_i(x) \leq d_t, \forall i \in B, \tag{1b}$$

where x is the instance coordinate, $WL(\cdot)$ is a wirelength cost function, and $d(\cdot)$ is the density function, B is the set of bins, and d_t is the whitespace in each bin.

B. Wasserstein Distance

The Wasserstein distance is used to measure the distance between two probability distributions. It is particularly useful when distributions have minimal or no overlap. Let a and b be two discrete distributions defined on a 2-dimensional grid $M \times N$. The Wasserstein distance of order p between a and b is defined as:

$$W_p(a,b) = \left\{ \inf_{\pi \in \Pi(a,b)} \sum_{(x_a,x_b)\in M\times N} \|x_a - x_b\|^p \, d\pi(x_a,x_b) \right\}^{1/p}, \tag{2}$$

where $\Pi(a,b)$ represents the set of all possible transport plans π that match the distributions a and b, $\|x_a - x_b\|^p$ is the cost of transporting mass from point x_a to point x_b raised to the power of p. In other words, the Wasserstein distance represents the minimum cost required to transform one probability distribution into another through mass transportation.

C. Hypergraph Modularity

Hypergraph modularity quantifies the quality of partitioning or clustering in hypergraphs by evaluating how well the clusters are separated while maintaining strong internal connections [14]. Consider a hypergraph $G = (V, E)$, where V represents the set of nodes and E denotes the set of hyperedges. For a given clustering result A, the hypergraph modularity Q is defined as:

$$Q = \frac{1}{|E|}\left(EC - \sum_{d=2}^{D} E_d \sum_{A_i \in A} \frac{|\text{Vol}(A_i)|}{|\text{Vol}(V)|}^d \right), \tag{3}$$

where EC represents the number of nets contained within a single cluster, E_d denotes the set of hyperedges with degree d, D is the maximum net degree in G, and $|\text{Vol}(V)|$ represents the sum of degrees of all nodes in set V.

III. Algorithm

In this section, we will further detail our algorithm.

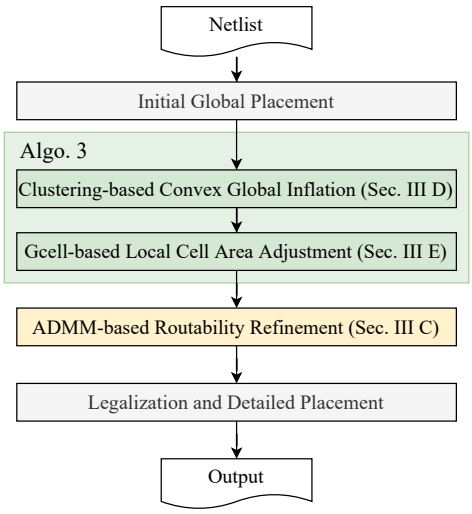


Fig. 1: The overall flow of our method.

A. Overview

The overall flow is shown in Fig. 1. The proposed algorithm initiates with a base global placement engine. When the density overflow is lower than a predefined threshold during GP, we perform cell inflation based on a convex optimization formulation for coarse-grained routability enhancement (see Sec. III-D). As the cells are inflated, the density overflow will increase again, we invoke the GP engine to perform incremental placement to spread cells. Once the density constraints are satisfied, we perform ADMM-based routability refinement for fine-grained improvement via a unified placement and routing formulation. The final stage involves legalization and detailed placement, which are performed using `DREAMPlace` [15].

B. Unified Placement and Routing Formulation

Given the placement solution x of Eq. (1), the routing problem can be formulated as an Integer Linear Programming (ILP) problem, denoted as problem $\mathcal{R}$ [16]:

$$\mathcal{R}: \min_f \quad \sum_{e\in E}\sum_{n\in N} c_e f_{n,e}, \tag{4a}$$

$$\text{s.t.} \quad \sum_n f_{n,e} \leq cap_e, \forall e \in E, \tag{4b}$$

$$Af = h(x) \tag{4c}$$

$$f_{n,e} \in \{0,1\}, \forall n \in N, e \in E, \tag{4d}$$

In this formulation, E represents the routing grid edges, e is a specific routing edge, and N is the set of all nets. The variable cap_e represents the routing capacity of edge e, and the binary variable $f_{n,e}$ indicates whether net n routes through edge e ($f_{n,e} = 1$) or not ($f_{n,e} = 0$). The constraint $Af = h(x)$ represents the flow conservation constraints for the nets, where A is a coefficient matrix for the flow conservation constraints and $h(x)$ is defined as:

$$h(x)_{n,u} = \begin{cases} +1, & \text{if the source pin of net } n \text{ in gcell } u, \\ -1, & \text{if the sink pin of net } n \text{ in gcell } u, \\ 0, & \text{otherwise,} \end{cases} \tag{5}$$

As the solution f is likely to be infeasible due to congestion, we define a congestion function as the total violation of

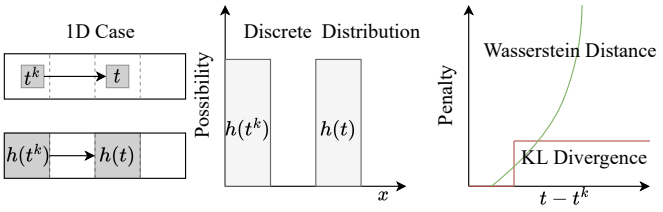


Fig. 2: Comparison of Wasserstein distance and KL divergencel. KL Divergence remains constant when two distributions have no overlap.

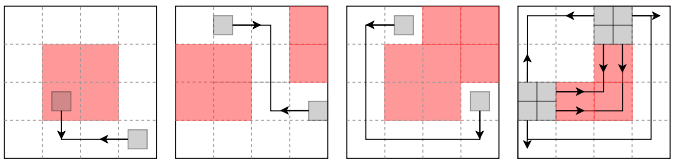


Fig. 3: Examples of how unified placement and routing resolve congestion under different scenarios. From left to right, the cases shown are the L shape net, Z shape net, 3-bend net, and long-wire congestion. Solid lines denote global routing paths. Arrows denote the intended directions of movement. The red area represents the congested region.

capacity constraints in Eq. (4b):

$$CONG(f) = \|\max(\sum_n f_{n,e} - cap_e, 0)\|. \tag{6}$$

Thus a final form of the Eq. (4) should be written as follows,

$$\mathcal{R}: \min_f \quad R(f) = \sum_{e\in E} c_e \sum_{n\in N} f_{n,e} + \mu CONG(f), \tag{7a}$$
$$\text{s.t.} \quad Af = h(x), \tag{7b}$$
$$f_{n,e} \in \{0,1\}, \forall n \in N, e \in E, \tag{7c}$$

where μ is a large weighting factor.

In routability-driven placement, we aim to optimize the post-routing result in the placement stage. Here we solve placement and routing concurrently so the objective function should follow Eq. (7a). Now we can define the concurrent placement and routing problem $\mathcal{CP}$:

$$\min_{x,f} R(f). \tag{8}$$

We expand problem $\mathcal{CP}$ considering the constraints in problem $\mathcal{BP}$ and problem $\mathcal{R}$ to obtain problem $\mathcal{UCP}$:

$$\mathcal{UCP}: \min_{x,f} \quad R(f), \tag{9a}$$
$$\text{s.t.} \quad d_i(x) \le d_t, \forall i \in B, \tag{9b}$$
$$Af = h(x), \tag{9c}$$
$$f_{n,e} \in \{0,1\}, \forall n \in N, e \in E, \tag{9d}$$

Eq. (9c) is the only constraint that couples x and f. This problem is the unified format of routability optimization for placement and routing.

C. ADMM-based Routability Refinement

Note that given x, the $\mathcal{UCP}$ is reduced to $\mathcal{R}$. Writing the optimal objective function value of $\mathcal{R}$ as $q(x)$, i.e., $q(x) = R(f^*(x); x)$, then adding $q(x)$ to the objective of $\mathcal{UCP}$ will not change the optimal solution of $\mathcal{UCP}$. Let $x = t$, $D = \{x | d_i(x) \le d_t\}, S = \{(f,t) | Af = h(t)\}$ and I_D, I_S be the indicator function of D, S, respectively, meaning $I_D(x) = 1$ if and only if $x \in D$, and $I_S(x) = 1$ if and only if $x \in S$. Then we can rewrite $\mathcal{UCP}$ to the following form,

$$\min_{x,f} \quad q(x) + I_D(x) + R(f) + I_S(f,t), \tag{10a}$$
$$\text{s.t.} \quad x = t, \tag{10b}$$
$$f_{n,e} \in \{0,1\}, \forall n \in N, e \in E, \tag{10c}$$

The augmented Lagrangian function is,

$$L(x,t,f,\lambda,\sigma) = q(x) + I_D(x) + R(f) + I_S(f,t) \tag{11a}$$
$$+ \lambda^\dagger (x - t) + \frac{\sigma}{2}(x-t)^2. \tag{11b}$$

We can leverage the alternating direction method of multipliers (ADMM) [17] to solve the problem,

$$t^{k+1}, f^{k+1} = \text{argmin}_{f,t} \quad L(x^k, t, f, \lambda^k, \sigma) \tag{12a}$$
$$x^{k+1} = \text{argmin}_x \quad L(x, t^{k+1}, f^{k+1}, \lambda^k, \sigma) \tag{12b}$$
$$\lambda^{k+1} = \lambda^k + \sigma(x^{k+1} - t^{k+1}) \tag{12c}$$

Approximating $q(x)$ by $WL(x)$ (note that $WL(x) \le q(x)$), we can expand Eq. (12b) to an analytical placement problem with new penalty terms,

$$\min_x \quad WL(x) + \lambda^{k\dagger}(x - t^{k+1}) + \frac{\sigma}{2}(x - t^{k+1})^2,$$
$$\text{s.t.} \quad d_i(x) \le d_t, \forall i \in B. \tag{13}$$

For problem Eq. (12a) its exact formulation is,

$$t^{k+1}, f^{k+1} = \text{argmin}_{Af=h(t)} R(f) + \lambda^{k\dagger}(x^k - t) + \frac{\sigma}{2}(x^k - t)^2 \tag{14}$$

which yields a problem in finding a new placement solution near x^k to minimize the routing objective. Given t, define the routing problem as $\mathcal{R}(t)$ and its solution as $\Psi(t) = \{f | f = \text{argmin}_{Af=h(t)} R(f)\}$. We can reformulate the Eq. (14) as following,

$$t^{k+1} = \text{argmin}_t \quad \mathcal{R}(t) + \lambda^{k\dagger}(x^k - t) + \frac{\sigma}{2}(x^k - t)^2 \tag{15}$$

which is a bilevel optimization problem [18]. Thus it can be decomposed into two subproblems,

$$f^{k+1} = \Psi(t^k), \tag{16a}$$
$$t^{k+1} = \text{argmin}_t \quad \hat{\mathcal{R}}(f^{k+1}, t) + \lambda^{k\dagger}(x^k - t) + \frac{\sigma}{2}(x^k - t)^2 \tag{16b}$$

where $\hat{\mathcal{R}}(f^{k+1}, t)$ serves as an linear approximation of $\mathcal{R}(t)$ in the Neighborhood of f^{k+1}. In the approximation function, we assume $h(t)$ is close to $h(t^k)$ and we only move t along the routed wires f^{k+1} so that the routed wires $\Psi(t)$ keeps the same topological structure as f^{k+1}. With this assumption, $\hat{\mathcal{R}}(f^{k+1}, t)$ is a linear function of t because if we move pins along the routed wire, the wirelength and congestion would be linearly reduced.

To ensure that $h(t)$ remains close to $h(t^k)$, a regularization penalty is required. It is important to note that a simple quadratic term, such as $(t - t^k)^2$, cannot be directly applied. This is because when certain cells are near the gridlines, even a small variation in t can lead to significant changes in $h(t)$. Since each item in $h(t)$ is in $\{-1, 0, 1\}$, we can view $h(t)$ as a distribution if we depart $h(t)$ to the positive part and negative part $h^+(t), h^-(t)$. However, certain distance metrics

Algorithm 1 ADMM-based Routability Refinement

Input: Initial position x^0
Output: Optimized position x
$\lambda \leftarrow 0$
while Not Converge **do**
  Set $t^k \leftarrow x^k$
  Solve global routing for f^{k+1} ▷ Eq. (16a)
  Compute t^{k+1} via gradient descent ▷ Eq. (17)
  Solve analytical placement for x^{k+1} ▷ Eq. (13)
  Update λ^{k+1} ▷ Eq. (12c)
  $k \leftarrow k+1$
return Final x^k

between distributions, such as KL divergence, are not suitable in this context. As illustrated in Fig. 2, when two distributions have no overlap, the penalty remains unchanged, resulting in a meaningless guiding measure. Hence, we introduce a Wasserstein distance regularization penalty to penalize the distance between $h(t)$ and $h(t^k)$, that is,

$$t^{k+1} = \text{argmin}_t \quad \hat{\mathcal{R}}(f^{k+1}, t) + \eta W_2(h^+(t), h^+(t^k))^2 + \eta W_2(h^-(t), h^-(t^k))^2 + \lambda^{k\dagger}(x^k - t) + \frac{\sigma}{2}(x^k - t)^2 \quad (17)$$

We know for each net n, the positive part $h_n(x)$ (the supply of n), written as $h_n^+(x)$, is a discrete Dirac distribution $\delta(x - x_n, y - y_n)$ where the source pin is located at grid (x_n, y_n). So Wasserstein distance can be easily calculated,

$$W_2(h_n^+(t), h_n^+(t^k))^2 = (x_n - x_n^k)^2 + (y_n - y_n^k)^2 \quad (18)$$

which is a quadratic penalty term aware of the discrete structure of $h(\cdot)$. Thus Eq. (17) is an unconstrained quadratic programming. Instead of solving it directly, we perform a single optimization step using gradient descent. The overall flow is written in Algo. 1. The entire optimization process is intuitively illustrated in Fig. 3. It can be understood as moving cells along the routed wires, enabling the cells and nets to shift away from congested regions. This also enables the unified framework to mitigate global congestion caused by long wires.

Although a unified placement and routing framework can optimize routability in various scenarios, it faces challenges when an entire net resides within a congested region, as the router cannot bypass the congested area. Consequently, this method struggles to move cells out of the congested region. To address such situations, particularly those involving large congested areas, we further propose a cell inflation technique to mitigate local congestion.

D. Clustering-based Convex Global Inflation

In this section, we globally detect the clusters with high connectivity which is likely to cause highly congested area, then uniformly inflate the nodes within same module to achieve global inflation. It is important to note that this clustering result is only used for cell inflation purposes and is not utilized during the global placement process. After detecting clusters, we assign an inflation ratio to each cluster. Following the inflation, we apply incremental placement to spread the cells. The overall flow of the Clustering-Based Convex cell inflation process is outlined in Algo. 3 line 3∼8.

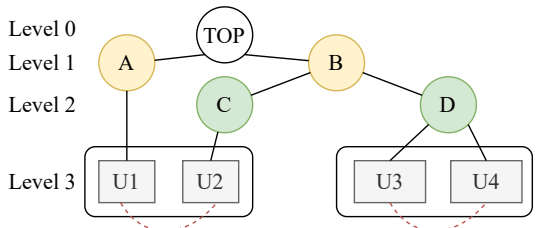


Fig. 4: Example of virtual clustering based on hypergraph modularity and logical hierarchy. We merge cells with high connectivity (red dashed lines) incrementally, level by level.

Algorithm 2 Modularity-Based Hierarchical Clustering

Input: Cell Position x, Hierarchy tree T, netlist H
Output: Clustering results
Construct initial clustering result P using the leaf nodes of T.
Compute initial modularity $Q(P, H)$
for each level l in T **do**
  while nodes can be merged **do**
    for each leaf node v in T **do**
      v belongs to module m in level l
      for each leaf node$u$ in the module m **do**
        Compute ΔQ for merging v and u in P
        if $\Delta Q > 0$ and dist$(v, u) < 4$ row_height **then**
          Merge node v and u in P and T
          Update modularity $Q(P, H)$
return P

1) Vritual Clustering Based on Hypergraph Modularity and Logical Hierarchy

We first construct a logical hierarchy tree like that in [19], shown in Fig. 4. Note [19] does not consider routability optimization and we only follow the method of constructing the logical hierarchy tree. To detect modules with high connectivity, we then proceed to merge leaf nodes level by level, guided by the hypergraph modularity. At each level, leaf nodes within the same module are merged if the modularity gain is positive and the distance betwee them lower than 4 row_height. Every merge operation is executed to ensure an increase in the overall modularity. The complete algorithm is presented in Algo. 2.

2) Convex Global Inflation

Let x be the coordinates, let $dmd_e(x)$ represent the routing demand distribution contributed by a net e, and let $D_g(x)$ denote the routing demand distribution contributed by a cluster g. To ensure that the overall routing demand distribution remains unchanged, we assume the relationship between the two distributions is governed by the equation,

$$D_g(x) = \sum_{e \in E} \frac{p(e, g)}{|e|} dmd_e(x), \quad (19)$$

where $p(e, g)$ denotes the number of pins shared by net e and cluster g, and $|e|$ is the total number of pins in net e.

Introducing an inflation ratio l_g to cluster g leads to a modified routing demand distribution $D'_g(x)$. Assuming uniform inflation across the cluster, the relationship between the original distribution $D_g(x)$ and the new distribution $D'_g(x)$ is given by

$$D'_g(x) = \frac{1}{l_g} D_g(\frac{x}{l_g}). \quad (20)$$

Algorithm 3 Global Inflation and Local Area Adjustment

1: **Input:** Cell Positions, Congestion Map
2: **Output:** Cell Inflation Ratios
3: // Global inflation once
4: Modularity-based clustering ▷ Algo. 2
5: Compute congestion distribution for each cluster ▷ Eq. (19)
6: Compute intra-cluster wirelength
7: Solve Eq. (22) using the augmented Lagrangian method
8: Inflate cells and perform incremental placement
9: // Local area adjustment
10: **while** #inflate $<$ 6 and congestion larger than 1% **do**
11: Adjust cell area ▷ Eq. (23b)
12: Incremental placement

Here, the value of $D'_g(x)$ decreases, but the affected area expands. Assume that the routing demand contribution of gcell b after inflation originates from the same gcell b' before inflation. Then the new total routing demand $Demand'(b)$ for a gcell b after inflation as follows:

$$Demand'(b) = \sum_{g \in G} D'_g(b) \sim \sum_{g \in G} \frac{1}{l_g} D_g(b'), \tag{21}$$

To ensure that the routing demand does not exceed the available routing capacity, we can formulate the cell inflation problem as a convex optimization problem:

$$\mathcal{F}: \quad \min_{v} \quad \sum_{g} \frac{WL_g}{v_g}, \tag{22a}$$

$$\text{s.t.} \quad \sum_{g} D_g(b) \cdot v_g \leq \text{cap}(b), \tag{22b}$$

where $v_g = \frac{1}{l_g}$, WL_g is the intra-cluster wirelength. The objective is to minimize the total wirelength and the constraints are designed to manage congestion by ensuring that the demand in each gcell does not exceed its capacity. Each subterm of the objective function WL_g/v_g is convex, and the constraints are linear, making this problem convex.

To solve this convex optimization problem, we apply the augmented Lagrangian algorithm, a well-established numerical method for handling constraints in optimization tasks.

E. Gcell-based Local Cell Area Adjustment

Convex global inflation assumes uniform scaling across clusters, which may mismatch routing demand and capacity. Some regions face residual congestion requiring inflation, while others have excess capacity needing shrinking. To address this, node areas are adjusted using the local demand $ldmd(b)$ and global demand $gdmd(b)$ for each gcell b. Here, $gdmd(b)$ represents the demand from nets crossing gcell b, while $ldmd(b)$ accounts for intra-gcell nets. If $ldmd(b) + gdmd(b) > cap(b)$, additional inflation is applied. Conversely, if $ldmd(b) + gdmd(b) < cap(b)$, the area is reduced.

For cells in gcell b, let $l_c(b)$ denote the inflation ratio produced by convex global inflation or the previous iteration of cell area adjustment, and let $\hat{l}_c(b)$ denote the cell inflation ratio for the current iteration of node area adjustment. The new inflation ratio can be computed as follows:

TABLE I: Benchmark Statistics [20], [21].

Design	#Macros	#Cell / K	#Nets / K	#Pins / K	Utilization
OPENC910	33	735	751	3029	0.50
NVDLA_S	45	113	123	442	0.40
NVDLA_L	376	1021	1098	3900	0.43
VORTEX_S	108	269	289	1051	0.42
VORTEX_L	80	1539	1695	6036	0.43
GEMMINI	737	926	981	3540	0.63
LARGEBOOM	636	737	756	2918	0.56

$$l'_c(b) = \max(1, \frac{ldmd(b)}{cap(b) - gdmd(b)}), \tag{23a}$$

$$\hat{l}_c(b) = (1-\gamma) l_c(b) + \gamma \min(\text{max_inflate}, l'_c(b)), \tag{23b}$$

Where γ is a fixed weighting factor controlling the balance between the old and newly computed inflation ratio $l'_c(b)$. In experiments we set $\gamma = 0.2$. We iteratively perform node area adjustments until the congestion lower than 1% or reach max iteration. The overall flow of the gcell-Based cell area adjustment process is outlined in Algo. 3 line 10∼12.

IV. Experimental Results

A. Experimental Settings

We implement our proposed algorithms based on DREAMPlace [15] and use HeLEM-GR [23] as the global router for a full GPU acceleration flow. Our experiments were conducted on a 64-bit Linux workstation equipped with an Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz (32 cores) and an NVIDIA A800 GPU with 80GB memory.

We compared our proposed algorithm with three state-of-the-art placers: OpenROAD (CPU) [1] [12], Xplace 2.0 (GPU) [2] [3], and DREAMPlace 4.1 (GPU) [3] [22]. Table I presents a comprehensive overview of our diverse benchmark suite, comprising open-source industrial designs from CircuitNet [20] and Chipyard [21]. All designs were synthesized and implemented using an advanced commercial 14nm Process Design Kit (PDK). The benchmark suite features two notable designs: NVDLA, a sophisticated deep learning accelerator, and VORTEX, an advanced RISC-V based GPGPU architecture. Each design comes in two variants - large versions (NVDLA_L, VORTEX_L) optimized for maximum performance, and small versions (NVDLA_S, VORTEX_S) tailored for energy efficiency in resource-limited environments. The inclusion of these diverse implementations underscores the comprehensive nature and versatility of our benchmark suite.

We demonstrate the effectiveness and efficiency of our proposed algorithm by evaluating three critical aspects: horizontal/vertical congestion, routed wirelength, and runtime. The metrics are reported using the industry-standard commercial tool Innovus [24] and its earlyGlobalRoute command to obtain accurate measurements.

B. Results and Analysis

Table II presents the comparison results of our proposed algorithm with three state-of-the-art placers. We can see that our proposed algorithm consistently and significantly achieves better congestion metrics than other placers, i.e., reducing

[1] With routability-driven placement enabled.
[2] With cell inflation enabled.
[3] With RUDY-based routability optimization enabled.

TABLE II: Routed Wirelenth (rWL/um), Horizontal Congestion (C_H/%), Vertical Congestion (C_V/%), and Runtime (RT/min) comparison with state-of-the-art placers.

Design	OpenROAD [12]				Xplace 2.0 [3]				DREAMPlace 4.1 [22]				RUPlace			
	rWL	C_H	C_V	RT	rWL	C_H	C_V	RT	rWL	C_H	C_V	RT	rWL	C_H	C_V	RT
OPENC910	1.34e7	7.17	4.18	20.4	1.47e7	7.27	2.68	2.8	1.22e7	10.56	5.47	1.6	1.56e7	2.02	0.72	4.3
NVDLA_S	4.98e6	0.90	0.26	4.1	4.67e6	1.01	0.37	0.7	4.43e6	1.54	0.49	0.8	4.95e6	0.09	0.09	1.8
NVDLA_L	3.92e7	3.67	0.55	28.0	3.80e7	3.78	0.69	4.3	3.58e7	4.78	1.36	3.3	4.43e7	1.36	0.23	7.1
VORTEX_S	2.63e6	2.42	0.94	5.8	1.64e6	0.85	0.34	0.5	1.59e6	1.22	0.59	0.3	1.71e6	0.28	0.16	0.8
VORTEX_L	1.17e7	0.17	0.08	12.6	1.12e7	0.24	0.14	1.6	1.10e7	0.60	0.29	2.2	1.09e7	0.13	0.10	4.9
GEMMINI	1.68e7	2.56	1.78	10.7	9.38e6	0.10	0.21	1.1	9.04e6	0.08	0.10	2.0	1.04e7	0.01	0.01	4.6
LARGEBOOM	1.20e7	0.06	0.02	10.5	1.00e7	0.97	0.51	1.4	9.78e6	1.55	0.93	1.7	1.17e7	0.31	0.11	4.0
Geo. Mean	1.07	4.74	3.47	3.67	0.93	4.11	3.88	0.45	0.88	5.91	5.80	0.43	1.00	1.00	1.00	1.00

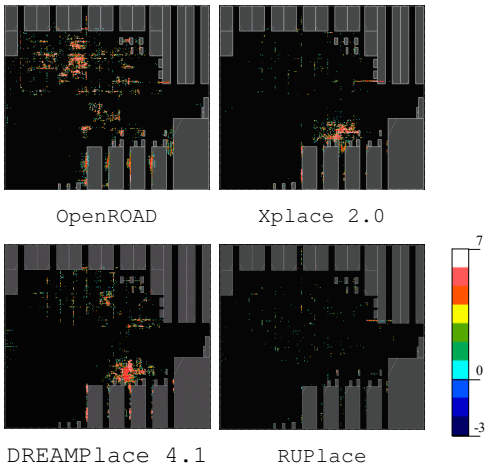


Fig. 5: Comparison of congestion distribution and severity across different placers on VORTEX_S. The colors indicate routing overflow, defined as the difference between routing demand and routing capacity. White regions represent severe congestion (demand exceeds capacity by more than 7), while blue regions indicate low congestion levels.

horizontal (vertical) congestion by 4.74× (3.47×) compared to OpenROAD, by 4.11× (3.88×) compared to Xplace 2.0, and by 5.91× (5.80×) compared to DREAMPlace 4.1. This demonstrates that our proposed algorithm can significantly reduce congestion across all test cases. Furthermore, compared to OpenROAD, our algorithm achieves 7% better routed wirelength, 4.74× smaller horizontal congestion, 3.47× smaller vertical congestion, and a 3.67× speedup in runtime, demonstrating that our proposed algorithm significantly improves routability while achieving better wirelength with reduced runtime.

Although our proposed algorithm achieves higher routed wirelength compared to Xplace 2.0 and DREAMPlace 4.1, the evaluation of placement quality requires considering both congestion and wirelength metrics. For cases with low congestion ($< 1\%$), further reduction does not provide significant improvement. However, for high congestion cases ($> 1\%$), lower congestion values are highly desirable. Our algorithm successfully maintains congestion values below 1% in almost all cases, except for hard cases like OPENC910 and NVDLA_L. Take the highly congeted case OPENC910 as an example. Our algorithm performs significantly better than the results of OpenROAD (7.17% and 4.18%), Xplace 2.0 (7.27% and 2.68%), and DREAMPlace 4.1 (10.56% and 5.47%). This demonstrates that our proposed algorithm can effectively reduce congestion even in highly congested designs. Additionally, our peak congestion values across all cases are significantly lower than those of other placers, demonstrating the practical viability of our approach in production workflows with acceptable runtime overhead.

C. Congestion Map Visualization

Fig. 5 visualizes congestion maps, highlighting differences across placers. We observe that OpenROAD exhibits more scattered overflow regions across the layout. In contrast, Xplace 2.0 and DREAMPlace 4.1 show more concentrated congestion areas with higher peak overflow values. Notably, Xplace 2.0 and DREAMPlace 4.1 share similar locations for their peak congestion regions, indicating these areas are inherently prone to congestion. The congestion map of our proposed algorithm demonstrates that it significantly alleviates congestion overall, achieving good results even in regions where previous placers struggled with severe congestion. This indicates that our algorithm effectively reduces both the occurrence and severity of congestion across the layout.

V. Conclusion

In this paper, we propose RUPlace, a unified framework for routability-driven placement in VLSI design, leveraging ADMM, Wasserstein distance, and bilevel optimization to jointly optimize placement and routing. Our approach provides a theoretically guided solution for congestion reduction, overcoming the limitations of traditional heuristic and probabilistic methods. Additionally, we introduced a convex programming-based cell inflation technique, incorporating modularity-based clustering to determine inflation ratios effectively, enhancing congestion mitigation beyond conventional heuristics. In experiments on CircuitNet and Chipyard benchmarks, our algorithm reduces horizontal (vertical) congestion by 4.74× (3.47×) compared to OpenROAD, 4.11× (3.88×) compared to Xplace 2.0, and 5.91× (5.80×) compared to DREAMPlace 4.1. It also achieves 7% lower wirelength and a 3.67× runtime speedup over OpenROAD, demonstrating its superiority in wirelength, efficiency, and routability optimization.

VI. Acknowledgment

This paper is supported in part by the National Natural Science Foundation of China (Grant No. T2293701), the Natural Science Foundation of Beijing, China (Grant No. Z230002), Grant QYJS-2023-2303-B, and the 111 project (B18001).

REFERENCES

[1] C.-C. Huang, H.-Y. Lee, B.-Q. Lin, S.-W. Yang, C.-H. Chang, S.-T. Chen, Y.-W. Chang, T.-C. Chen, and I. Bustany, "Ntuplace4dr: A detailed-routing-driven placer for mixed-size circuit designs with technology and region constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 3, pp. 669–681, 2018.

[2] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2019.

[3] L. Liu, B. Fu, S. Lin, J. Liu, E. F. Young, and M. D. Wong, "Xplace: An extremely fast and extensible placement framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023. [Online]. Available: https://github.com/cuhk-eda/Xplace/releases/tag/v2.2.1

[4] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *2007 Design, Automation Test in Europe Conference Exhibition*, 2007, pp. 1–6.

[5] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[6] T. Lin and C. Chu, "Polar 2.0: An effective routability-driven placer," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6.

[7] X. He, Y. Wang, Y. Guo, and E. F. Y. Young, "Ripple 2.0: Improved movement of cells in routability-driven placement," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 22, no. 1, sep 2016. [Online]. Available: https://doi.org/10.1145/2925989

[8] J.-M. Lin, C.-W. Huang, L.-C. Zane, M.-C. Tsai, C.-L. Lin, and C.-F. Tsai, "Routability-driven global placer target on removing global and local congestion for vlsi designs," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2021, pp. 1–8.

[9] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2021, pp. 1821–1824.

[10] S. Park, D. Kim, S. Kwon, and S. Kang, "Routability prediction and optimization using explainable ai," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–8.

[11] M.-K. Hsu, Y.-F. Chen, C.-C. Huang, S. Chou, T.-H. Lin, T.-C. Chen, and Y.-W. Chang, "Ntuplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1914–1927, 2014.

[12] T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, R. G. Dreslinski, M. Fogaça, S. M. Hashemi, A. A. Ibrahim, A. B. Kahng, M. Kim, J. Li, Z. Liang, U. Mallappa, P. I. Pénzes, G. Pradipta, S. Reda, A. Rovinski, K. Samadi, S. S. Sapatnekar, L. K. Saul, C. Sechen, V. Srinivas, W. Swartz, D. Sylvester, D. Urquhart, L. Wang, M. Woo, and B. Xu, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:210937106

[13] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "eplace: Electrostatics-based placement using fast fourier transform and nesterov's method," *ACM TODAES*, vol. 20, no. 2, p. 17, 2015.

[14] M. E. J. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, p. 8577–8582, Jun. 2006. [Online]. Available: http://dx.doi.org/10.1073/pnas.0601602103

[15] Y. Lin, S. Dhar, W. Li, H. Ren, B. Khailany, and D. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," in *Proc. DAC*, 2019.

[16] P. Yao, P. Zhang, and W. Zhu, "Pathfinding model and lagrangian-based global routing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–6.

[17] D. P. Bertsekas, *Constrained optimization and Lagrange multiplier methods*. Academic press, 2014.

[18] Y. Beck and M. Schmidt, "A gentle and incomplete introduction to bilevel optimization," 2021.

[19] A. Kahng, S. Kang, S. Kundu, K. Min, S. Park, and B. Pramanik, "Ppa-relevant clustering-driven placement for large-scale vlsi designs," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3655991

[20] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[21] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, P. Rigge, C. Schmidt, J. Wright, J. Zhao, Y. S. Shao, K. Asanović, and B. Nikolić, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[22] Y. Chen, Z. Wen, Y. Liang, and Y. Lin, "Stronger mixed-size placement backbone considering second-order information," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9. [Online]. Available: https://github.com/limbo018/DREAMPlace/releases/tag/4.1.0

[23] C. Zhao, Z. Guo, R. Wang, Z. Wen, Y. Liang, and Y. Lin, "Helem-gr: Heterogeneous global routing with linearized exponential multiplier method," in *2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[24] *Cadence Innovus Implementation System*, Cadence Design Systems, Inc., 2023, available: https://www.cadence.com.

GLiTCH : GLiTCH induced Transitions for Secure Crypto-Hardware

C. Rohin Menon *, Jayanth Balasubramanian *, Akshay Kumar E, Annapurna Valiveti, Chester Rebeiro and Janakiraman Viraraghavan
Indian Institute of Technology Madras, India

***Abstract*—Conventionally, glitch reduction is well-studied in digital design to improve power, efficiency, and security. In contrast, this paper combines the addition and removal of glitches to minimize the power side-channel leakage. Glitch Manipulation is achieved through gate sizing-based arrival time control, which is cast as a Geometric Programming formulation. We develop a framework, GLiTCH, for glitch manipulation that is guided by functional and timing simulations. The framework is evaluated on popular cipher designs like AES, CLEFIA, and SM4. Our findings illustrate up to 52.82% improvement in the Guessing Entropy for a 38.74% area overhead on average across the evaluated ciphers.**

***Index Terms*—Glitches, Side-channel attacks, geometric programming, gate-resizing.**

I. Introduction

Glitches in digital circuits are unintended transitions at the output of logic gates that are caused by differential arrival times at the gates' inputs. Glitches present a significant challenge to the designer as they can contribute to an increase in the circuit's power consumption [1]. Besides these, glitches can significantly undermine the device's security. They are known to be the main cause of side-channel leakages in protected implementations of cryptographic ciphers. Due to these leakages, information about the secret key of ciphers can leak through side-channels such as the device's power consumption [2] and EM radiation [3]. Such attacks can compromise the secret key of ciphers like the AES within a few hours, even with strong countermeasures, like masking, in place [4].

Numerous approaches have been introduced to minimize power consumption resulting from glitch activity. These include gate sizing [5], [6] and selective gate freezing [7], as well as techniques that adjust input delays [8] and flip-flop insertions [9] to prevent or mitigate unnecessary transitions. Furthermore, several tools have been developed [10] to quantify and provide guidance on reducing glitch-induced power. In security-sensitive designs, alternative methods such as threshold cryptography [11] have been explored to address glitches. However, despite their efficacy, these techniques often face implementation challenges and generally introduce considerable area ($3\times$) and timing ($2\times$) overheads [12].

Contrary to popular belief that considers glitches as a curse that needs to be eliminated, we show in this paper that glitches can, in fact, be harnessed to safeguard against side-channel attacks. We observe that glitches in a crypto-circuit are of two types: glitches that are correlated with the secret key and those that are not. Correlated glitches tend to increase side-channel leakage, while uncorrelated glitches act as noise, reducing the signal-to-noise ratio. Increasing uncorrelated glitches can, therefore, enhance side-channel security. On the other hand, by strategically altering the arrival times, correlated glitches can be reduced, enhancing the overall security of the design.

We propose a framework called GLiTCH, that, given the netlist of a cryptographic hardware design, uses gate sizing to reduce correlated glitches and increase uncorrelated glitches. GLiTCH works in two phases. The first phase quantifies the leakage of each gate through functional and timing simulations to identify critical gates in the given netlist. It then determines gate-sizing objectives in order to eliminate correlated glitches and increase uncorrelated glitches while adhering to user-specified timing constraints. The second phase performs glitch manipulation using a Geometric Programming (GP) formulation to control the arrival times of signals at the critical gates through gate-sizing [5]. The output of GLiTCH is a gate-sized netlist with reduced side-channel leakage.

The primary contributions of this work are summarized as follows:

- We present GLiTCH, a novel framework that leverages glitch manipulation to reduce side-channel leakage in RTL designs.
- Through GLiTCH, we demonstrate that in addition to reducing glitches, strategically enhancing glitches at specific gates in the design can aid in minimizing side-channel leakage.
- We validate the effectiveness of GLiTCH by evaluating it on widely used cryptographic ciphers, showcasing its ability to significantly reduce side-channel leakage.

The paper is organized as follows: Section II summarizes differential power analysis attacks and the necessary examples. The related work on glitch minimization and side-channel leakage optimization are summarized in Section III. We motivate the glitch manipulation strategy with the help of an example in Section IV. We present our framework in Section V followed by the results in Section VI. The final result concludes the paper.

* Both authors contributed equally. Corresponding authors contact: rohin@cse.iitm.ac.in, ee20b050@smail.iitm.ac.in.

TABLE I: The non-uniform distribution in Hamming weights leak information about the 2-bit k in AND gates, while the uniform distribution in EXOR gates do not leak information in EXOR gates.

m	pair of AND			
	k			
	00	**01**	**10**	**11**
00	00	00	00	00
01	00	01	00	01
10	00	00	10	10
11	00	01	10	11
$\sum$ **HW**	**0**	**2**	**2**	**4**

m	pair of EXOR			
	k			
	00	**01**	**10**	**11**
00	00	01	10	11
01	01	00	11	10
10	10	11	00	01
11	11	10	01	00
$\sum$ **HW**	**4**	**4**	**4**	**4**

II. Background

A. Differential Power Analysis

In a Differential Power Analysis (DPA) attack [2], an adversary correlates the power consumption traces collected from a crypto-device with a hypothetical power model to predict the cipher's secret key. Typically, Hamming Weight or Hamming Distance are used as the hypothetical power models. The success of a DPA attack depends on how well the secret key can be inferred from the device's power consumption. To understand this, consider a pair of EXOR and AND gates, with 2-bit inputs m and secret k. The truth tables for these gates are shown in Table I. Assume that m is uniform and that an adversary can observe the Hamming weight of each gate's output, then certain values of k can be inferred from the AND gate due to the non-uniform distribution of Hamming weights. On the other hand, the adversary cannot infer anything about k from the EXOR gate, due to the uniform distribution of Hamming weights. Thus, the signals in a design can be considered either to be *correlated* or *uncorrelated*. *Correlated* signals are the ones that can be used to infer information about k, while signals that cannot be used to glean information about k are called *uncorrelated* signals.

B. Glitches

If the inputs to a 2-bit NAND gate toggle from 0 to 1 and 1 to 0 respectively, then output remains at logic 1. However, in practice, the gate's inputs may arrive at different times, which might cause the output to briefly drop to logic 0 before recovering to the steady state logic 1 value, as seen in Figure 1. These spurious transitions are called glitches and may form if the arrival time imbalance at the gate's input exceeds the gate's inertial delay. For example, if the arrival for the inputs to the NAND gate are a_2 and a_1 respectively and its delay is d_1, then glitches may form only if *i.e.* $|a_2 - a_1| \geq d_1$. As glitches propagate in the circuit, its effects may be noticed at other gates in the NAND gates fanout. Such glitches are usually considered a menace, considerably affecting the device's power consumption and side-channel leakage. However, in this paper, we investigate glitches to show that, if used carefully, glitches can enhance the device's security by reducing side-channel leakage.

III. Related Work

As our work proposes a framework to minimize side-channel leakage using glitch manipulation, we recollect the related works in the domain of glitch manipulation and defense strategies from the side-channel community.

A. Glitch Minimization

Glitches can constitute up to 70% of the dynamic power [13] in a digital circuit, and hence, glitch reduction techniques have been explored in the prior art. Common techniques include gate freezing [7] and latch insertion [9], which require standard cell modification and ECO-like critical path modifications. Gate sizing [5], [6], [8] eliminates glitches by matching arrival times, mostly focusing on gate downsizing by localizing the timing slack. A more universal approach for gate sizing-based glitch reduction using Geometric Programming (GP) was proposed in [14], which allows both up-sizing and down-sizing while adhering to timing specifications. We adopt the GP formulation for reasons that will be explained in Section V.

B. Glitches and Side-channel Leakages

Cryptographic hardware is particularly susceptible to side-channel attacks, where adversaries exploit information leaked through power consumption to extract bits of a cipher's secret key. A widely adopted countermeasure is masking [15], which obfuscates sensitive variables in the implementation to conceal side-channel leakage. However, masking schemes are often vulnerable to glitches that correlate with the secret [4].

Threshold implementations mitigate leakage caused by glitches by decomposing the vulnerable function into multiple independent, non-interfering components, effectively nullifying the impact of glitches [11]. Despite their effectiveness, these techniques incur significant area and power overheads, and not all functions can be efficiently transformed into equivalent threshold implementations.

In this paper, we propose an alternative approach by examining the role of glitches in side-channel leakage. The subsequent section demonstrates the potential of controlled glitch manipulation as a novel and effective method for mitigating side-channel vulnerabilities.

IV. A Motivating Example

The success of a DPA attack depends on how well the adversary is able to correlate the observed signals to the secret key. We refer to the ratio of the number of correlated signals to the total number of observed signals, as the adversary's Distinguishing Advantage (DA). As an example, we use Figure 1 to motivate how glitches can affect the DA of an adversary at the output of the EXOR gate, *i.e.* at the wire n_6.

We assume that every gate has unit delay and all inputs are initially at Logic '1'. In the circuit, consider m_1 and m_2 as two uniformly distributed values and k as a fixed secret value. Due to the varying logic depths (fan-in cones), each input may arrive at different times. This is captured by the arrival times a_1, a_2, a_3, and a_4 that can take any value in $\{1\}$, $\{1, 3, 4, 5\}$, $\{2, 3, 4, 5\}$ and $\{2, 4, 5, 6\}$, respectively. The

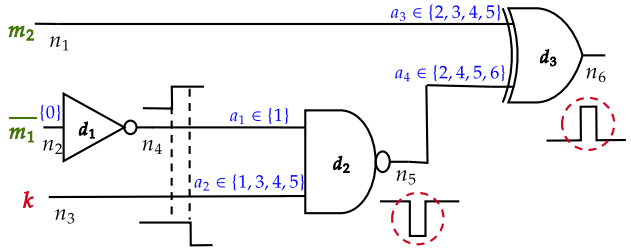


Fig. 1: Glitch generation with arrival times and signal transitions.

transitions observed at wire n_6 depend on the specific arrival times of signals at the wires n_1 and n_5. These timing variations result in one of three transition patterns at n_6:

1) **Case 1:** $a_3 = a_4$

$$\langle 1 \rightarrow \overline{m_1 \cdot k} \oplus m_2 \rangle$$

2) **Case 2:** $a_3 > a_4$

$$\langle 1 \rightarrow \overline{m_1 \cdot k} \oplus 1 \rightarrow \overline{m_1 \cdot k} \oplus m_2 \rangle$$

3) **Case 3:** $a_4 > a_3$

$$\langle 1 \rightarrow \overline{m_1} \oplus 1 \rightarrow \overline{m_1} \oplus m_2 \rightarrow \overline{m_1 \cdot k} \oplus m_2 \rangle$$

Among these transitions, Case 2 features a single correlated signal associated with $\overline{m_1 \cdot k} \oplus 1$, while the signal corresponding to $\overline{m_1 \cdot k} \oplus m_2$ is effectively masked by the uniform distribution of m_2. In contrast, Cases 1 and 3 exhibit no correlated signals. Consequently, DA for Case 2 is $\frac{1}{2}$ (ignoring the initial signal), whereas for Cases 1 and 3, the DA is 0.

A. Effect of a glitch

Glitches can either aid or hinder an attacker, depending on their influence on the adversary's DA. We examine the dual impact of glitches, analyzing their effects on DA in both scenarios.

Glitch decreasing DA: Suppose that $a_3 > a_4$ and a glitch occurs on the wire n_1. Then, the signal on the wire n_1 transitions as: $\langle 1 \rightarrow m_2 \rightarrow \overline{m_2} \rightarrow m_2 \rangle$. Due to this, the signal transitions at wire the n_6 are:

$$\langle 1 \rightarrow \overline{m_1 \cdot k} \oplus 1 \rightarrow \overline{m_1 \cdot k} \oplus m_2 \rightarrow \overline{m_1 \cdot k} \oplus \overline{m_2} \rightarrow \overline{m_1 \cdot k} \oplus m_2 \rangle$$

The glitch at n_1 has resulted in a reduced DA from $\frac{1}{2} = 0.5$ to $\frac{1}{4} = 0.25$. Hence, it becomes evident that a glitch on the preceding gate of the circuit could, in fact, introduce uncorrelated signals at subsequent nodes.

Glitch increasing DA: Suppose that $a_3 > a_4$ and a glitch occurs on the wire n_3. Then, the signal on the wire n_3 transitions as $\langle 1 \rightarrow k \rightarrow \overline{k} \rightarrow k \rangle$. Then the transitions at wire n_6 are:

$$\langle 1 \rightarrow \overline{m_1 \cdot k} \oplus 1 \rightarrow \overline{m_1 \cdot \overline{k}} \oplus 1 \rightarrow \overline{m_1 \cdot k} \oplus 1 \rightarrow \overline{m_1 \cdot k} \oplus m_2 \rangle$$

The corresponding DA has gone up from 0.5 to $\frac{3}{4} = 0.75$. We can observe that a glitch in the circuit may either increase or decrease the adversary's DA depending on the specific signal that glitches.

B. Gate sizing as a solution

A key insight from the preceding analysis is that the circuit does not leak if the computation of $\overline{m_1 \cdot k}$ occurs simultaneously with or after the arrival of m_2, establishing the condition $a_4 \geq a_3$. *Gate sizing* offers an effective mechanism to control signal arrival times without modifying the logical implementation or introducing additional delay elements, such as buffers, into the circuit. In Figure 1, gate sizing is applied to the NAND gate to control the arrival of k, ensuring $a_2 \geq a_1$. This adjustment delays the computation of $\overline{m_1 \cdot k}$, ultimately ensuring $a_4 \geq a_3$ and minimizing the adversary's ability to distinguish key-related operations. Note that the condition $a_2 \geq a_1$ aligns with the occurrence of a glitch at the NAND gate.

Contemporary works [16], [17] have explored gate-sizing as a side-channel countermeasure. Karna [16], for instance, reconfigured gates in the design to reduce leakage occurring due to dynamic power consumption. On the other hand, [17] considered leakage due to static power consumption. However, neither study has examined the impact of glitches on side-channel leakage, which is the primary focus of this paper.

V. GLiTCH : The Framework

This section presents the two phases of our framework, *leakage assessment* and *glitch manipulation formulation* while providing details about the steps involved in each phase.

The complete workflow of GLiTCH is illustrated in Figure 2. Our analysis employs a leakage analysis tool to identify circuit leakage. Initially, functional simulation is performed on the gate-level netlist with zero gate delay, producing a leakage vector for each gate, denoted as L_{FN}. This zero-delay netlist is then analyzed using the leakage tool to quantify the circuit's *functional* leakage. Subsequently, the area-minimized (AM) version of the netlist is simulated with timing delays to capture practical gate behavior. The leakage analyzer estimates the leakage vector for this configuration as L_{AM}. Since the zero-delay simulation (FN) considers only functional transitions, the differences in leakage between FN and AM can be attributed exclusively to glitches introduced by timing delays. If the leakage in the AM circuit is lower than that in the FN circuit, it indicates the presence of randomizing glitches that act as noise, reducing leakage. To amplify this effect, we intentionally increase the disparity in arrival times at the corresponding gate to encourage the occurrence of such randomizing glitches. Conversely, if the AM leakage exceeds that of FN, it signifies the presence of correlated glitches, contributing to higher leakage. In this case, we minimize the difference in arrival times at the gate to suppress or eliminate these correlated glitches. This approach allows us to systematically identify critical gates whose glitch behavior significantly influences the overall leakage. We then

determine modification objectives for these critical gates based on the difference $L_{FN} - L_{AM}$. A non-negative value indicates the need to increase the disparity in signal arrival times at the gate, while a negative value necessitates bringing the arrival times closer together to achieve glitch reduction. This enhancement or reduction is achieved by gate sizing while ensuring compliance with the timing constraints of the AM circuit.

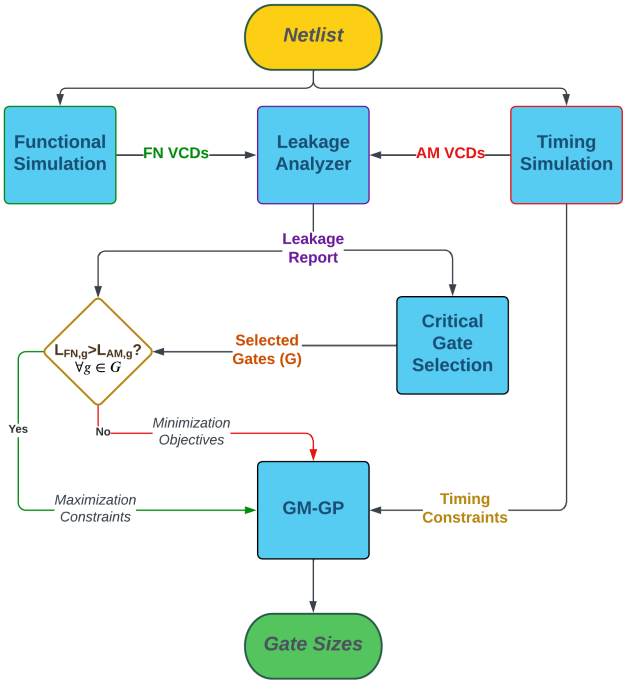


Fig. 2: GLiTCH manipulation framework.

A. Leakage Analysis

To enhance side-channel resilience in designs, we propose a systematic approach for identifying and mitigating critical leakage points within the design. This process begins by evaluating the leakage characteristics of the design of both the functional and timing simulations. Each module in the design is analyzed individually, with power leakage quantified by examining signal transitions and their correlation with secret-dependent operations. The leakage analyzer operates in multiple phases: it first captures the signal transition traces (**V**alue **C**hange **D**ump files) from the functional and timing simulations and correlates these traces with the secret dependent operations to assess leakage. A leakage metric is computed based on this correlation, identifying the gates that contribute most significantly to leakage. The gates are then ranked according to their leakage impact, enabling the prioritization of countermeasures at the most critical points in the design's netlist.

B. Glitch Manipulation using Geometric Programming

We motivated the need for glitch manipulation using gate sizing in Section IV. In general, a GP optimization problem is mathematically formulated as follows:

$$\begin{aligned} \min \quad & f_0(\mathbf{x}) \\ \text{subject to} \quad & f_i(\mathbf{x}) \leq 1, \quad i = 1, 2, \ldots, m \\ & g_j(\mathbf{x}) = 1, \quad j = 1, 2, \ldots, p \end{aligned} \tag{1}$$

where f_i are posynomial functions, and g_j are monomial functions [18]. Glitch manipulation (GM) can be categorized into two primary approaches: glitch addition and glitch removal. Within the selected critical gates, as outlined in the proposed framework, the choice to introduce additional glitches or to reduce existing ones is determined using the leakage analyzer. The GP formulation for both are discussed below:

1) Glitch Addition: We use the inertial delay model for the gates, according to which a gate does not respond to a signal if the next incoming signal (at the same input or another input) arrives within one inertial delay of the gate. Consequently, by ensuring that the difference in arrival times of signals at the gate inputs exceeds the inertial delay, the likelihood of generating additional glitches at the gate's output can be increased. For example, in Figure 1, the NAND2 gate requires $|a_1 - a_2| \geq d_{inert}$ for the gate to generate glitches, where d_{inert} is the inertial delay of the NAND2 gate. Without loss of generality, it is assumed $a_2 \geq a_1$. The equation can be rewritten as follows:

$$\frac{a_1 + d_{\text{inert}}}{a_2} \leq 1. \tag{2}$$

Equation (2) represents a valid posynomial expression that is suitable for inclusion in a geometric programming framework, as it adheres to the structural requirements of the formulation as outlined in 1.

2) Glitch Removal: To implement glitch removal, the inequality in equation (2) is to be reversed. However, this modification introduces a structural incompatibility with the formulation of the optimization problem as defined in equation (1). To ensure glitch removal, it is essential to minimize the difference in arrival times at the gate inputs. For the scenario depicted in Figure 1, this corresponds to reducing $(a_2 - a_1)$ so that the arrival times come close together. An objective compatible with the GP formulation for glitch removal [14] is as follows:

$$\min \left(\frac{a_2}{a_1}\right)^2, \quad \text{subject to constraint } a_1 \leq a_2.$$

Putting both the objective for the glitch removal, which can be extended for gates with more than two inputs [14] and the GP constraint for glitch addition into the formulation, we get the following set of equations:

The area minimization of the circuit, which is used as the baseline for timing simulations, can be realized by replacing the objective function with $\min(\sum_{i \in G} x_i)$ and retaining the AM-GP constraints from the above set of GM-GP formulation. The AM-GP solution provides the gate sizes that minimize area to meet a given timing specification.

Let,

C :	Set of all critical gates for glitch removal
A_c :	Set arrival times of critical gate (c) inputs for glitch removal
A_m :	Set arrival times at critical gate (c) inputs for glitch addition
A_i :	Set of all arrival times of the inputs to gate i
G :	Set of all gates
O_i :	Set of gate sizes of the fanout of gate i
g_i :	Logical effort of gate i
p_i :	Parasitic delay of gate i
P_O :	Set of all gates with a primary output
x_{max} :	Maximum size available in the library
T_{spec} :	Timing specification for the circuit

GM-GP:

$$\min \quad \sum_{c \in C} \sum_{m \neq n} \left(\frac{a_n}{a_m} \right)^2$$

subject to

$$a_m \leq a_n$$

$$\frac{a_x + d_{inert}}{a_y} \leq 1$$

AM-GP constraints ←

$$a_l + g_i \left(\frac{\sum_{x_j \in O_i} g_j x_j}{g_i x_i} \right) + p_i \leq a_i$$

$$a_k \leq T_{spec}$$

$$x_i \leq x_{max}$$

$$x_i \geq 1$$

$$\forall\, a_l \in A_i\ ,\ \forall\, i \in G,\ \forall\, k \in P_O$$

$$\forall\, a_m, a_n \in A_c$$

$$\forall\, a_x, a_y \in A_m\ \&\ a_y \geq a_x$$

VI. Experiments and Discussion

We validate the GM-GP method on widely-used cipher designs like AES, CLEFIA, and SM4, with a particular focus on their S-Boxes, which are well-established as the primary source of leakage [19]. We conducted functional and timing simulations for each S-Box, selected critical gates in different configurations, determined their objectives, and applied the Glitch Manipulation methodology outlined in Section V.

A. Methodology

The total power and glitch count were analyzed using $1,000$ uniformly sampled input vectors. These ciphers were synthesized using Cadence while employing the TSMC 28-nm standard cell library. The timing specification (T_{spec}) was carefully chosen between 1.2 and $1.4 \times T_{\text{wall}}$, where T_{wall} is the fastest achievable timing for the circuit [14]. Setting T_{spec} beyond this range would result in most gates becoming unit sizes. Simulations were performed on a 3.6 GHz Intel i7 system with 32 GB of memory running CentOS. The optimization framework was implemented in Python, utilizing the GPkit library [20] alongside the Mosek solver [21].

The results presented in the paper validate the insights derived from the motivating example in Section IV, highlighting the critical role of glitches in leakage differences. In the motivating example, we demonstrated how variations in signal arrival times at a gate lead to correlated or uncorrelated transitions, directly influencing the leakage. The results obtained across FN, AM, and GM configurations provide empirical evidence supporting this explanation. By keeping the circuit structure and the number of gates constant across FN, AM, and GM, the results isolate glitches as the sole contributor to leakage variations. In addition, we assessed the Guessing Entropy (GE) [22] of the secret key for each scenario (FN/AM/GM), quantifying the number of attempts an attacker would need to correctly guess the key irrespective of the guessing strategy. This metric provides insights into the circuit's resilience to side-channel attacks in practical scenarios. Higher GE indicates stronger resistance, with an 8-bit S-Box limiting the range to integers between 1 and 256.

B. Results

The results of this evaluation are summarized in Table II, providing insights into the impact of gate modifications on the design across different configurations. The table details key parameters, including the number of modified gates, changes in glitch occurrences, the number of gates affected, the resulting Guessing Entropy (and therefore the DA), and the percentage increases in area and power consumption. Each entry for a chosen cipher corresponds to a varying choice of gates for glitch modification, allowing for a comparative analysis of their performance and security.

For instance, in the first configuration of the AES cipher, starting with a netlist of 195 gates, the design was altered by introducing 7 gate constraints for maximization and 5 gate objectives for minimization relative to the AM configuration. Maximization here refers to pulling apart signal arrival times to create intentional delays, while minimization aims to align arrival times more closely. These adjustments led to an increase of $8,446$ glitches, with 115 gates experiencing a rise in glitches at their output and 69 gates showing a reduction. As a result, the GE rose to 232 from an initial 75, reflecting a 61.33% improvement. This increase in GE implies a lower DA, which correlates with reduced vulnerability to side-channel attacks. Notably, while this configuration incurs some area and power overheads (40.56% and 51.34%, respectively), it achieves the highest level of side-channel security among the evaluated setups, making it the most effective in mitigating leakage. Subsequent configurations, outlined in other rows of the table, present varying trade-offs in terms of the GE, DA, area, and power depending upon the decision made during the critical gate selection phase. Here, the first configuration, highlighted in green, demonstrates the best balance between security improvement and the incurred overhead, solidifying its role as the optimal choice for secure cryptographic hardware.

While the GE indicates the global effects of the modification, it is also interesting to study the impact of the changes on the leakage at a gate level. Figure 3 shows a comparison

TABLE II: Results of the proposed framework on various S-boxes indicating the cipher, Number of gates modified, Change in number of glitches, Guessing Entropy in DPA and % Area and Power overheads

Cipher	# Gates	# Gates Maximized	# Gates Minimized	Δ Glitches (GM - AM)	# Gates Glitch ↑	# Gates Glitch ↓	Guessing Entropy			% Area ↑	% Power ↑
							FN	AM	GM		
AES	195	7	5	8446 ↑	115	69	53	75	232	40.56	51.34
		0	5	45370 ↓	25	158			229	36.74	11.05
		1	0	26024 ↑	122	62			217	36.29	64.72
CLEFIA	156	4	0	18592 ↑	99	22	76	68	228	41.91	70.1
		4	0	18670 ↑	99	22			159	42.02	70.51
		5	0	15610 ↑	93	24			141	41.69	60.41
SM4	128	1	0	29914 ↓	29	95	214	105	209	34.62	7.54
		1	0	25682 ↓	42	83			205	35.76	29.85
		1	1	31250 ↓	36	89			154	39.09	27.89

of leakage scores for three different settings - FN, AM, and GM - across the range of signals present in the AES S-Box. The x-axis represents the signals with a select few marked for readability, while the y-axis shows the leakage score. The zoomed-in section of Figure 3 highlights the local and global effects of glitch modification on circuit behavior. Gate 1 and Gate 3 show a significant decrease in leakage scores, indicating a dominance of uncorrelated glitches. In contrast, Gate 2 exhibits a slight increase in the GM leakage score, suggesting a marginal rise in correlated glitches. This suggests that while glitch modification can lead to a reduction in overall glitches in some parts of the circuit, it can also result in an increase in correlated glitches in other parts. This can be attributed to the addition or removal of glitches in the fan-in or fan-out cones of the chosen gates due to changes in arrival times.

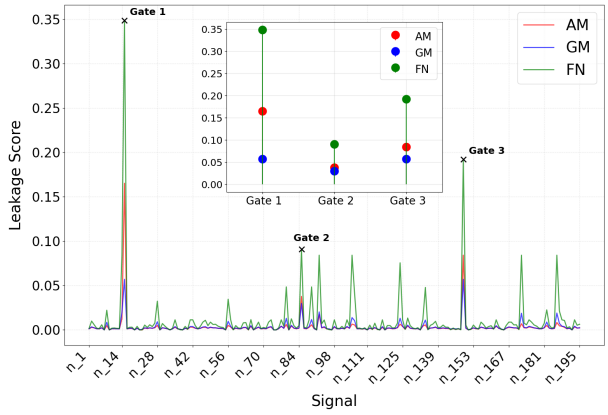


Fig. 3: AES Leakage Comparison

C. Discussion

The GM configuration, on average, incurs a 38.74% area overhead. Power, on the other hand, is dependent on the activity factor of each gate and its corresponding load capacitance. This is largely affected by the choice of critical gates, and hence, we observe that the power overhead ranges from $7 - 70\%$ across the various cipher implementations. These overheads are accompanied by a notable reduction (52.82% on average) in the leakage and enhanced resilience, suggesting its potential suitability for secure cryptographic hardware. This result highlights the framework's ability to facilitate the design of hardware with stronger defenses against side-channel attacks. In this paper, we have summarized the results for $8-$bit S-Boxes. However, we have also evaluated ciphers with smaller S-Boxes like PRESENT and ASCON5. While we observed some reduction in the leakage, it came at the cost of large areas and power overheads. This can be attributed to the minimal number of re-sizable gates for the GM-GP formulation. For example, ASCON5 gave only a 9.38% improvement in the GE at the cost of 113.25% area and 104.27% power overheads.

Further, in this work, we consider relatively smaller circuits ($128 - 195$ gates) than a full cipher ($\sim 30k$ gates) to demonstrate a Proof-of-Concept. In general, the computational complexity of the AM-GP formulation scales as N^{λ} [23], where N is the number of gates in the circuit, and λ typically ranges between 1 and 2, depending on the circuit. Two kinds of constraints on a handful of gates are appended to the AM-GP formulation to incorporate the GM-GP formulation. As a result, the overall time complexity remains largely unaffected and should scale to a million gates as reported in [23].

VII. Conclusion and Future Work

The GM configuration demonstrates a significant improvement in leakage reduction through targeted gate sizing. The results clearly demonstrate that the GE in the GM circuit is significantly higher compared to the AM circuit. Locally, the modifications directly impact the gates involved by altering their arrival times. Globally, these changes propagate throughout the entire fan-out network of the corresponding outputs in the circuit as the modified gates generate and transmit glitches. This can increase uncorrelated glitches across the circuit, which helps reduce side-channel leakage. The results show a 52.82% improvement in the GE with the 38.74% area overhead on average. Future directions involve understanding the global effects of the modifications on the leakages through the fan-in and fan-out cones. This insight would enhance the critical gate selection process.

VIII. Acknowledgment

The authors thank Arvind T. K. R for his work and support. The authors also thank MietY (Ministry of Electronics & Information Technology) for the C-HERD (Centre on Hardware-Security Entrepreneurship Research & Development) project, and CSR from IDBI for the funding. The authors also thank the reviewers for their time and effort in evaluating the work.

REFERENCES

[1] Farid N Najm. A survey of power estimation techniques in vlsi circuits. *IEEE transactions on very large scale integration (VLSI) systems*, 2(4):446–455, 1994.

[2] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[3] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002.

[4] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171. Springer, 2005.

[5] Masanori Hashimoto, Hidetoshi Onodera, and Keikichi Tamaru. A power optimization method considering glitch reduction by gate sizing. In *Proceedings of the 1998 international symposium on Low power electronics and design*, pages 221–226, 1998.

[6] Lei Wang, Markus Olbrich, Erich Barke, Thomas Büchner, Markus Bühler, and P Panitz. A gate sizing method for glitch power reduction. In *2011 IEEE International SOC Conference*, pages 24–29. IEEE, 2011.

[7] Luca Benini, Giovanni De Micheli, Alberto Macii, Enrico Macii, Massimo Poncino, and Riccardo Scarsi. Glitch power minimization by selective gate freezing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(3):287–298, 2000.

[8] Tezaswi Raja, Vishwani D Agrawal, and Michael L Bushnell. Variable input delay cmos logic for low power design. *IEEE transactions on very large scale integration (VLSI) systems*, 17(10):1534–1545, 2009.

[9] Hyeonmin Lim, Kyungsoo Lee, Youngjin Cho, and Naehyuck Chang. Flip-flop insertion with shifted-phase clocks for fpga power reduction. In *ICCAD-2005. IEEE/ACM International Conference on Computer-Aided Design, 2005.*, pages 335–342. IEEE, 2005.

[10] Shivani Bathla, Rahul M Rao, and Nitin Chandrachoodan. A simulation-based metric to guide glitch power reduction in digital circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(2):376–386, 2018.

[11] Svetla Nikova, Christian Rechberger, and Vincent Rijmen. Threshold implementations against side-channel attacks and glitches. In *International conference on information and communications security*, pages 529–545. Springer, 2006.

[12] Thomas De Cnudde, Begül Bilgin, Oscar Reparaz, Ventzislav Nikov, and Svetla Nikova. Higher-order threshold implementation of the aes s-box. In *Smart Card Research and Advanced Applications: 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers 14*, pages 259–272. Springer, 2016.

[13] F.N. Najm. A survey of power estimation techniques in vlsi circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):446–455, Dec 1994.

[14] Karthikeyan Muthamizh Vithagan, Vignesh Sundaresha, and Janakiraman Viraraghavan. Geometric programming approach to glitch minimization via gate sizing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(6):1988–2001, 2023.

[15] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[16] Patanjali Slpsk, Prasanna Karthik Vairam, Chester Rebeiro, and V. Kamakoti. Karna: A gate-sizing based security aware eda flow for improved power side-channel attack protection. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, 2019.

[17] Jitendra Bhandari, Likhitha Mankali, Mohammed Nabeel, Ozgur Sinanoglu, Ramesh Karri, and Johann Knechtel. Beware your standard cells! on their role in static power side-channel attacks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[18] S Boyd, SJ Kim, L Vandenberghe, and A Hassibi. A tutorial on geometric programming, optimization and engineering, 2007.

[19] Sylvain Guilley, Philippe Hoogvorst, and Renaud Pacalet. Differential power analysis model and some results. In Jean-Jacques Quisquater, Pierre Paradinas, Yves Deswarte, and Anas Abou El Kalam, editors, *Smart Card Research and Advanced Applications VI, IFIP 18th World Computer Congress, TC8/WG8.8 & TC11/WG11.2 Sixth International Conference on Smart Card Research and Advanced Applications (CARDIS), 22-27 August 2004, Toulouse, France*, volume 153 of *IFIP*, pages 127–142. Kluwer/Springer, 2004.

[20] Edward Burnell, Nicole B Damen, and Warren Hoburg. Gpkit: A human-centered approach to convex optimization in engineering design. In *Proceedings of the 2020 chi conference on human factors in computing systems*, pages 1–13, 2020.

[21] MOSEK ApS. Mosek optimization suite. 2017.

[22] J.L. Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, pages 204–, 1994.

[23] Stephen P Boyd, Seung-Jean Kim, Dinesh D Patil, and Mark A Horowitz. Digital circuit optimization via geometric programming. *Operations research*, 53(6):899–932, 2005.

A Novel Image-Graph Heterogeneous Fusion Framework for Static IR Drop Prediction

1st Dan Niu
School of Automation
Southeast University
Nanjing, China

2nd Dekang Zhang
School of Automation
Southeast University
Nanjing, China

3rd Yichao Cao
School of Automation
Southeast University
Nanjing, China

4th Zhou Jin
School of Integrated Circuits
Zhejiang University
Hangzhou, China

5th Chao Wang
School of Automation
Southeast University
Nanjing, China

6th Yichao Dong
School of Automation
Southeast University
Nanjing, China

7th Changyin Sun
School of Artificial Intelligence
Anhui University
Hefei, China

***Abstract*—IR drop analysis is crucial for ensuring the reliability and performance of integrated circuits (ICs) but poses computational challenges as the IC designs grow larger, especially for ultra deep-submicron VLSI designs. Deep learnings (DL) as the efficiency-promising solutions, mainly employ various CNN-based networks to achieve image-to-image IR drop predictions. However, they neglect and lose the power delivery network (PDN) global spatial features and cell instance topological information. This paper proposes a novel image-graph heterogeneous fusion framework (IGHF), which integrates the effectiveness and complementarity of dual branches (CNN and GNN) for higher prediction performance. In the CNN-based Power ScaleFusion Unet branch, the proposed long-range and local-detail encoder (LLE) integrates seamlessly with the hierarchical and adjacent compensation group (HACG) module. This design facilitates effective multi-scale global-to-local spatial power feature extraction within the PDN and enables adaptive high-to-low-level feature fusion and compensation in the decoder. Moreover, a cell voltage aware (CVA) module in the GNN branch is designed to adaptively aggregate PDN topological features of heterogeneous neighbors of different orders. Comparative experiments demonstrate that the proposed IGHF achieves significant accuracy improvements, outperforming the state-of-the-art MAUNet and widely-used IREDGe methods by considerable margins of 24.6% and 55.0% reduction in prediction error, while the prediction maps possess higher structural fidelity. Transfer experiments indicate that IGHF with transfer learning can improve the accuracy in real circuits with the few-shot real circuit test cases.**

***Index Terms*—IR drop prediction, Convolutional neural network, Graph neural network, Heterogeneous fusion.**

I. Introduction

IR drop analysis refers to obtaining the voltage drops caused by the supply current flowing through the power delivery network (PDN) [1]. The demand for IR drop analysis is gradually increasing due to the lower noise margin required for power reliability in low-power designs. It is crucial due to its close relation to chip performance, reliability, and functionality [2]. As the chip integration density escalates, traditional simulation-based IR drop analysis requires formidable computational and memory costs owing to high-dimensional circuit equations [3], [4]. Moreover, various methods including the multi-grid method [5], the boundary element method [6], and the random walk algorithm [7] are designed to mitigate the computational burden. However, the computational overhead of matrix solution operations remains substantial, especially for irregular and non-uniform power grids [8].

Currently, convolutional neural network (CNN)-based deep learning methods provide efficient and promising solutions for full-chip IR drop prediction [9]-[13]. IREDGe [12] introduces the pad distribution feature and employs encoder-decoder architecture for pixel-level prediction. In [13], MAUNet integrates multi-scale convolutional blocks, attention mechanisms, and U-Net to optimize IR drop prediction accuracy. However, due to the constraints imposed by the limited size of convolution kernels, these approaches primarily emphasize the IR drop effects induced by local-area cell instances, making it challenging to capture the global IR drop patterns, particularly the distribution of cell current and resistance [14], [15]. The sizes of the kernels for spatial feature extraction in CNN pose challenges. Small windows risk violating the locality principle of IR drops, whereas larger windows for the global receptive field are information-leaking, memory-intensive, and time-consuming [16]. In this case, *how to improve CNNs to capture both the global and local PDN spatial features for higher IR drop prediction accuracy is vital and under exploration*.

In addition, not only do the current/instance load patterns impact the IR drop distribution, but the PDN topologies also contain the actual cell current flow and interference information, which are indispensable for IR drop [9], [17]. However, CNNs primarily capture image-like representations from current load patterns. While they lack topology perception mechanisms [18], and a single image pixel may contain multiple cell instances, which loses instance information that affects the hotspot area [19]. Therefore, it's a challenge to represent the practical non-Euclidean PDN connectivities effectively [20]. It's essential to introduce rich and supplementary PDN topological features to address the problem of inconsistent correspondence between images and cell instances. Currently, graph neural networks (GNNs) have been widely applied to

capture the circuit topology, addressing issues such as routing congestion and power performance area (PPA) optimization [21], [22]. However, *how to introduce GNN to effectively incorporate the PDN topological features for higher IR drop prediction performance remains a novel challenge*.

This work proposes an image-graph heterogeneous fusion framework (IGHF) that integrates a CNN-based branch with a GNN-based branch. The CNN-based branch employs a Power ScaleFusion Unet, while the GNN-based branch constructs a cell voltage aware module. The main contributions include:

- A novel image-graph heterogeneous fusion framework (IGHF) is proposed, which integrates a CNN-based branch to capture multi-scale global-to-local PDN spatial features and a GNN-based branch to perceive the non-Euclidean connection relation in the PDN topology.
- In the CNN branch, a long-range and local-detail encoder (LLE) is proposed to integrate the long-distance regional superposition effects and local instance-specific current characteristics. Moreover, a hierarchical and adjacent compensation group (HACG) module is designed to adaptively rescale the multi-scale PDN spatial features for guiding a high-to-low-level feature fusion.
- In the GNN branch, the cell heterogeneous edge construction is designed to ease information propagation. Moreover, a cell voltage aware (CVA) module is proposed to adaptively aggregate PDN topological features from heterogeneous neighbor cells of different orders.

II. Preliminaries

IR drop analysis aims to compute the voltage drops induced by the supply current flowing through the PDN. It is obvious that both the current load patterns and the PDN architectures affect analysis accuracy [23]. The precise IR drop analysis is crucial since voltage overshoot can lead to irreversible device damage and chip malfunction.

For the full-chip IR drop analysis, a prevailing method in commercial software is Modified Nodal Analysis (MNA), where the Kirchhoff Current Law is employed to formulate a system of linear equations. The system is characterized by a conductance matrix G for the PDN, a node voltage vector V, and a current source vector J, represented by $GV = J$ [24].

However, the system can rapidly balloon to millions of voltage nodes and interconnect segments in modern chip designs as the integration density escalates, which makes the analysis both computationally intensive and memory-demanding [25]. Therefore, machine learning-based IR drop predictions are introduced to accelerate the process.

III. The Proposed Method

In this work, our DL-driven IR drop prediction method employs a heterogeneous dual-branch network, as shown in Fig. 1. It integrates a CNN-based branch for multi-scale global-to-local spatial power feature extraction and a GNN-based branch to complement PDN topological information, which significantly boosts the prediction performance.

A. Input Image and Graph Construction

Image Construction: The inputs of the CNN branch are the feature images I according to the mapping of spatial cell coordinates, which are extracted from raw layout and netlist files. To enhance analysis convergence and reduce redundant features, the Permutation Importance (PI) filter proposed in [26] is employed. This method randomly permutes the order of the original input images I and identifies features as critical for the analysis if they result in significant changes in model accuracy upon permutation, with a correlation coefficient importance $\geq \alpha$. In this work, the power features (total power, leakage power), representing current spatial distribution, and the resistance feature (min. path resistance), representing the worst-case PDN current condition, are employed. In contrast, we exclude toggle power, internal power, switching power, effective path resistance, and cell frequency.

Graph Construction: An unweighted bidirectional heterogeneous graph $G = (V, E, R)$ is constructed based on the .dspf files to represent the practical non-Euclidean PDN topology, as illustrated in Fig. 1. In this graph, the set V comprises all cell instances v, while the set E represents the connections e between them. The set R encompasses all types of relationships between v (the following R_I and R_d). The above-selected power and resistance features are assigned to V's attributes. Next, heterogeneous edge relations are constructed according to two criteria. The first is the original PDN instance connection relation R_I, representing the voltage propagation by the practical current flowing. The second is the designed cell distance-aware edge relation R_d, which adds the physical-inexistent edges considering the spatial adjacencies (the Euclidean distance $||v_i, v_j||_2 \leq \delta$ for the instances v_i, v_j). The virtual edges ease the information propagation in GNN. Thus, R consists of relations R_I and R_d. The heterogeneous graph G and outputs of CNN are used as the input to GNN.

B. CNN Branch: Power ScaleFusion Unet

The CNN branch adopts the Unet architecture. First, the long-range and local-detail encoder is proposed to achieve multi-scale global-to-local spatial power feature extraction. In decoder, the hierarchical interaction compensation group is designed to adaptively rescale and guide hierarchical feature fusion and compensation.

1) Long-Range and Local-Detail Encoder (LLE): Due to the localized IR drop phenomena caused by nearby cells and the global IR drop phenomena resulting from regional activity, IR drop at each location relies on both local and global power information. Therefore, the LLE, combining long-range modules and local-detail modules, is proposed to capture the multi-scale long-distance and local-detail PDN power and current load characteristics, as shown in Fig. 1.

Long-Range Module (LRM): The LRM in LLE serves as a global PDN extractor, which achieves the long-distance spatial power characteristic awareness through the long-range block (LRB), as shown in Fig. 2 (a). For LRB, inspired by the Swin Transformer [27], it is divided into a perception network and a convolutional feed-forward network (FFN). In the perception

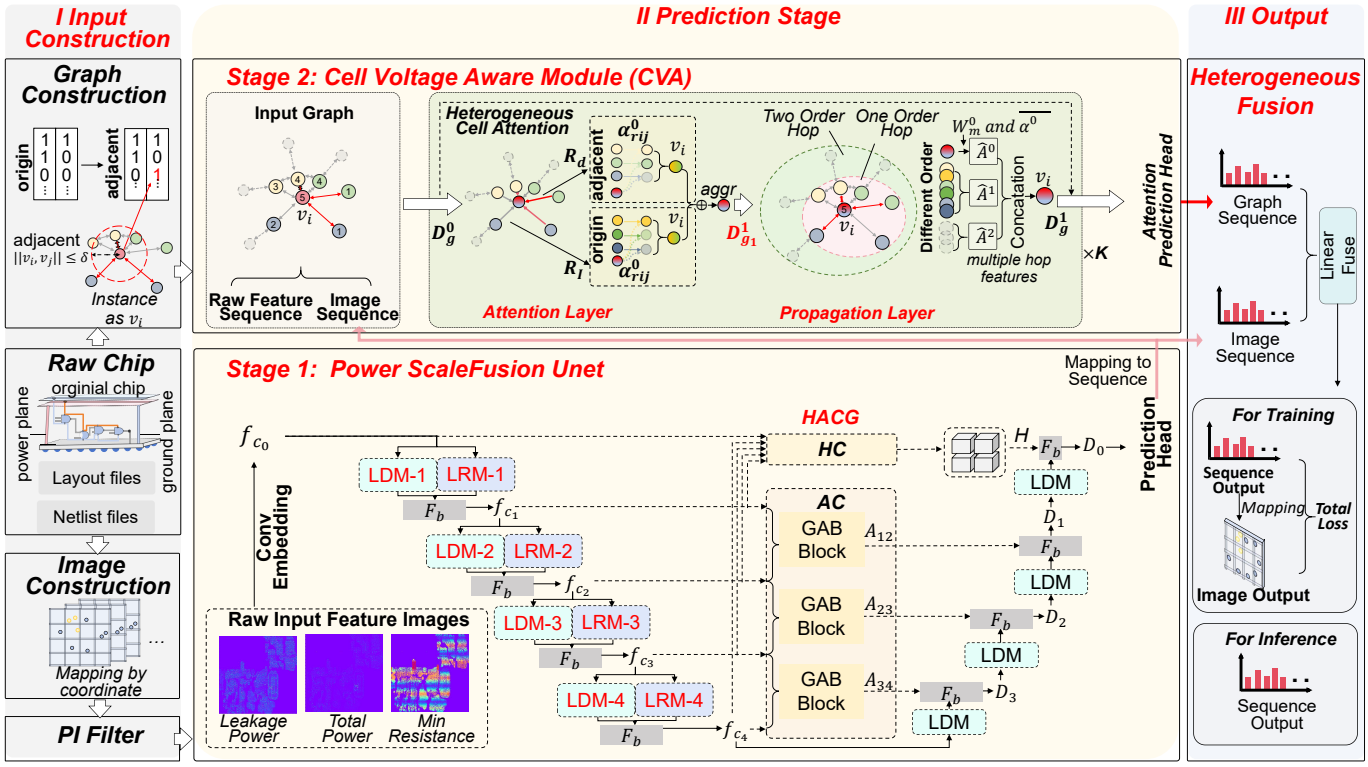


Fig. 1: Proposed IGHF framework for IR drop prediction. First, raw SoCs are transformed into images and graphs in the left part. Then the GNN-based branch (the upper in middle part) and CNN-based branch are proposed to extract the PDN topological features and multi-scale global-to-local spatial features from the graphs and images, respectively. Finally, the aligned GNN features and the CNN features are fused to generate the final prediction in the right part.

network, the large kernel depthwise (DW) convolution [28] is introduced to significantly improve the effective receptive fields to contain broader power and current distribution information. The design of large convolution kernels, as opposed to attention mechanisms in the Swin Transformer, is better suited for addressing problems with limited circuit data due to their inherent inductive bias. Additionally, large kernels help mitigate optimization challenges associated with increasing model depth when relying on stacked small kernels.

Furthermore, to mitigate the memory-intensive costs of LRB due to the large kernel design, LRB is designed as an inverted bottleneck residual structure as a whole [29]. The inefficient large-kernel convolutions are designed with fewer channels, whereas the more efficient and dense 1×1 layers within the FFN are responsible for managing the heavy computational workload, including feature dimensionality and the model's non-linearity. Moreover, multi-level LRMs allow model to perceive various-scale spatial power characteristics.

Local-Detail Module (LDM): Different from the LRM for long-distance power features, the LDM with small kernel convolutions focuses on encoding fine-grained PDN spatial features and obtaining distinct small-scale current load characteristics of standard cells. As shown in Fig. 2 (b), it employs more non-linear Leaky ReLU (LReLU) to enhance encoding diversity. Moreover, the stacking LDMs further extract multi-scale local-detail power features.

Branch Fusion F_b: The long-range dependencies and fine-grained information from each LRM and LDM are superposed by the branch fusion layer. In this work, the small kernel convolution operation is chosen as the fusion operation F_b. The fusion layer is updated by Eq. (1), where $f_{c_i}(i = 1, \ldots, 4)$ denotes the encoder feature map.

$$f_{c_i} = F_b\{LDM\text{-}i(f_{c_{i-1}}), LRM\text{-}i(f_{i-1})\} \tag{1}$$

2) Hierarchical and Adjacent Compensation Group (HACG): The multi-scale power encoder features contribute differently to IR drop due to significant variations in PDN size and cell power density [30], [31]. Different from the simple skip connections in vanilla Unets, the HACG in the decoder is proposed. It combines a hierarchical compensation and adjacent compensation module to achieve the different-scale encoder feature map fusion interactions and compensate the decoder information degradation through the lightweight group aggregation bridge (GAB), as shown in Fig. 3. It enables the fusion of muti-scale information at the final decoding stage and reduces sampling losses at different scales during the upsampling process.

Hierarchical Compensation: First, the hierarchical compensation (HC) is designed for the top decoder layer to integrate feature maps with different semantics at each scale. Owing to the varying receptive fields, it is necessary to superimpose hierarchical features to model the trend of voltage

fluctuations more comprehensively. The top decoder combines the output of the top encoder with those of the other encoders through multiple dilated convolution groups within the HC module. This design effectively leverages the contributions of different-scale encoder semantics to enhance the representation of node voltages. The HC output H is computed as Eq. (2).

$$H = HC(f_{c_0}, \cdots, f_{c_N}) = \sum_{i=1:N} GAB(f_{c_0}, f_{c_i}) \quad (2)$$

where N denotes the total number of high-level encoded features to integrate. $GAB(f_{c_0}, f_{c_i})$ utilizes depthwise separable convolution and bilinear interpolation to align low-level and high-level feature sizes f_{c_0} and f_{c_i} and utilizes four different dilation convolution groups to extract information adaptively.

Adjacent Compensation: The adjacent compensation (AC) module primarily aims to compensate for the inevitable local information degradation of the downsampling layers. $A_{i,i+1} = GAB(f_{c_i}, f_{c_{i+1}})$ performs GAB interaction on adjacent encoder maps $f_{c_i}, f_{c_{i+1}}$ to adaptively rescale and compensate the corresponding decoder layer D_{c_i}, which guides a high-to-low level IR drop prediction. Different from the standard skipping connections between the corresponding encoder and decoder layers, AC reduces the upsampling loss of features between adjacent layers.

C. GNN Branch: Cell Voltage Aware Module (CVA)

The GNN-based branch utilizes the CVA for PDN topology feature extraction to consider the intricate topological connections and adjacent effects. CVA consists of cascade K residual blocks, the block is composed of an attention layer and a propagation layer. The CVA adaptively learns the influence weights of heterogeneous connections from neighboring nodes to the central node in the attention layer. Moreover, it combines voltage-related features from neighbors at varying distances in the propagation layer, as shown in Fig. 1.

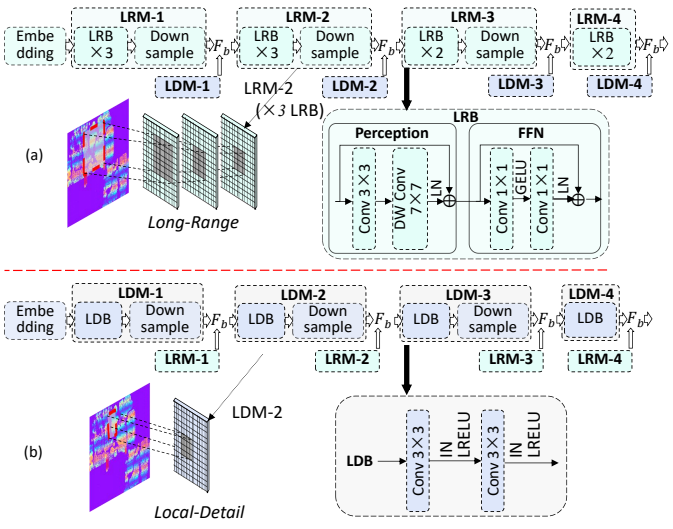


Fig. 2: The architecture of LLE. The bottom left corner contrasts the receptive field sizes for LRM (a) and LDM (b). The rest is the detailed LRM and LDM. Here L is 1.

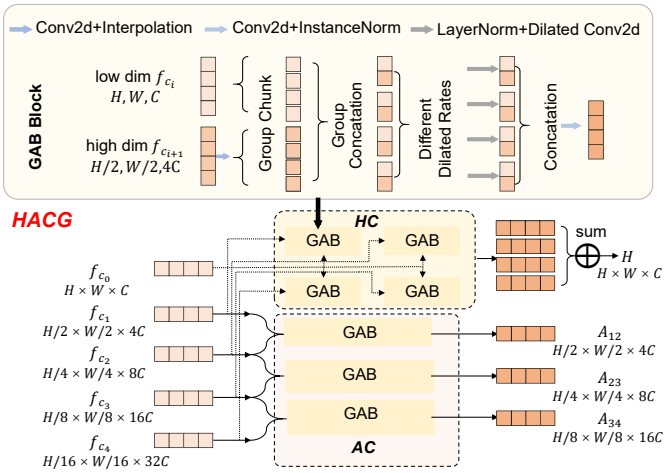


Fig. 3: The architecture of the HACG. The top yellow part denotes the GAB block. And the rest is the HACG consisting of HC and AC submodules. H, W, C represent the height, width, and number of channels of the features, respectively. Here dilated rates of convolution groups are 1, 2, 5, and 7.

Attention Layer: The virtual edge and the actual current transmission edge exhibit heterogeneous characteristics in their impact on the voltage of the central cell node. And the cell instance connection edges E cause complex interactive relationships due to discriminative distances or resistances between nodes. To adaptively rescale and aggregate significant information from heterogeneous edge connections, the heterogeneous cell attention mechanism is designed as Eqs. (3) and (4). It learns the critical interactions between nodes in heterogeneous edge type r, using the attention factor α^l_{rij}. Then all outputs are aggregated into $D^l_{g_1}$. Moreover, the average of α^l_{rij} is also used as the attention factor of propagation layer for two heterogeneous edge connection weights.

$$D^l_{g_1} = \sum_r LReLU(\alpha^l_r \circ (\hat{W}^l_r A_r D^l_g)) \quad (3)$$

$$\alpha^l_{rij} = softmax(\frac{\exp(e^l_{rij})}{\sum_{j \in N_i} \exp(e^l_{rij})}) \quad (4)$$

where D^l_g and $D^l_{g_1}$ is the input and output of the l_{th}-layer attention layer, r denotes the edge type in the relation set R, $\hat{W}^l_r$ is the linear parameter in type r, N_i represents neighbor set of v_i. α^l_{rij} as an element of matrix α^l_r, is the attention weight of v_i to v_j in type r. e^l_{rij} is the attention score between v_i and v_j in type r, using Eq. (2) of [32].

Propagation Layer: The voltage fluctuation of a node v_i is considered as the superposition of multiple surrounding cell nodes. This layer learns voltage representation through adjacent virtual and practical physical connections from neighbors. Different-order neighbors influence the central node to varying degrees. Vanilla graph convolutional networks (GCNs) only aggregate first-order neighborhood information, neglecting higher-order neighborhood data [33]. Consequently, these methods are suitable only for graphs where the first-order

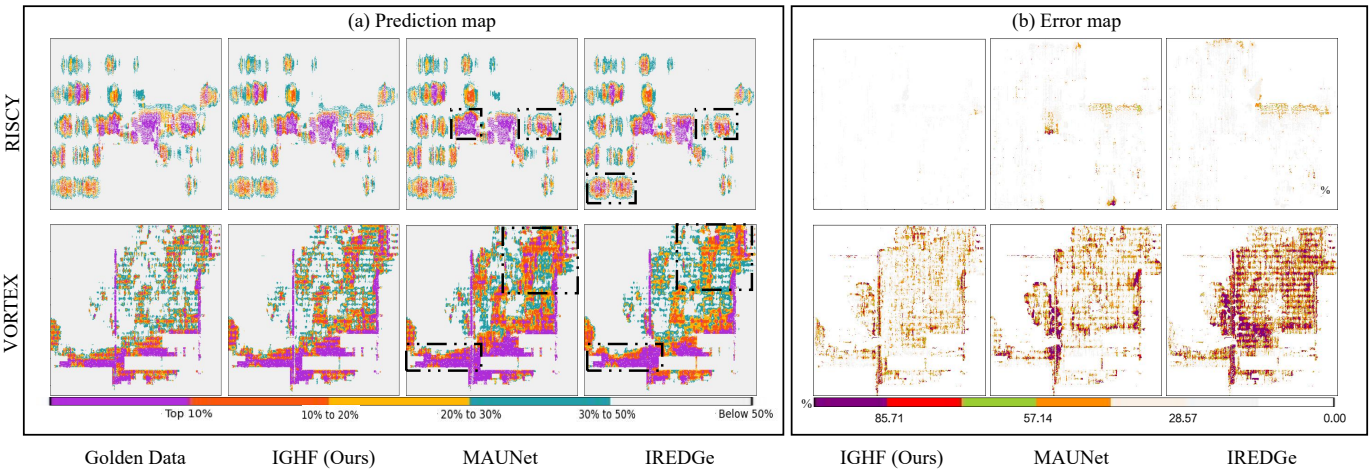


Fig. 4: two visual cases for RISCY (44,855 cell instances for CPU chip) and VORTEX (108,028 cell instances for GPU chip). (a) Prediction maps: The top ***10%*** and ***20%*** of the IR drops are highlighted as ***purple*** and ***red*** hotspots. The top ***30%*** and ***50%*** of the IR drops are indicated as ***yellow*** and ***green*** regions. (b) Relative error maps between golden data and two predictions.

neighbors exhibit high compatibility with the central node v_i. This limitation renders them ineffective for graphs where the first-order neighbors are incompatible with the central node v_i, but higher-order neighborhoods demonstrate compatibility.

To address this, MixHop is employed [33]. It learns the relationships among different cells by repeatedly mixing the voltage-related features of neighbors at varying distances and incorporating different operators. In this approach, graph convolution as described in Eq. (5) can represent the information aggregation of neighbors spanning two order hops. MixHop's node update method can extract linear combinations of neighbor features from a farther distance, allowing it to capture meaningful signals even in extremely sparse connections. Eq. (5) aggregates the information of self-loops, first-order and second-order neighbors through the average attention factor $\bar{\alpha}^l$ in attention layer and its own learnable weight W_m^l to form the output of the l-th layer, where $m = 0, 1, 2$.

$$D_g^{l+1} = LReLU\left(\left[\bar{\alpha}^l I D_{g_1}^l W_0^l \mid \bar{\alpha}^l \hat{A} D_{g_1}^l W_1^l \mid \bar{\alpha}^l \hat{A}^2 D_{g_1}^l W_2^{l)}\right]\right) \tag{5}$$

where $\hat{A}$ is a adjacency matrix with self-connections, $D_{g_1}^l$ and D_g^l is the input and output of the l-th layer, I is the diagonal degree matrix, W_m^l is the learnable weight. $\mid$ denotes concatenation. And after passing through K residual blocks, the graph instances are processed by the GAT prediction head to produce the final graph sequence.

Fusion: To generate the final IR drop prediction sequence, the final graph sequence is linearly fused with the CNN image output, leveraging the node image sequence aligned with the cell instance coordinates.

IV. Experiments and Results

A. Experiments Setting and Results

The experiment was conducted on an Ubuntu computer with a GeForce RTX 3090 GPU, using the public CircuitNet-ICISC benchmark dataset. This dataset includes over 7,000 samples from five 28-nm open designs, each with varying layouts, 54 synthesized circuit netlists (with different timing constraints and macros), and cell utilizations [34]. The training to test set ratio was 8:2.

We use an L1/L2 loss with Adam optimizer, including a 5-epoch linear warm-up and cosine annealing (1.5×10^{-3} to 1×10^{-5}). The CNN is pre-trained on 320×320 images for 100 epochs, then frozen and combined with the GNN and fusion block. During training, the model outputs instance sequences and images, with the total loss calculated as the sum of outputs for supervised learning. For IGHF's filter settings: stage 1 (CNN) has embedding [3,32], LLE [32,64,128,256,512], and decoder [512,256,128,64,32,1]; stage 2 (GNN) uses CVA [4,16,32,64,128,64,32,16,1]; the heterogeneous fusion stage has linear fuse [2,1].

To evaluate the performance, the proposed IGHF is compared with two excellent CNN-based methods including newly-proposed SOTA MAUNet [13], and widely-used IREDGe [12]. IGHF, MAUNet, and IREDGe have training times of 12, 8, and 5.5 hours, respectively. Note that, the famous method PowerNet [11] is excluded due to its unaffordable forward time. Moreover, accuracy and graph metrics (Pearson correlation coefficient (PCC), coefficient of determination R^2, mean absolute error (MAE) (mV), and average max MAE (AMMAE) (mV)) are used for assessment [34]. MAE evaluates prediction accuracy, AMMAE assesses errors in hotspot regions, and PCC and R^2 measure precision.

From Tab. I, IGHF shows significant performance gains in all four metrics. Compared to MAUNet, our method still outperforms with a notable margin in metrics (MAE: **0.378 mV** vs. 0.501 mV (over **24.6%** $\downarrow$) PCC: **0.973** vs. 0.961). It also reduces MAE by 55.0% compared to IREDGe. Moreover, lower AMMAE indicates the better robustness of IGHF in worst-case situations. In addition, Fig. 4 shows prediction and error maps for two chips. Compared with the SOTA MAUNet and IREDGe methods, our prediction images have

TABLE I
PREDICTION PERFORMANCE COMPARISONS.

Model / RTL	IGHF (Ours)				IREDGe [12]				MAUNet [13]			
	PCC ↑	R^2 ↑	MAE ↓	AMMAE ↓	PCC ↑	R^2 ↑	MAE ↓	AMMAE ↓	PCC ↑	R^2 ↑	MAE ↓	AMMAE ↓
ZERO-RISCY	0.975	0.934	0.360	17.818	0.932	0.808	0.793	32.250	0.960	0.897	0.489	24.944
RISCY	0.974	0.940	0.357	22.123	0.930	0.790	0.790	41.513	0.961	0.899	0.473	28.012
RISCY-FPU	0.977	0.941	0.358	21.839	0.943	0.750	0.814	38.061	0.966	0.897	0.486	27.386
VORTEX	0.908	0.791	1.310	267.136	0.882	0.771	2.054	392.454	0.899	0.780	1.430	269.440
NVDLA	0.863	0.742	1.311	196.230	0.704	0.567	5.750	192.005	0.849	0.731	1.374	199.104
Total	**0.973**	**0.910**	**0.378**	**24.790**	**0.933**	**0.795**	**0.840**	**41.296**	**0.961**	**0.874**	**0.501**	**30.988**

TABLE II
TRANSFER PERFORMANCE COMPARISONS.

RTL	Model	PCC	R^2	MAE	AMMAE
VORTEX Avg. Cells: 109944	IGHF	0.891	0.783	2.223	251.098
	IGHF w/ TL	0.938	0.859	1.053	183.207
	MAUNet	0.888	0.721	1.340	288.057
	MAUNet w/ TL	0.919	0.756	1.113	254.213
NVDLA Avg. Cells: 279228	IGHF	0.803	0.718	2.223	225.207
	IGHF w/ TL	0.908	0.811	0.919	182.632
	MAUNet	0.737	0.663	3.837	244.402
	MAUNet w/ TL	0.880	0.753	1.261	182.791

TABLE III
ABLATION EXPERIMENTS FOR IGHF.

GNN	CNN		Metrics			
CVA	LLE	HACG	PCC ↑	MAE ↓	R^2 ↑	AMMAE ↓
✓	✓	✓	**0.973**	**0.378**	**0.910**	**24.790**
	✓	✓	0.969	0.389	0.905	27.717
✓			0.832	1.171	0.781	41.790
	✓		0.964	0.456	0.897	29.070
		✓	0.958	0.458	0.895	29.037

higher structural fidelity. There are significant error reductions in multi-level (high and low IR drop regions, especially in the top 10% and 20% drop hotspot areas) and multi-scale (global and local) IR drop regions.

B. Transfer Experiments

To adapt to different IR drop characteristics of circuits, transfer learning (TL) is used to enhance the IGHF model's performance and generalization for limited unseen real-circuit test cases. To adapt to different PDN designs, we applied the same fine-tuning scheme, scale and shift feature adapter (SSF-ADA)[35], as MAUNet. The pre-trained set includes ZERO-RISCY, FPU, and RISCY-FPU RTL (avg. cell count: 46,290). The fine-tuning set has some NVDLA and VORTEX design cases (avg. cell counts: 109,944 and 279,228), while the remainings are kept hidden for evaluation. During training, the encoder was frozen, while the SSF-ADA module and decoder were activated. The IGHF model's performance with MAUNet was compared before and after TL in Tab. II.

The IGHF model consistently outperforms the MAUNet model in Tab. II, both before and after TL. Specifically for the NVDLA RTL, the MAE of IGHF with TL is reduced by 58.70% compared to IGHF without TL (from 2.223 mV to 0.919 mV). Furthermore, the MAE of IGHF with TL decreases by 27.11% compared to that of MAUNet with TL.

C. Ablation Experiments

This section performs two ablation experiments. First, we investigate the effectiveness of heterogeneous feature fusion by comparing line 1 (Proposed IGHF), line 2 (IGHF without GNN), and line 3 (IGHF without CNN) in Tab. III. The results show that combining the improved CNN branch with the GNN branch significantly enhances IR drop prediction performance, as the GNN branch supplements the PDN topological awareness that the CNN branch lacks.

For the CNN branch, the effectiveness of the proposed LLE and HACG modules can be assessed by comparing line 2 with lines 4 and 5, respectively. It is clear that both the LLE, which extracts multi-scale global-to-local spatial power features, and the HACG module, which facilitates different-scale feature adaptive interaction and compensation, are beneficial for significantly enhancing prediction accuracy.

V. CONCLUSIONS

In this paper, a novel image-graph heterogeneous fusion framework is proposed, which combines a CNN branch for the multi-scale global-to-local spatial feature extraction, and a GNN branch for aggregating discriminative cell instance topological features. In the CNN branch, LLE and HACG modules complete the global and local PDN feature perception and the fusion of features at each scale, respectively. In the GNN branch, the attention and propagation layers in CVA adaptively aggregate PDN topological features of heterogeneous neighbors of different orders. Extensive experiments on a large public dataset demonstrate that our IGHF outperforms the newly-proposed SOTA MAUNet and popular IREDGe methods for IR drop prediction by a large margin (over 24.6% and 55.0%). Transfer experiments denote that the error metrics of IGHF with TL show a decrease of 27.1% compared to those of MAUNet with TL. We hope that the proposed method can advance the development of CNN-GNN heterogeneous architecture design in the IR drop analysis.

ACKNOWLEDGMENT

Corresponding author is Zhou Jin. This work is partially supported by NSFC (Grant No. 62374031), Natural Science Foundation of Jiangsu Province (BK20240173), NSFC (Grant No. 62234010, 92473107, 62204265, U23A20301).

REFERENCES

[1] X. Zhan, P. Li, and E. Sánchez-Sinencio, "Distributed on-chip regulation: Theoretical stability foundation, over-design reduction and performance optimization," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2016.

[2] B. Liu, H. Li, Y. Chen, X. Li, T. Huang, Q. Wu, and M. Barnell, "Reduction and ir-drop compensations techniques for reliable neuromorphic computing systems," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 63–70, 2014.

[3] K. F. Yong, C. T. Lim, and W. K. Teng, "System Level IR Drop Impact on Chip Power Performance Signoff for RISC-V System on Chip," in *2022 17th International Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT)*, pp. 1-4, 2022.

[4] D. Niu, Y. Dong, Z. Jin, C. Zhang, Q. Li, and C. Sun, "OSSP-PTA: An Online Stochastic Stepping Policy for PTA on Reinforcement Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4310-4323, 2023.

[5] Z. Liu, and W. Yu, "Accuracy-Preserving Reduction of Sparsified Reduced Power Grids with A Multilevel Node Aggregation Scheme," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1-9, 2023.

[6] P. Li, M. Tang, Z. X. Huang, L. J. Jiang and H. Bağcı, "DC IR-Drop Analysis of Multilayered Power Distribution Network by Discontinuous Galerkin Method With Thermal Effects Incorporated," *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 10, no. 6, pp. 1035-1042, 2020.

[7] Yuan Liang, Wenjian Yu, and Haifeng Qian. A hybrid random walk algorithm for 3-d thermal analysis of integrated circuits. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 849–854, 2014.

[8] Z. Zeng, T. Xu, Z. Feng, and P. Li, "Fast static analysis of power grids: Algorithms and implementations," in *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 488-493, 2011.

[9] Y. Zhao, Z. Chai, X. Jiang, Y. Lin, R. Wang, and R. Huang, "PDNNet: PDN-Aware GNN-CNN Heterogeneous Network for Dynamic IR Drop Prediction," arXiv preprint arXiv:2403.18569, 2024.

[10] Chi-Hsien Pao, An-Yu Su, and Yu-Min Lee, "Xgbir: An xgboost-based ir drop predictor for power delivery network," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1307–1310, 2020.

[11] Z. Xie, H. Ren, B. Khailany, Y. Sheng, S. Santosh, J. Hu, and Y. Chen, "PowerNet: Transferable Dynamic IR Drop Estimation via Maximum Convolutional Neural Network," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 13–18, 2020.

[12] V. A. Chhabria, V. Ahuja, A. Prabhu, N. Patil, P. Jain, and S. S. Sapatnekar, "Thermal and IR Drop Analysis Using Convolutional Encoder-Decoder Networks," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 690-696, 2021.

[13] M. Wang, Y. Cheng, Y. Lin, K. Peng, S. Yang, Z. Jin, and W. Xing, "MAUnet: Multiscale Attention U-Net for Effective IR Drop Prediction," in *2024 61st ACM/IEEE Design Automation Conference (DAC)*, 2024.

[14] A. H. Ajami, K. Banerjee, A. Mehrotra, and M. Pedram, "Analysis of IR-drop scaling with implications for deep submicron P/G network designs," in *Fourth international symposium on quality electronic design (isqed'03)*, pp. 35-40. 2003.

[15] A. Dubey, "P/G pad placement optimization: problem formulation for best IR drop," in *Sixth international symposium on quality electronic design (isqed'05)*, pp. 340-345, 2005.

[16] N. Tomen, and J. C. van Gemert, "Spectral leakage and rethinking the kernel size in cnns," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 5138-5147, 2021.

[17] Y. Xu, and M. Vaziri-Pashkam, "Limits to visual representational correspondence between convolutional neural networks and the human brain," *Nature communications*, vol. 12, no. 1, pp. 2065, 2021.

[18] H. Sun, C. Jiang, X. Ye, D. Feng, B. Tan, Y. Ma, and K. Liu, "Interpretable CNN-Based Lithographic Hotspot Detection Through Error Marker Learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[19] R. S. Shelar, and M. Patyra, "Impact of Local Interconnects on Timing and Power in a High Performance Microprocessor," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 10, pp. 1623-1627, 2013.

[20] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri and O. Sinanoglu, "GNN-RE: Graph Neural Networks for Reverse Engineering of Gate-Level Netlists," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2435-2448, 2022.

[21] H. Ren, S. Nath, Y. Zhang, H. Chen, and M. Liu, "Why are Graph Neural Networks Effective for EDA Problems?," in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1-8, 2022.

[22] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," AI open, vol. 1, pp. 57-81, 2020.

[23] S. Köse, and E. G. Friedman, "Fast algorithms for IR voltage drop analysis exploiting locality," in *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 996-1001, 2011.

[24] L. Li, Z. Liu, K. Liu, S. Shen, and W. Yu, "Parallel Incomplete LU Factorization Based Iterative Solver for Fixed-Structure Linear Equations in Circuit Simulation," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 339-345, 2023.

[25] V. A. Chhabria, Y. Zhang, H. Ren, B. Keller, B. Khailany, and S. S. Sapatnekar, "MAVIREC: ML-Aided Vectored IR-Drop Estimation and Classification," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1825-1828, 2021.

[26] S. Chang, N. Dasgupta, and L. Carin, "A Bayesian approach to unsupervised feature selection and density estimation using expectation propagation," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pp. 1043-1050, 2005.

[27] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012-10022, 2021.

[28] S. Woo, S. Debnath, R. Hu, X. Chen, Z. Liu, I. S. Kweon, and S. Xie, "Convnext v2: Co-designing and scaling convnets with masked autoencoders," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16133-16142, 2023.

[29] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510-4520, 2018.

[30] J. Ruan, M. Xie, J. Gao, T. Liu, and Y. Fu, "Ege-UNet: an efficient group enhanced UNet for skin lesion segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 481-490, 2023.

[31] R. Cong, H. Yang, Q. Jiang, W. Gao, H. Li, C. Wang, Y. Zhao, and S. Kwong, "BCS-Net: Boundary, context, and semantic for automatic COVID-19 lung infection segmentation from CT images," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1-11, 2022.

[32] Y. Dong, Q. Liu, B. Du and L. Zhang, "Weighted Feature Fusion of Convolutional Neural Network and Graph Attention Network for Hyperspectral Image Classification," *IEEE Transactions on Image Processing*, vol. 31, pp. 1559-1572, 2022.

[33] S. Abu-El-Haija, B. Perozzi, A. Kapoor, N. Alipourfard, K. Lerman, H. Harutyunyan, G. Ver Steeg, and A. Galstyan, "Mixhop: Higher-order graph convolutional architectures via sparsified neighborhood mixing," in *international conference on machine learning*, 2019, pp. 21-29.

[34] Z. Chai, Y. Zhao, W. Liu, Y. Lin, R. Wang, and R. Huang, "CircuitNet: An Open-Source Dataset for Machine Learning in VLSI CAD Applications With Improved Domain-Specific Evaluation Metric and Learning Strategies," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 5034-5047, 2023.

[35] D. Lian, D. Zhou, J. Feng, and X. Wang, "Scaling & shifting your features: A new baseline for efficient model tuning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 109-123, 2022.

Device-Algorithm Co-Design of Ferroelectric Compute-in-Memory In-Situ Annealer for Combinatorial Optimization Problems

Yu Qian[1], Xianmin Huang[1], Ranran Wang[1], Zeyu Yang[1], Min Zhou[1], Thomas Kämpfe[2,3], Cheng Zhuo[1,4*] and Xunzhao Yin[1,4*]
[1]Zhejiang University, Hangzhou, China; [2]Fraunhofer IPMS, Dresden, Germany; [3]TU Braunschweig, Braunschweig, Germany;
[4]Key Laboratory of CS&AUS of Zhejiang Province, Hangzhou, China; *Corresponding author email: {czhuo, xzyin1}@zju.edu.cn

ABSTRACT

Combinatorial optimization problems (COPs) are crucial in many applications but are computationally demanding. Traditional Ising annealers address COPs by directly converting them into Ising models (known as direct-E transformation) and solving them through iterative annealing. However, these approaches require vector-matrix-vector (VMV) multiplications with a complexity of $O(n^2)$ for Ising energy computation and complex exponential annealing factor calculations during annealing process, thus significantly increasing hardware costs. In this work, we propose a ferroelectric compute-in-memory (CiM) in-situ annealer to overcome aforementioned challenges. The proposed device-algorithm co-design framework consists of (i) a novel transformation method (first to our known) that converts COPs into an innovative incremental-E form, which reduces the complexity of VMV multiplication from $O(n^2)$ to $O(n)$, and approximates exponential annealing factor with a much simplified fractional form; (ii) a double gate ferroelectric FET (DG FeFET)-based CiM crossbar that efficiently computes the in-situ incremental-E form by leveraging the unique structure of DG FeFETs; (iii) a CiM annealer that approaches the solutions of COPs via iterative incremental-E computations within a tunable back gate-based in-situ annealing flow. Evaluation results show that our proposed CiM annealer significantly reduces hardware overhead, reducing energy consumption by 1503/1716× and time cost by 8.08/8.15× in solving 3000-node Max-Cut problems compared to two state-of-the-art annealers. It also exhibits high solving efficiency, achieving a remarkable average success rate of 98%, whereas other annealers show only 50% given the same iteration counts.

1 INTRODUCTION

Combinatorial optimization problems (COPs) have widespread applications across various fields, including logistics, resource allocation, communication network design, and transportation systems [1–4], and many of these problems are categorized as non-deterministic polynomial-time hard (NP-hard), reflecting their computational complexity. A wide range of COPs, including Max-Cut, graph coloring, and knapsack problems, etc., can be effectively reformulated into Ising models or Quadratic Unconstrained Binary Optimization (QUBO) forms, which can be addressed by Ising machines and QUBO solvers [5–8]. For example, digital ASIC annealers [9, 10] implement different annealing algorithms within digital circuits, and dynamical system Ising machines [11, 12] utilize the intrinsic system dynamics and their tendency to settle at ground state to solve COPs. These solvers address COPs more quickly and efficiently than von Neumann hardware, yet still suffer from significant hardware overhead and limited robustness.

Recently, numerous compute-in-memory (CiM) frameworks based on non-volatile memories (NVMs) such as ferroelectric field effect transistor (FeFET) and resistive random access memory (ReRAM), have been proposed to further improve the hardware performance in solving COPs, known as CiM annealers [7, 13–16]. Compared to the aforementioned solvers, CiM annealers utilize compact and highly parallel NVM-based crossbar arrays to offer lower hardware costs and higher robustness. As Fig. 1(a) shows, after COPs are converted to a direct Ising model, CiM annealers compute the Ising energy within each simulated annealing iteration, progressively approaching optimal solutions as the temperature T decreases. As illustrated in Fig. 1(b), at a given T, the annealer computes the new Ising energy through a CiM accelerated vector-matrix-vector (VMV) multiplication, and calculates the incremental energy difference ΔE in the digital circuits. Depending on ΔE, an exponential annealing factor $e^{-\frac{\Delta E}{T}}$ is computed to decide whether to accept the new solution. Within the annealing process, CiM acceleration and digital computation are performed iteratively. As can be seen, the CiM part involves a complex VMV multiplication with a computational complexity of $O(n^2)$, leading to large energy consumption and increased time cost [17]. Additionally, the digital computation part requires calculating $e^{-\frac{\Delta E}{T}}$, which demands significant hardware resources, further reducing the computational efficiency [18].

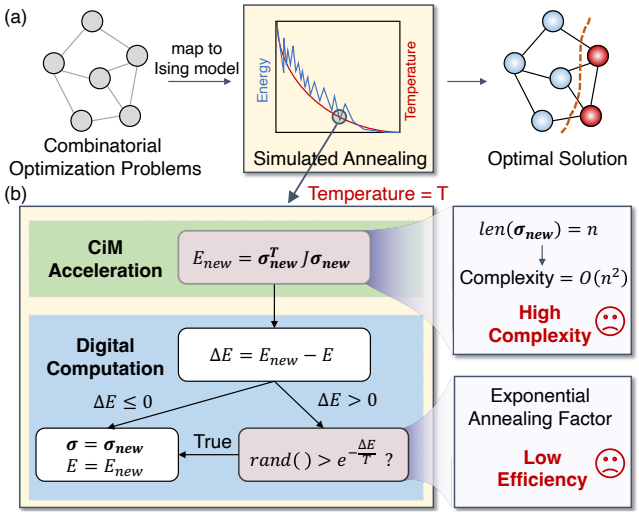


Figure 1: CiM annealer solves COPs. (a) General flow based on simulated annealing algorithm; (b) Current solvers have high complexity in CiM acceleration part, and low efficiency in digital computation part.

To address aforementioned challenges, we introduce a device-algorithm co-design approach of ferroelectric CiM in-situ annealer that not only significantly reduces the complexity of direct Ising energy and exponential factor calculations, but also enhances the efficiency by leveraging the unique characteristics of a double gate ferroelectric FET (DG FeFET)-based CiM crossbar for in-situ annealing. The key contributions of this work are summarized as follows:

- We propose an incremental-E transformation method that converts a COP into an efficient incremental-E (E_{inc}) form. The E_{inc}

form eliminates redundant computations within the VMV multiplications, and approximates complex exponential annealing factor with simpler fractional annealing factor. As a result, the complexity of E_{inc} computation is reduced from $O(n^2)$ to $O(n)$, significantly alleviating the associated hardware overhead.

- We propose to realize a single DG FeFET-based four-input multiplication, as the device naturally supports the operator by applying three inputs to the double gates and drain, and storing an input as threshold voltage (V_{TH}) state, respectively. Based on the unique characteristic of devices, we propose a DG FeFET-based CiM crossbar array that computes the entire E_{inc} form, implementing the in-situ product of the VMV multiplication result and fractional annealing factor within a single array.
- Based on the proposed transformation method and the unique CiM design, we propose a CiM in-situ annealer, where the DG FeFET's back gate (BG) is utilized to mimic the temperature decay within the annealing flow due to its V_{TH} tunability. A tunable BG-based in-situ annealing flow is employed, performing all the analog computations within the CiM array, and leaving only the solution update in digital part.
- The proposed CiM annealer demonstrates significant reductions in both time and energy consumption while achieving high accuracy in solving COPs. In solving 30 Max-Cut problems with varying node numbers from 800 to 3000, it offers 410× to 1706× energy reductions and 7.98× to 8.15× time cost reductions compared to other two state-of-the-art annealers. It also achieves a 98% success rate in average, while other annealers reach only 50%, given the same iteration counts.

2 BACKGROUND

In this section, we first introduce the Ising models and related works, then review CiM and DG FeFET basics.

2.1 Ising models and existing solvers

Ising models provide a powerful framework for modeling and solving COPs [8], where problem variables are represented as spins with binary states, and constraints are captured through spin couplings. The objective function of the problem is formulated as the Hamiltonian energy function of the Ising model. Solving the problem involves finding the spin configuration that minimizes the corresponding Ising Hamiltonian H_{P}:

$$\min H_{\mathrm{P}} = \sum_{i,j=1}^{n} J_{ij}\sigma_i\sigma_j + \sum_{i=1}^{n} h_i\sigma_i \tag{1}$$

where n denotes the number of spins and $\sigma_i \in \pm 1$ represents the state of spin i. J_{ij} and h_i stand for the coupling between spin i and j and the self-coupling of spin i, respectively. Eq. (1) can be expressed in an alternative general form by setting $J_{ii} = h_i$:

$$\min E = \boldsymbol{\sigma}^T J \boldsymbol{\sigma} \tag{2}$$

where $\boldsymbol{\sigma}$ presents the spin vector. The Ising model is equivalent to the QUBO form [19] through a simple variable transformation $\sigma_i = 1 - 2x_i, x_i \in \{0, 1\}$. Thus, the Ising machines and QUBO solvers can be regarded as equivalent COP solving designs.

Numerous Ising machines have been developed to address COPs. Digital ASIC annealers, which implement various annealing algorithms to solve Ising models [9, 10], often suffer from high energy consumption and time costs due to the exponential growth of explicit computations in each annealing iteration. Dynamic system Ising machines, on the other hand, exploit the natural dynamics of physical systems to explore optimal solutions [20, 21], but they

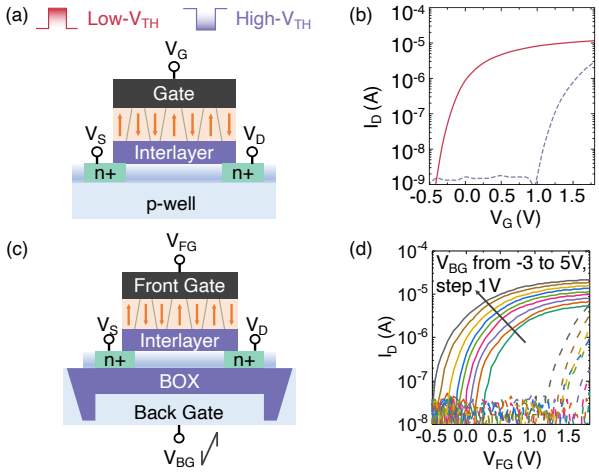


Figure 2: FeFET and DG FeFET. (a) A FeFET device with three terminals; (b) $I_D - V_G$ curves of a FeFET with programmed low/high V_{TH}. (c) A DG FeFET device with four terminals; (d) $I_D - V_G$ curves of the DG FeFET, where V_{BG} can flexibly adjust its V_{TH} after programming.

are highly sensitive to spin coupling implementations. Even slight deviations in coupling strength can hinder convergence [12, 20].

To further enhance the robustness and hardware efficiency of solving COPs, a novel framework known as the CiM annealer has been proposed recently [7, 14, 15]. Fig. 1(a) depicts the general flow of the CiM annealer. The COPs, e.g., Max-Cut problems, are formulated as Ising models. The Ising energy E is progressively minimized as temperature decreases, following the simulated annealing (SA) algorithm. At each temperature, the value of E_{new} based on updated spin states $\boldsymbol{\sigma_{new}}$ is computed using the CiM architecture, followed by peripheral digital computations. A FeFET-based CiM annealer in [7] provides a lossless compression method to reduce the cost of CiM architecture and enhances the SA algorithm into Multi-Epoch SA (MESA) for improved performance. The HyCiM framework in [15] addresses general COPs with inequality constraints by employing an inequality filter CiM structure. A C-Nash CiM framework is proposed to solve COPs with the MAX function embedded in the objective function [14]. While these CiM annealers leverage the CiM to reduce energy and time costs and enhance SA algorithm efficiency, they still face challenges on hardware overhead:

As shown in Fig. 1(b), the core VMV multiplication operation within each annealing iteration involves n^2 product terms given n spins, exhibiting a high complexity of $O(n^2)$. Additionally, as shown in Fig. 1(b), a complex exponential annealing factor calculation is required when $\Delta E > 0$, further reducing the efficiency of the CiM annealer. This has inspired us to propose an incremental-E transformation that reduces the complexity of VMV multiplication from $O(n^2)$ to $O(n)$, and eliminates the costly exponential function. Such incremental-E form is then implemented by a CiM array.

2.2 CiM Preliminaries and DG FeFET Basics

Prior CiM designs employ NVMs, such as ReRAM, magnetic tunneling junction (MTJ) and FeFET to build crossbars that store weight matrices, and perform parallel vector-matrix multiplication or VMV multiplication operations [7, 22–26]. Among these devices, FeFET offers advantages in energy and area efficiency due to the CMOS compatibility, voltage-driven mechanism, high ON/OFF ratio and

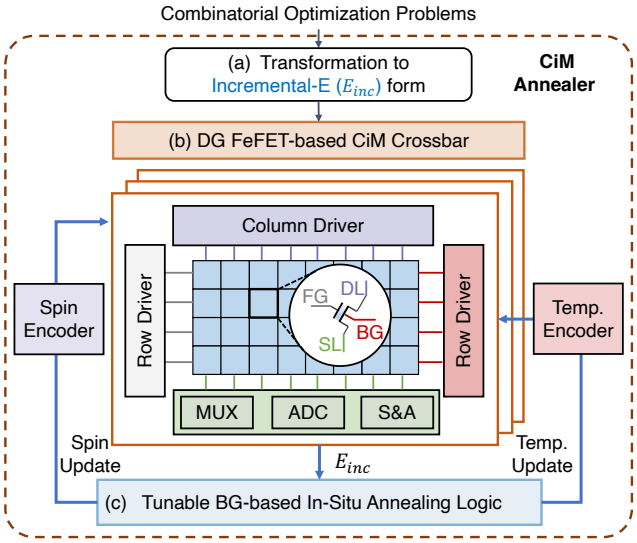


Figure 3: Overview of the proposed CiM annealer. (a) Incremental-E (E_{inc}) transformation method; (b) DG FeFET-based crossbar for E_{inc} computation; (c) Tunable BG-based in-situ annealing for approaching optimal solutions.

three-terminal structure [27–33]. When the gate is applied with different gate pulses as shown in Fig. 2(a), a FeFET can store high or low V_{TH} states, which are experimentally measured from the FeFET [7] as shown in Fig. 2(b). Prior works exploit the intrinsic three-input multiplication operator of FeFETs for VMV multiplication acceleration, thus enabling efficient CiM-based COP solvers [7, 14, 15]. That said, the potentials of ferroelectric devices are yet to be explored.

Recently, a novel technology known as DG FeFET [34] has been developed, which incorporates a non-ferroelectric (non-FE) dielectric in the BG. A notable example of such a device structure is fully-depleted silicon-on-insulator (FDSOI) FeFET, as shown in Fig. 2(c), where the buried oxide (BOX) layer serves as the non-ferroelectric dielectric. Unlike the fixed $I_D - V_G$ curves of a conventional FeFET, the $I_D - V_G$ curves for the low-V_{TH}/high-V_{TH} state of a DG FeFET can be dynamically adjusted by applying different voltage on BG, as illustrated in Fig. 2(d). This V_{TH} tuning does not change the ferroelectric states, but is rather due to the electric field exerted through the BG. This unique BG-based V_{TH} tuning is used in this work to create a cell that efficiently operates with three input terminals (i.e., double gates and drain), enabling in-situ annealing computations.

3 PROPOSED CIM ANNEALER

Here we introduce our proposed device-algorithm co-design approach, including the incremental-E transformation, the DG FeFET-based CiM array and tunable BG-based in-situ annealing flow.

3.1 Overview

Fig. 3 illustrates the overview of the proposed CiM annealer. The COP is firstly transformed into an incremental-E (E_{inc}) form (Sec. 3.2), which is then mapped onto the proposed DG FeFET-based CiM crossbar for computation (Sec. 3.3). Finally, the tunable BG-based in-situ annealing flow is employed, approaching optimal solutions (Sec. 3.4).

During each annealing iteration, the tunable BG-based annealing flow updates the spin states based on the previous E_{inc} value, transmitting solely the updated spins to the spin encoder. The updated temperature is relayed to the temperature encoder. The DG FeFET-based CiM array is then activated to compute the new E_{inc}, which is fed back to the annealing logic to initiate next iteration.

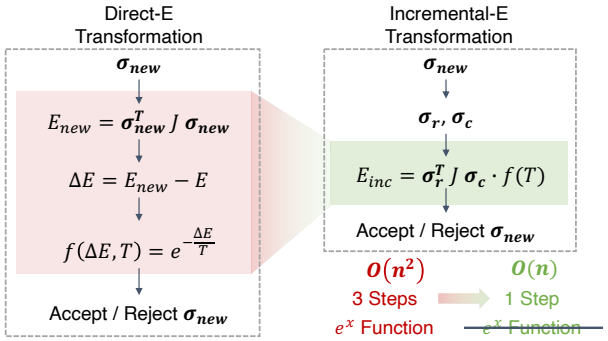


Figure 4: Compared to direct-E, the incremental-E transformation reduces complexity from $O(n^2)$ to $O(n)$, condenses three steps into a single operation, and eliminates e^x function.

3.2 Incremental-E Transformation

Fig. 4 shows the proposed incremental-E transformation method, which replaces the costly VMV multiplication of E_{new} and e^x function in the direct-E transformation with the product of an incremental-based VMV multiplication and a fractional factor, thus reducing the complexity from $O(n^2)$ to $O(n)$. Such transformation merges the three critical computations into a single step, and eliminates the need for e^x calculation.

As shown in Fig. 1(b), current annealers calculate ΔE within an iteration as follows:

$$E_{new} = \boldsymbol{\sigma_{new}^T} J \boldsymbol{\sigma_{new}} \tag{3}$$

$$\Delta E = E_{new} - E \tag{4}$$

where E and E_{new} denote the current and new Ising energy, and $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma_{new}}$ denote the current and new spin vectors, respectively. Fig. 5 shows the process of complexity reduction in the proposed incremental-E transformation. Within an iteration, the current spin vector $\boldsymbol{\sigma}$ (Fig. 5(a)) is flipped to generate $\boldsymbol{\sigma_{new}}$ (Fig. 5(b)). The corresponding E_{new} can be decomposed as:

$$E_{new} = \sum_{i,j\in\overline{F}} \sigma_i\sigma_j J_{ij} + \sum_{i,j\in F} \sigma_i\sigma_j J_{ij} + \sum_{i\in F, j\in\overline{F}} \sigma_i\sigma_j J_{ij} + \sum_{i\in\overline{F}, j\in F} \sigma_i\sigma_j J_{ij} \tag{5}$$

where each spin $\sigma_i \in \{-1, 1\}$, F denotes the index set of flipped spins, and $\overline{F}$ denotes the rest set. The computation of E_{new} entails n^2 product terms, exhibiting a complexity of $O(n^2)$. These terms can be categorized into three groups, i.e., $\sigma_i\sigma_j J_{ij}$ with 0/1/2 flipped spins, as shown in Fig. 5(b), corresponding to the boxes in blue/orange/green within the spin coupling matrix. If spin i and j are both flipped (switch to the other binary state), the term $\sigma_i\sigma_j J_{ij}$ remains unchanged. Therefore, the green terms can be recolored to blue, as depicted in Fig. 5(c). E_{new} can then be expressed as:

$$E_{new} = \sum_{(i,j\in F)\cup(i,j\in\overline{F})} \sigma_i\sigma_j J_{ij} + \sum_{(i\in\overline{F},j\in F)\cup(i\in F,j\in\overline{F})} \sigma_i\sigma_j J_{ij} \tag{6}$$

where the latter term in orange corresponds to the distinct terms between E and E_{new}. These orange terms can be merged along the diagonal of matrix due to the symmetry. After the merging, we define a logical vector $\boldsymbol{\sigma_f}$, where $\sigma_{f,i} = 1$ indicates that spin i is flipped. We also define two vectors, $\boldsymbol{\sigma_c}$ and $\boldsymbol{\sigma_r}$, based on $\boldsymbol{\sigma_{new}}$:

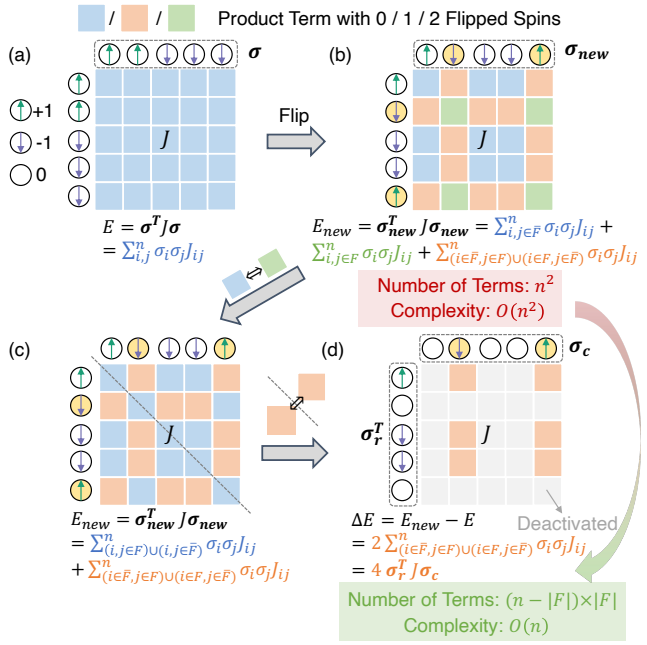


Figure 5: Complexity reduction of VMV multiplication in the incremental-E transformation. After (a) σ is flipped to (b) σ_{new}, E_{new} has a complexity of $O(n^2)$, composing of three groups of product terms in different colors; (c) The distinct terms between E and E_{new} in orange are merged by diagonal symmetry, resulting in (d) ΔE with $O(n)$ complexity.

$$\boldsymbol{\sigma_c} = \boldsymbol{\sigma_{new}} \circ \boldsymbol{\sigma_f} \tag{7}$$

$$\boldsymbol{\sigma_r} = \boldsymbol{\sigma_{new}} \circ (1 - \boldsymbol{\sigma_f}) \tag{8}$$

where $\circ$ denotes element-wise multiplication. The vector $\boldsymbol{\sigma_c}$ maintains the values of the flipped elements in $\boldsymbol{\sigma_{new}}$, setting all others to zero, while $\boldsymbol{\sigma_r}$ preserves the unflipped spin values, zeroing out the flipped elements. As a result, ΔE can be expressed as an incremental-based VMV multiplication:

$$\Delta E = 4\boldsymbol{\sigma_r^T} J \boldsymbol{\sigma_c} \tag{9}$$

where the number of product terms is only $(n-|F|)\times|F|$, exhibiting a reduced complexity of $O(n)$ because $|F|$ is set to be constant, as depicted in Fig. 5(d).

The exponential annealing factor in Fig. 1(b) is approximated as

$$e^{-\frac{\Delta E}{T}} \approx 1 - \frac{\Delta E}{T} \propto -\frac{\Delta E}{T} \tag{10}$$

Therefore, after simplification, the incremental-E form is as follows:

$$E_{inc} = \boldsymbol{\sigma_r^T} J \boldsymbol{\sigma_c} \cdot f(T) \tag{11}$$

where $f(T) = \frac{a}{bT+c} + d$, with parameters a, b, c, and d properly chosen. Note that Eq. (11) must satisfy (i) $f(T) > 0$, as $e^{-\frac{\Delta E}{T}} > 0$, and (ii) $f(T) \propto T$. The resulted E_{inc} formulation can be efficiently implemented by our proposed DG FeFET-based CiM crossbar.

3.3 DG FeFET-based CiM Crossbar

As shown in Fig. 6(a), a single DG FeFET has four terminals: the data line (DL), front gate (FG), back gate (BG) and source line (SL). A low/high V_{TH} is stored in the DG FeFET, denoted as G = '1'/'0'. The inputs at FG, DL and BG are denoted as x, y and z, where x and y are binary inputs ($x/y \in \{0,1\}$) and z is an analog input. SL is the output terminal. Fig. 6(b) shows the $I_{SL} - V_{BG}$ characteristics given fixed DL and FG inputs. It can be seen that DG FeFET naturally performs the operation $I_{SL} = x \cdot G \cdot y \cdot z$, where I_{SL} remains 0 when

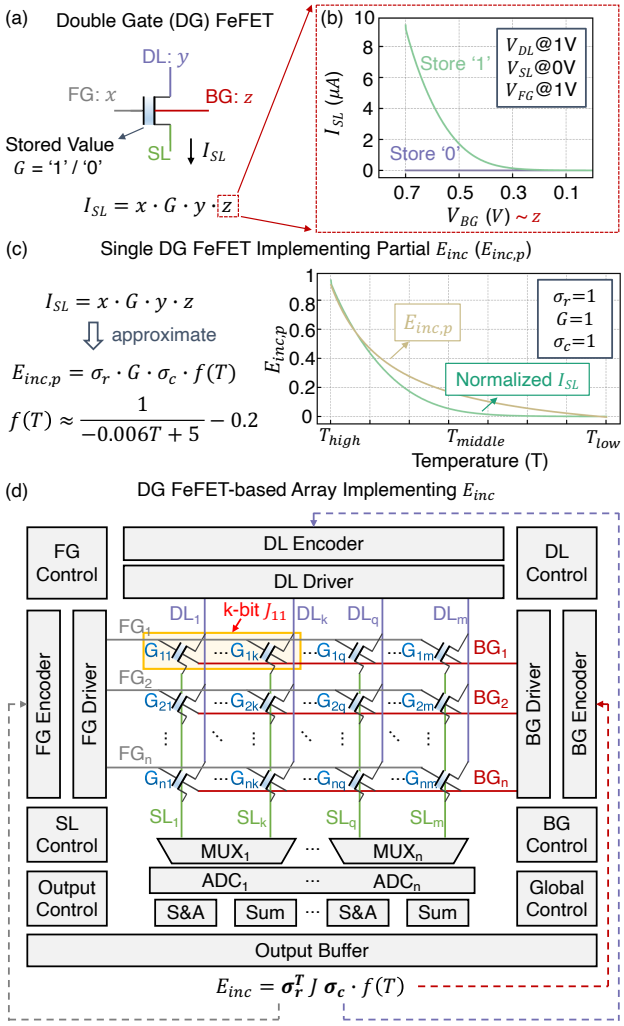


Figure 6: DG FeFET-based CiM crossbar array. (a) A DG FeFET naturally performs $I_{SL} = x \cdot G \cdot y \cdot z$; (b) V_{BG} can adjust the V_{TH}, thus correspondingly tune I_{SL} with DG FeFET storing '1'; (c) $I_{SL} - V_{BG}$ of a DG FeFET can approximate the partial incremental-E form $E_{inc,p}$, where the fractional annealing factor $f(T)$ is mapped as I_{SL}; (d) The DG FeFET-based crossbar array can implement incremental-E form E_{inc}.

G = '0', and follows V_{BG} when storing '1'. This intrinsic operation of the DG FeFET resembles the partial incremental-E formulation, i.e., $E_{inc,p} = \sigma_r \cdot G \cdot \sigma_c \cdot f(T)$. As illustrated in Fig. 6(c), the normalized output current I_{SL} (with '1' stored) can approximate $E_{inc,p}$ assuming $f(T) \approx \frac{1}{-0.006T+5} - 0.2$, where G is stored as binary V_{TH} state, σ_r, σ_c are applied to FG, DL, respectively, and T corresponds to V_{BG}. I_{SL} is sensed as the value of $E_{inc,p}$.

Fig. 6(d) shows the schematic of the proposed DG FeFET-based crossbar array. Each row and column share the same FG/BG and DL/SL, respectively. The peripheral circuits include controllers, encoders and drivers for FG, DL and BG inputs, and multiplexers (MUXs), ADCs, shift&adders (S&As), summations (Sums) and an output buffer for SL outputs. Every k columns share the same MUX, ADC, S&A and Sum. Each element in the matrix J, e.g., J_{11}, is mapped onto a $1 \times k$ subarray, with each cell storing 1 bit under

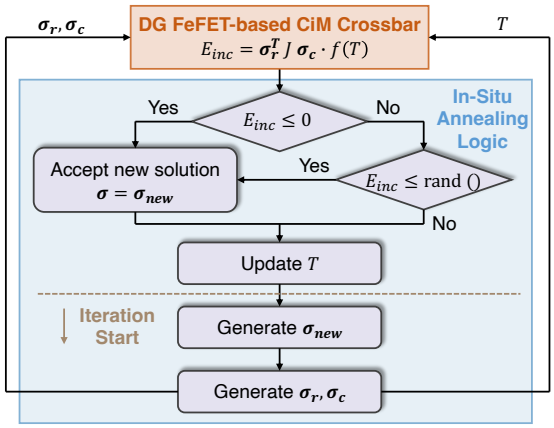


Figure 7: Working flow of the proposed CiM annealer.

k-bit quantization. In this way, an $n \times n$ matrix J is mapped onto an $n \times m$ crossbar ($m = n \times k$).

During each E_{inc} computation, the input vectors σ_r^T and σ_c are simultaneously directed to FG and DL encoders, respectively, and the input scalar $f(T)$ is sent to the BG encoder. FG and DL encoders convert σ_r^T and σ_c into binary voltages and drive the array through FG and DL drivers, respectively. The BG encoder looks at T value and generates the corresponding analog voltage via the BG driver to control all BGs. For the array output, k SLs form a group, and all groups perform the computations in parallel. k output currents from a group are sensed sequentially, multiplexed by the MUX and converted into k digital values via the ADC. The k digital values are then shifted, added and summed to generate the partial result. These partial results from all groups are aggregated to compute the final E_{inc} value, stored in the output buffer. Since the crossbar only supports non-negative inputs, the E_{inc} components associated with positive and negative inputs are separately calculated, respectively, and summed ultimately. To this end, the DG FeFET-based crossbar array successfully implements $E_{inc} = \sigma_r^T J \sigma_c \cdot f(T)$.

3.4 Tunable BG-based In-Situ Annealing Flow

Based on the incremental-E transformation and DG FeFET-based CiM crossbar array, we present the proposed CiM annealer that leverages the V_{TH} tunable BG of DG FeFETs for efficient in-situ annealing process. Fig. 7 illustrates the work flow of the annealer. In each annealing iteration, a new solution σ_{new} is randomly generated based on the current solution σ. The vectors σ_r and σ_c are derived from σ_{new}. The σ_r, σ_c and temperature T are then directed to the DG FeFET-based CiM crossbar for in-situ E_{inc} computation. The computed result of E_{inc} is sent to the annealing logic. If $E_{inc} \leq 0$, it indicates that the new solution corresponds to lower energy compared to current solution, thus the new solution is accepted, updating the current solution $\sigma = \sigma_{new}$. If $E_{inc} > 0$, the value is compared with a randomly generated number $r \in [0, 1]$. If $E_{inc} \leq r$, the new solution is still accepted; otherwise, σ remains unchanged. The current solution σ then serves as the input of next iteration and the temperature T is updated. The optimal solution is identified through iterative computations. Alg. 1 presents the pseudocode for the annealing algorithm.

The range of annealing temperature T is normalized to match the BG voltage range V_{BG} (0.7V to 0V) for the DG FeFET-based crossbar with a gradient of 0.01V, such that the temperature related annealing

Algorithm 1 In-Situ Annealing Algorithm

Input: J, t ▷ Ising model and number of flipped spins ($t = |F|$)
Output: σ ▷ Spin configuration that minimizes $E = \sigma^T J \sigma$

```
Initialize σ with random values from {−1, 1}
for each annealing iteration do
    Select t elements and get σ_f                ▷ Marks spins to flip
    σ_new = σ ∘ (1 − 2 · σ_f)                    ▷ Flip elements in σ
    σ_c = σ_new ∘ σ_f, σ_r = σ_new ∘ (1 − σ_f)
    E_inc = σ_r^T J σ_c · f(T)                   ▷ DG FeFET-based CiM acceleration
    if E_inc ≤ 0 then
        σ = σ_new                                ▷ Accept new solution
    else
        if E_inc ≤ random(0, 1) then
            σ = σ_new                            ▷ Accept new solution
        end if
    end if
    Update T based on annealing schedule
end for
```

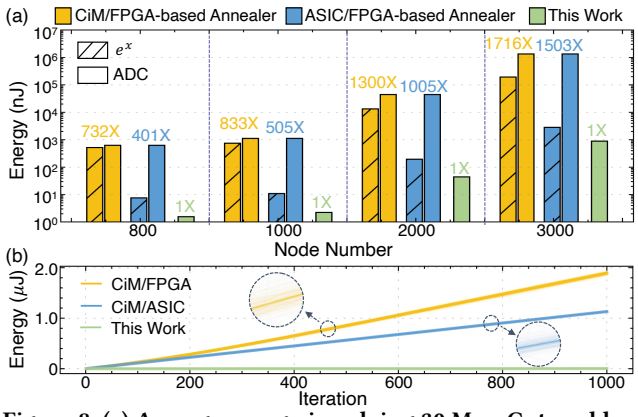


Figure 8: (a) Average energy in solving 30 Max-Cut problem instances and (b) energy trends in solving a 1000-node instance across three CiM annealers.

factor $f(T)$ can be approximately mapped to the crossbar currents by tuning V_{BG}, as illustrated in Fig. 6(c). T does not necessarily decrease with each iteration. For longer processes, T decreases only after a pre-set number of iterations. Once V_{BG} reaches 0V, it remains at zero, terminating the annealing.

4 EVALUATION

In this section, we first compare the hardware overhead of the proposed CiM annealer to other two CiM annealers [7, 18]. We then evaluate their efficiency in solving COPs. All simulations were conducted using SPECTRE. The 22nm BSIM IMG DG FeFET model [34] and Preisach FeFET model [35] are adopted for DG FeFET and FeFET, respectively. A 22nm commercial model is used for MOSFETs with TT process corner at a temperature of 27°C. Each annealer contains a single crossbar. 8-to-1 multiplexed ADCs are employed [36], which are scaled to 22nm. The hardware overhead for the exponential function implementation is derived from [18]. The wiring parasitics are extracted from DESTINY [37]. The Max-Cut problem is selected as the representative COP [38].

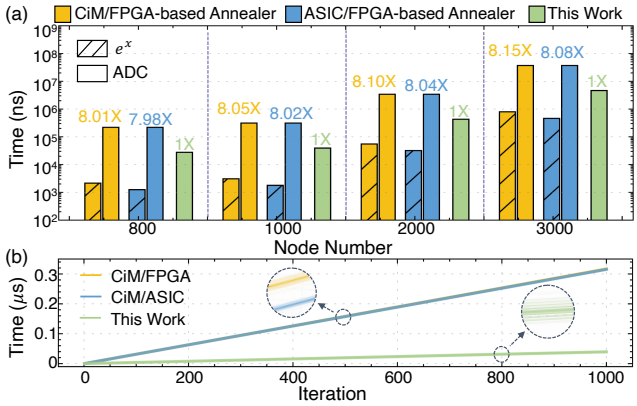


Figure 9: (a) Average time cost in solving 30 Max-Cut problems and (b) time cost trends in solving a 1000-node instance across three CiM annealers.

4.1 Hardware Overhead

For a fair comparison, the other two annealers are based on FeFET CiM accelerations [7] and FPGA/ASIC modules for exponential computation [18], denoted as CiM/FPGA and CiM/ASIC-based annealers, respectively. We selected 30 Max-Cut problem instances across four groups: 9 with 800 nodes, 9 with 1000 nodes, 9 with 2000 nodes, and 3 with 3000 nodes. For each instance, we conducted 100 runs. The iteration numbers per run are set according to problem size, with 700/1000/10,000/100,000 iterations for the 800-/1000-/2000-/3000-node instances, respectively.

Fig. 8(a) illustrates the average energy consumption of the three annealers. The major energy consumption are from the ADC and the exponential function implementation. It can be seen that, our proposed in-situ annealer achieves significant energy savings by reducing the ADC execution times and eliminating the energy associated with e^x calculation. This efficiency stems from the incremental-E transformation, which removes redundant VMV multiplications and simplifies the exponential calculation to a fractional function. Such transformation fully takes advantages of the DG FeFET-based CiM array, enabling efficient in-situ E_{inc} computations. Notably, the energy efficiency improvement increases as the problem size grows. Fig. 8(b) shows the energy consumptions versus iteration count of the three annealers when solving a 1000-node instance. The CiM/FPGA and CiM/ASIC-based annealers display a rapid linear energy increase with iteration counts. Conversely, our proposed annealer demonstrates a significantly slower rise in energy consumption. This is because larger-scale COPs require more annealing iterations, enlarging the energy savings associated with reduced ADC executions and simplified fractional factor calculations.

Fig. 9(a) presents the average time for problem solving. The proposed annealer shows substantial time savings across different problem sizes. Unlike prior annealers that activate the entire array for energy computation, the proposed annealer only activates specific columns associated with updated spins, thus significantly reducing the overall ADC time cost. Fig. 9(b) shows the time cost trends in solving a 1000-node instance. The other two annealers have identical trends due to the dominant ADC time. The proposed annealer shows much less time, as it eliminates significant ADC executions associated with redundant energy form computations.

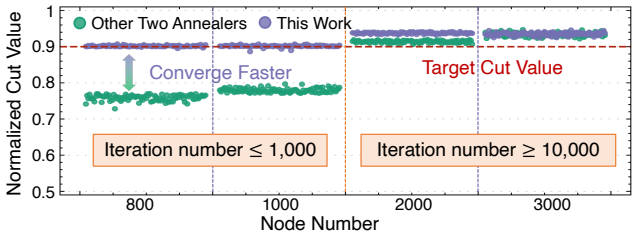


Figure 10: Normalized Cut values solved by the proposed CiM in-situ annealer and other annealers.

Table 1: Summary of COP Solvers

Reference	[39]	[7]	[13]	[15]	[14]	This Work
COP	Max-Cut	Graph Coloring	Knapsack	Quadratic Knapsack	Nash Equilibrium	Max-Cut
Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n)$
e^x Computation	Yes	Yes	Yes	Yes	Yes	No
Hardware Imp.* for Crossbar	Memristor	FeFET	RRAM	FeFET	FeFET	DG FeFET
Problem Size	60 node	21 node	10 node	100 node	104 node	3000 node
Time to Solution	6.6μs	5.1μs‡	3.8μs‡	1.3ms‡	0.08s	4.6ms
Energy to Solution	0.07μJ	0.2μJ‡	-	2.1μJ‡	-	0.9μJ
Average Success Rate (%)	65†	-	92.4†	98.54	81.9	98

*: Imp.: implementation. †: Extracted from literature. ‡: Extracted from literature and estimation at 22nm node.

4.2 Problem Solving Efficiency

We compare the COP solving efficiency of the three CiM annealers. Because CiM/FPGA and CiM/ASIC-based annealers adopt the same algorithm, the results are consistent. We used Monte Carlo sampling to select 100 runs for each Max-Cut problem group, and the normalized Cut values are shown in Fig. 10. The target Cut value is set as 90% of the true optimal value, suggesting a successful instance solving. The results clearly demonstrate that the proposed annealer consistently approaches the optimal solution for all COPs, achieving an impressive average success rate of 98%. In contrast, the other two annealers show a lower average success rate of 50%, only solving the 2000-node and 3000-node problems successfully. Interesting, the fractional annealing factor approximately implemented by DG FeFET's normalized current enables faster convergence than other annealers with exponential annealing factor, achieving higher solving efficiency. The solver summary in Table. 1 shows that the proposed CiM in-situ annealer can solve larger COPs with high energy and time efficiency, achieving a high success rate.

5 CONCLUSION

In this paper, we introduce a ferroelectric CiM in-situ annealer to efficiently solve COPs. We present an incremental-E transformation method that converts a COP into an E_{inc} form, reducing computational complexity and eliminating costly exponential function calculations. We propose a DG FeFET-based CiM crossbar to implement the E_{inc} form by utilizing the unique characteristic and structure of DG FeFETs. Overall, we build a device-algorithm co-design framework that accommodates efficient in-situ E_{inc} computation and annealing flow to solve COPs. Evaluation results show that the proposed annealer significantly reduces energy consumption and time costs while improving the problem solving efficiency compared to prior counterpart solvers.

ACKNOWLEDGMENT

This work was partially supported by NSFC (624B2126, W2412034, 92164203) and SGC Cooperation Project (Grant No. M-0612).

REFERENCES

[1] G. Yu, *Industrial applications of combinatorial optimization.* Springer Science & Business Media, 2013, vol. 16.

[2] V. T. Paschos, *Applications of combinatorial optimization.* John Wiley & Sons, 2014, vol. 3.

[3] G. Naseri and M. A. Koffas, "Application of combinatorial optimization strategies in synthetic biology," *Nature communications*, vol. 11, no. 1, p. 2446, 2020.

[4] F. Barahona, M. Grötschel, M. Jünger *et al.*, "An application of combinatorial optimization to statistical physics and circuit layout design," *Operations Research*, vol. 36, no. 3, pp. 493–513, 1988.

[5] M. Jiang, K. Shan, C. He *et al.*, "Efficient combinatorial optimization by quantum-inspired parallel annealing in analogue memristor crossbar," *Nature Communications*, vol. 14, no. 1, p. 5927, 2023.

[6] W. Yue, T. Zhang, Z. Jing *et al.*, "A scalable universal ising machine based on interaction-centric storage and compute-in-memory," *Nature Electronics*, vol. 7, no. 10, pp. 904–913, 2024.

[7] X. Yin, Y. Qian, A. Vardar *et al.*, "Ferroelectric compute-in-memory annealer for combinatorial optimization problems," *Nature Communications*, vol. 15, no. 1, p. 2419, 2024.

[8] N. Mohseni, P. L. McMahon, and T. Byrnes, "Ising machines as hardware solvers of combinatorial optimization problems," *Nature Reviews Physics*, vol. 4, pp. 363–379, 2022.

[9] K. Katsuki, D. Shin, N. Onizawa *et al.*, "Fast solving complete 2000-node optimization using stochastic-computing simulated annealing," in *2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS).* IEEE, 2022, pp. 1–4.

[10] T. Takemoto, K. Yamamoto, C. Yoshimura *et al.*, "4.6 a 144kb annealing system composed of 9× 16kb annealing processor chips with scalable chip-to-chip connections for large-scale combinatorial optimization problems," in *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 64. IEEE, 2021, pp. 64–66.

[11] W. Moy, I. Ahmed, P.-w. Chiu *et al.*, "A 1,968-node coupled ring oscillator circuit for combinatorial optimization problem solving," *Nature Electronics*, vol. 5, no. 5, pp. 310–317, 2022.

[12] I. Ahmed, P.-W. Chiu, W. Moy *et al.*, "A probabilistic compute fabric based on coupled ring oscillators for solving combinatorial optimization problems," *IEEE Journal of Solid-State Circuits*, vol. 56, no. 9, pp. 2870–2880, 2021.

[13] K. Taoka, N. Misawa, S. Koshino *et al.*, "Simulated annealing algorithm & reram device co-optimization for computation-in-memory," in *2021 IEEE International Memory Workshop (IMW).* IEEE, 2021, pp. 1–4.

[14] Y. Qian, K. Ni, T. Kampfe *et al.*, "C-nash: A novel ferroelectric computing-in-memory architecture for solving mixed strategy nash equilibrium," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[15] Y. Qian, Z. Yang, K. Ni *et al.*, "Hycim: A hybrid computing-in-memory qubo solver for general combinatorial optimization problems with inequality constraints," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[16] Y. Qian, D. Huang, A. Vardar *et al.*, "Ferroelectric compute-in-memory framework for solving pure and mixed strategy nash equilibrium," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2025.

[17] S. Kim, S. Um, W. Jo *et al.*, "Scaling-cim: edram in-memory-computing accelerator with dynamic-scaling adc and adaptive analog operation," *IEEE Journal of Solid-State Circuits*, 2024.

[18] M. A. Hussain, S.-W. Lin, and T.-H. Tsai, "An area-efficient and high throughput hardware implementation of exponent function," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS).* IEEE, 2022, pp. 3369–3372.

[19] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using qubo models," *arXiv preprint arXiv:1811.11538*, 2018.

[20] A. Mallick, Z. Zhao, M. K. Bashar *et al.*, "Cmos-compatible ising machines built using bistable latches coupled through ferroelectric transistor arrays," *Scientific reports*, vol. 13, p. 1515, 2023.

[21] R. Afoakwa, Y. Zhang, U. K. R. Vengalam *et al.*, "Brim: bistable resistively-coupled ising machine," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA).* IEEE, 2021, pp. 749–760.

[22] C. Zhuo, Z. Yang, K. Ni *et al.*, "Design of ultra-compact content addressable memory exploiting 1t-1mtj cell," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[23] Y. Qian, L. Zhao, F. Meng *et al.*, "Enhancing convnets with convfifo: A crossbar pim architecture based on kernel-stationary first-in-first-out dataflow," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 01, pp. 1–12, 2024.

[24] L. Zhao, Y. Qian, F. Meng *et al.*, "Convfifo: A crossbar memory pim architecture for convnets featuring first-in-first-out dataflow," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC).* IEEE, 2024, pp. 824–829.

[25] A. Mondal and A. Srivastava, "In-situ stochastic training of mtj crossbar based neural networks," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2018, pp. 1–6.

[26] X. Yin, Q. Huang, H. E. Barkam *et al.*, "A homogeneous fefet-based time-domain compute-in-memory fabric for matrix-vector multiplication and associative search," *IEEE TCAD*, 2024.

[27] K. Ni, X. Yin, A. F. Laguna *et al.*, "Ferroelectric ternary content-addressable memory for one-shot learning," *Nature Electronics*, vol. 2, no. 11, pp. 521–529, 2019.

[28] H. Xu, J. Yang, T. Kämpfe *et al.*, "On the challenges and design mitigations of single transistor ferroelectric content addressable memory," *IEEE EDL*, 2023.

[29] J. Cai, H. E. Barkam, M. Imani *et al.*, "A scalable 2t-1fefet based content addressable memory design for energy efficient data search," *IEEE TCAD*, 2024.

[30] X. Yin, F. Müller, Q. Huang *et al.*, "An ultracompact single-ferroelectric field-effect transistor binary and multibit associative search engine," *Advanced Intelligent Systems*, vol. 5, no. 7, p. 2200428, 2023.

[31] X. Yin, F. Müller, A. F. Laguna *et al.*, "Deep random forest with ferroelectric analog content addressable memory," *Science Advances*, vol. 10, p. eadk8471, 2024.

[32] X. S. Hu, M. Niemier, A. Kazemi *et al.*, "In-memory computing with associative memories: A cross-layer perspective," in *2021 IEEE International Electron Devices Meeting (IEDM).* IEEE, 2021, pp. 25–2.

[33] X. Yin, Y. Qian, M. Imani *et al.*, "Ferroelectric ternary content addressable memories for energy-efficient associative search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 4, pp. 1099–1112, 2022.

[34] Z. Jiang, Y. Xiao, S. Chatterjee *et al.*, "Asymmetric double-gate ferroelectric fet to decouple the tradeoff between thickness scaling and memory window," in *2022 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits).* IEEE, 2022, pp. 395–396.

[35] K. Ni, M. Jerry, J. A. Smith *et al.*, "A circuit compatible accurate compact model for ferroelectric-fets," in *2018 IEEE symposium on VLSI technology.* IEEE, 2018, pp. 131–132.

[36] J. Liu, X. Tang, W. Zhao *et al.*, "16.5 a 13b 0.005 mm 2 40ms/s sar adc with kt/c noise cancellation," in *2020 IEEE International Solid-State Circuits Conference-(ISSCC).* IEEE, 2020, pp. 258–260.

[37] M. Poremba, S. Mittal, D. Li *et al.*, "Destiny: A tool for modeling emerging 3d nvm and edram caches," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, 2015, pp. 1543–1546.

[38] "Stanford Max-Cut dataset," https://web.stanford.edu/~yyye/yyye/Gset/.

[39] F. Cai, S. Kumar, T. Van Vaerenbergh *et al.*, "Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks," *Nature Electronics*, vol. 3, no. 7, pp. 409–418, 2020.

ADVeRL-ELF: *ADVe*rsarial *ELF* Malware Generation using *R*einforcement *L*earning

Akshara Ravi[1], Vivek Chaturvedi[1], Muhammad Shafique[2]
[1]*Department of CSE, Indian Institute of Technology*, Palakkad, India
[2]*eBRAIN Lab, Division of Engineering, New York University*, Abu Dhabi, UAE

***Abstract*—Deep learning models are now pervasive in the malware detection domain owing to their high accuracy and performance efficiency. However, it is critical to analyze the robustness of these models by introducing adversarial attacks that can expose their vulnerabilities. Nevertheless, adversarial malware generation problem for Linux has not been well-investigated. In this work, we propose a novel reinforcement learning framework, ADVeRL-ELF to generate adversarial ELF malware by adding semantic NOPs within the executable region. Experimental results show that ADVeRL-ELF achieved an attack success rate of 59.5%. These adversarial malware can be leveraged to harden the Linux based malware detection systems.**

***Index Terms*—adversarial malware, reinforcement learning, Linux ELF, semantic NOPs**

I. Introduction

Malware or malicious software is any software that is introduced intentionally to the computer systems or servers to steal confidential data, disrupt services, or corrupt the system files. Malware has been a serious threat to the security landscape of business enterprises, banking, and e-commerce over the past few decades. Although security experts have developed novel and effective malware detection techniques, malware continues to be a growing concern. According to SonicWall's 2024 Mid-Year Cyber Threat Report [1], there is a surge in malware attacks (30%) in the first half of 2024. Recent years have witnessed a substantial growth of Linux malware as shown in Fig. 1. One of the main reasons for this explosion is attributed to the proliferation of IoT devices such as smart cameras, smart watches, and smart home appliances. According to IoT Analytics [3], the number of connected IoT devices is expected to reach 18.8 billion by the end of 2024. These devices are the weakest points in a network with limited resources and lapse security exacerbated by constant online connection, default credentials, and open ports. Attackers use these compromised IoT devices as tools for launching cyberattacks such as Distributed Denial of Service (DDoS), cryptomining, and phishing, thus translating to an enormous 107% increase in IoT malware attacks in 2024 [1]. To combat this escalating threat, there is a pressing need for innovative and robust malware detection techniques for Linux.

Apart from the commercial antivirus, state-of-the-art malware detection tools are increasingly exploiting the machine learning (ML) and deep learning (DL) based static and dynamic malware analysis techniques. However, these neural networks are not completely secure and are known to be

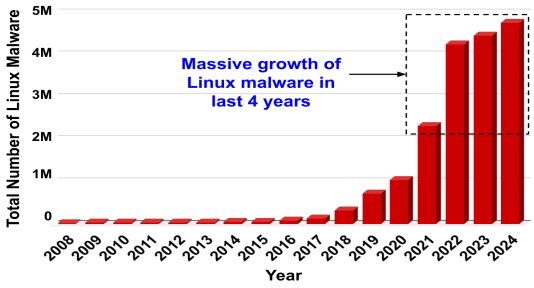


Fig. 1: Growth of Linux malware from 2008 to 2024 [2]

vulnerable to adversarial samples. Adversarial examples (AEs) are maliciously crafted inputs generated by adding small perturbations such that the neural network misclassifies the sample with high confidence. Adversarial attacks were first introduced in the image classification domain [4], where the goal is to make AEs imperceptible to human eyes. Fast Gradient Sign Method (FGSM) [4] and Projected Gradient Descent (PGD) [5] are the most popular adversarial attacks against images. These AEs are used for performing the vulnerability analysis to harden the models. *However, these attacks cannot be directly applied to the malware problem space due to the inverse feature mapping problem [6] illustrated in Fig. 2.*

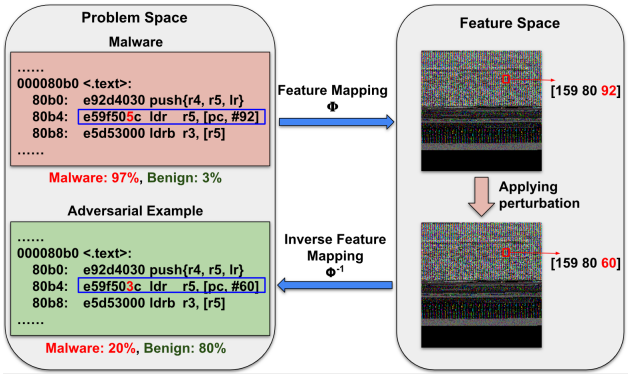


Fig. 2: Illustration of how adding perturbations in the feature space does not translate to adversarial executables with the same functionality in the problem space

In this scenario, malware executables constitute the problem space and images generated from their bytecodes represent the feature space of malware image-based detection models. Other feature spaces for malware include API calls, header features, and opcode sequences. When we apply adversarial attack such as FGSM on these malware images, it will modify the pixel values. For example, the pixel value at the position

bounded by red box is changed from [159 80 92] to [159 80 60]. Here, only the B channel is modified in the feature space. The generated adversarial image looks exactly similar to the original one and successfully evades the malware detection model. Nevertheless, we could see that the inverse feature mapping function ϕ^{-1} is neither invertible nor differentiable [6] i.e., the single byte modification translates to a different instruction (shown in blue boxes) altering the functionality of the executable. Thus, the perturbations added in the feature space should be carefully crafted such that they do not break the executable. Practically, it is better to impose adversarial perturbations in the problem space. A successful adversarial malware should evade the malware detection model and maintain the malicious and executable functionality [7].

Adversarial malware generation (AMG) techniques for Linux-based operating systems are not well explored. In this work, we propose a novel reinforcement learning (RL) framework ADVeRL-ELF for AMG, where the generated AEs satisfy the malicious and executable functionalities. We have chosen a RL approach since it follows a sequential decision-making process that enables us to generate AEs with minimum perturbation [8]. Further, RL can effectively handle the discrete perturbation space of malware detection domain [9]. ADVeRL-ELF works by inserting semantic NOPs (NO Operations) in the *.text* section of Linux executables. *To the best of our knowledge, this work is a pioneering contribution in AMG for IoT binaries with Executable and Linkable Format (ELF) that add perturbations in the executable region.* To guide the evasion process with a minimum number of NOP insertions, we extract the GradCAM region from the malware images.

Our novel contributions are listed below:

- Developed a novel reinforcement learning framework ADVeRL-ELF for generating adversarial examples that can be leveraged to make the Linux based malware detection systems robust against adversarial attacks.
- Focused on IoT malware with Linux ELF format. State-of-the-art works primarily focus on adversarial Windows PE malware generation.
- In contrast to existing works that alter the non-executable regions, we added perturbations in the executable *.text* section of ELF by adding semantic NOPs.
- Demonstrated that a diverse action space can be advantageous against preprocessing filters that easily filter out naive NOPs for instance, 0x90 (NOP) in Windows and 0xe1a00000 (MOV r0, r0) in Linux ARM).
- Achieved an attack success rate of 59.5%.

Paper Organization: Section 2 discusses the background and related work. Section 3 elaborates on the motivational study. Our proposed methodology is discussed in Section 4. Section 5 summarizes the experiments and results. Finally, Section 6 concludes the paper.

II. Background and Related Work

There is an exponential growth of malware in recent years owing to the surge in computational resources and advanced technologies. This sheer increase in the number of malware places an immense burden on security teams, and hence they opt for automated malware detection techniques using ML/DL. For instance, ELF-Miner [10] and NtMalDetect [11] utilize ELF header features and system call traces, respectively, for detecting malware. Regardless of how smart and effective these solutions are, adversaries may uncover new adversarial attacks to evade these malware detection techniques.

AMG techniques in the literature mainly leverages iteration based on gradient [12] and RL to apply perturbations. COPYCAT [13] generates adversarial examples using two main configurations; appending the AE generated using techniques such as FGSM, MIM, C&W, DeepFool, and PGD attack and injecting the samples from the target class at the end. RL-based adversarial attacks have recently gained popularity due to the greater potential of RL in the context of black-box attacks [14] [15]. Some of the actions used by RL agents in the literature [16] [17] [18] [19] include appending random bytes at the end, adding random sections, creating a new entry point that immediately jumps to the original entry point, etc. Majority of these attacks alter the non-executable regions of Windows PE. This ensures the executable and malicious functionality of the malware. However, these perturbations are prone to easy detection by a preprocessing filtering that searches these obvious places where perturbations can be hidden. Daniel et al., [20] introduced a RL framework based on Double Q-network to select the optimal positions to add naive NOPs in the *.text* section. However, naive NOPs can be effectively filtered out by preprocessing filters, thereby reducing the effectiveness of the approach. Further, most malware detection models utilize only the first "x" bytes of the executable rendering the attacks that append at the end of the binary meaningless.

Adversarial attacks in the literature primarily focus on Windows PE or Android binaries. This could be due to the availability of open source tools to parse and edit PE files and the misconception that Linux systems are secure. EvilELF [21] applies 5 methods to modify ELF binaries, such as header, debug, and padding alteration, end appendix, and dynamic extension. Xue et al., [22] applied the Proximal Policy Optimization (PPO) RL algorithm to generate adversarial malware by applying 11 actions. These works alter the non-executable regions of ELF binary. In order to fill this gap, we propose an adversarial attack that hides the perturbations in the executable region of the ELF binaries. Towards that, we conducted a motivational study comparing our proposed framework with other state-of-the-art as shown in Table I. From Table I, we can clearly observe that our work is the first adversarial attack that alters the executable region of the ELF binaries.

III. Motivational Case Study

First, we have conducted three preliminary experiments that motivated us to leverage a reinforcement learning approach for adversarial malware generation. We have selected the image-based malware detection approach, as it tends to take less feature extraction time than other static approaches like header features or opcodes [24]. Due to the popularity of residual networks in the image-based malware detection domain [25]

TABLE I: Summary of existing adversarial malware attacks

Work	Technique	Perturbation in executable region ?	Target binary	Executable & malicious ?
FGAM [7]	Gradient	✗	PE	✓
COPYCAT [13]	Gradient	✗	PE, IoT	✗
Daniel et al., [20]	RL	✓	PE	✓
EvilELF [21]	Random bytes	✗	ELF	✓
Xue et al., [22]	RL	✗	ELF	✓
Qiao et al., [23]	Gradient	✗	ELF	✓
ADVeRL-ELF	RL	✓	ELF	✓

[26] owing to its capability in capturing complex image features, we assume the target model to be ResNet18. Input for this model will be the first 150528 bytes of the executable that correspond to a (224,224) RGB image, a standard size for residual networks. We chose the first part of the image since these portions include header regions that are considered imperative by state-of-the-art malware detection models. Experiments in this section were carried out on a sample dataset of 2572 benign and 3000 malware samples. ResNet18 achieved a test accuracy of 96.22% on original malware samples. Next, we discuss the three motivational studies in detail.

A. Applying adversarial attacks in the malware image domain

We have applied popular adversarial attacks such as FGSM and PGD in the image domain. Results are shown in Fig. 3. Here, for both FGSM and PGD, we obtain an attack success rate (ASR) of 100% with $\epsilon = 0.1$, where ϵ bounds the amount of perturbation added to input. Resulting perturbation rate is 62%-79% which is not desirable for our case.

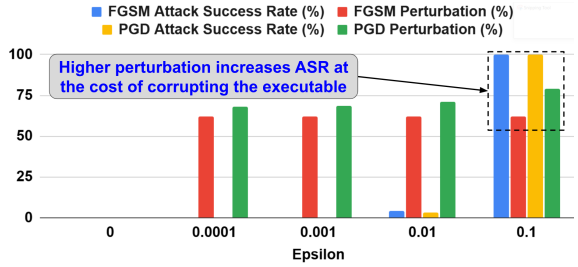


Fig. 3: Attack success rate and perturbation rate when applying FGSM and PGD attacks to malware image domain

B. Adding random benign patches from the dataset

We need to add perturbations such that the malicious functionality is not altered. One approach is to add benign patches to the malware. We need to address 3 questions here.

1) How to get these patches?: In this study, we have used GradCAM (Gradient-weighted Class Activation Mapping) to select a (32,32,3) portion of benignware using ResNet18. It uses the gradients of the input image flowing into the final convolutional layer to produce a heat map that highlights the important regions in the image that influence the model decision [27]. Thus, these regions in benignware will incorporate significant benign characteristics.

2) Where to insert these patches?: We will insert these benign patches to the GradCAM region of malware so that the malicious probability can reduce at a faster rate. However, if the GradCAM position falls within a section, we cannot add the patch exactly at that point, since it will corrupt the executable. Hence, we found the section where the GradCAM lies and added a new non-executable section of size 3072 bytes (to add (32,32,3) benign patch) before that. Thus, the applied perturbation rate is constant.

3) How to add these patches?: First, we randomly selected benign patches for each malware image and repeated the experiment for 10 runs. The average ASR is shown in Table II.

C. Adding random semantic NOPs

Mostly, the GradCAM region falls in the *.text* section of a binary since this section encompasses the actual instructions that determine the malicious or benign behavior and hence mostly affect the model's decision. Still, we cannot add the extracted benign patches or any random perturbation at these positions. In such a case, semantic NOPs, instructions that "do nothing" to the execution flow come to our aid. We have chosen ARM as the use case here, since it is the most predominant architecture for IoT. We have downloaded the list of semantic NOPs for the ARM architecture from the github link [28]. Now, we need to decide on

1) Where to add these NOPs?: We again choose the GradCAM region. For a (32,32,3) GradCAM region, we have 32 positions to add semantic NOP sequences of length (32*3=96 bytes). First, we add random semantic NOP sequences at all 32 positions of malware whose GradCAM lies within the *.text* section. Then, we combined benign patch and semantic NOP methods by adding a random benign patch if the GradCAM lies outside the *.text* section. Table II shows the results.

TABLE II: Average attack success rate (ASR) over 10 runs for adding benign patches and semantic NOPs randomly

Approach	ASR (%)
Add a random benign patch for all malware	33.22
Add random semantic NOP sequences of length 96 bytes at 32 positions in the GradCAM region	0
Add random semantic NOP sequences in the GradCAM region if *.text* section else benign patch	22.57

2) How many NOPs should be added ?: From Table II, we could see that even after adding random semantic NOP sequences of length 3072 (32*32*3) bytes, ASR was 0%. We have conducted experiments to investigate whether we can increase the attack success rate by increasing the number of semantic NOPs. However, ASR remained around 20%-26% while adding 1 to 32 semantic NOPs for the case where both semantic NOPs and benign patches are added based on the GradCAM position. Thus, randomly adding semantic NOPs even though reduces the malicious probability, it is not enough to change the classification label. Hence, there is a need for an approach that investigates the effect of each semantic NOP on the malicious score of an executable and decide based on its knowledge which NOP to use. Our ADVeRL-ELF framework discussed next solves this problem.

IV. Proposed Methodology

This section presents the overall workflow of the adversarial attack process shown in Fig. 4. It consists of 3 major phases: ADVeRL-ELF framework that generates adversarial ELF malware, ELF manipulation procedure that adds semantic NOPs to the executable, and target malware detector that evaluates the evasion capability of the generated adversarial samples. First, we examine the threat model of the attack.

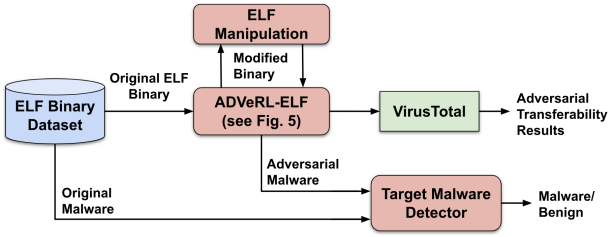


Fig. 4: Overall workflow of the adversarial attack process

A. Threat Model

We assume that the target malware detector uses a visualization-based static analysis approach to detect malware. Specifically, we make the following assumptions.

1) **Adversary's Goal**: Generate adversarial ELF malware samples that are misclassified as benign by the target model and satisfy the malicious and executability properties. We are interested in non-targeted attack that transforms malware to benign class and not vice versa.
2) **Adversary's Capabilities**: The adversary has access to a set of malicious ELF samples. Further, the adversary has knowledge about the structure of the target malware detector. This helps the adversary estimate the GradCAM region from the last convolutional layer of the model.

B. ADVeRL-ELF Framework

Our proposed adversarial ELF malware generation framework, ADVeRL-ELF shown in Fig. 5, employs the Asynchronous Advantage Actor-Critic (A3C) algorithm [29]. A3C is a policy gradient-based deep reinforcement learning algorithm in which the actor and the critic agents work in tandem to optimize the policy. We will explain the rationale for choosing A3C in Section V.

Here, the environment consists of ELF malicious binaries and the target malware detector, which could be any neural network or antivirus scanners (here, ResNet18). State space of the environment is represented by a (224,224) RGB malware image. We generate RGB images from the bytecode of executables by grouping it into 24 bit sequences of red, green, and blue channels (8 bits each with each byte representing integers in the range [0, 255]). Exploring a high dimensional action space with 183375 NOPs from [28] is cumbersome. We identified that they can be scaled down to 120 NOPs with the below 5 operations that constitute our action space.

[REVxx REVxx], [REV16xx REV16xx], [ADDxx SUBxx], [SUBxx ADDxx], [EORxx EORxx]

where xx corresponds to the condition codes EQ, NE, LT, GT, LE, GE, CS, CC, MI, PL, VS, VC, HI, LS or AL. The

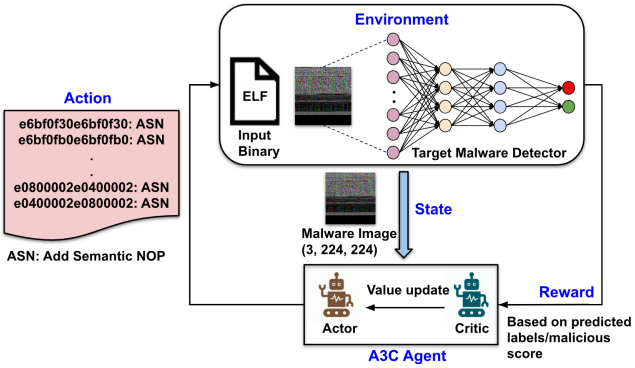


Fig. 5: Proposed ADVeRL-ELF Framework

operands for these operations can be 16 registers R0-R15 in ARM32 and immediate values 0-255.

The actor component of A3C RL agent chooses an 8 byte semantic NOP action from list and inserts it to the GradCAM position in the *.text* section of the given binary as illustrated in Fig. 6. Then the RGB image corresponding to the modified binary is given as input to ResNet18. Based on the response of the malware detector, we designed two reward functions r_t.

1) Based on label: Access to the predicted labels only; 0 (Benignware) and 1 (Malware)

$$r_t = \begin{cases} 10 & \text{if } label_{s_t} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

2) Based on score: Access to the malicious score. Reward is calculated based on malicious threshold as:

$$r_t = \begin{cases} 10 & \text{if } mal_score_{s_t} < mal_threshold \\ (mal_score_{original} - mal_score_{s_t}) & \text{otherwise} \end{cases} \tag{2}$$

Here, if the malicious probability of binary after adding a semantic NOP is decreasing, then we assign a positive reward proportional to the rate of decrease to the action and vice versa. Despite the fact that a high threshold (90%) can increase the ASR, we chose a hard threshold of 50% since a higher threshold will result in higher false negative rate (missed threats) that can be catastrophic.

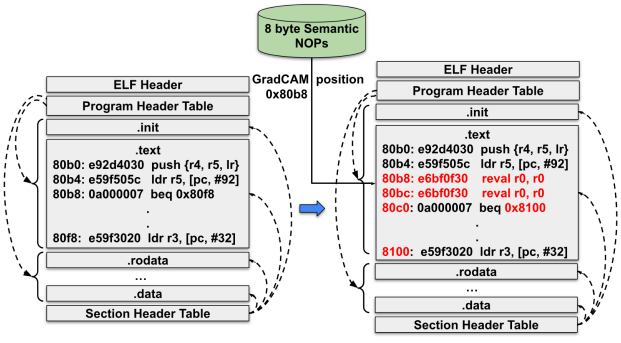


Fig. 6: Overall workflow of adding semantic NOPs

Finally, critic will evaluate how good the selected action was in driving towards our goal i.e., benign based on the reward and state and update the information to actor. Actor will further continue the step by selecting the next action. This process

continues until the malware evades the target malware detector or maximum turns are reached. Algorithm 1 provides a high-level overview of the training procedure for label reward. Here, *IMAGE_FEATURE_EXTRACT()* converts malware bytecode to (224,224) RGB image and *ADD_SEMANTIC_NOP()* edits the ELF executable by adding the semantic NOP corresponding to action a_t using the steps discussed next.

C. ELF Binary Manipulation

In our work, we focus on generating AEs for Linux-based IoT devices. All ARM instructions are 32 bits (4 byte) long, that will expedite our process to identify the instruction boundaries to insert NOPs. We utilize NOPs of length 8 bytes. For adding semantic NOPs in the *.text* section, we leverage *lief* [30], a cross platform library that can parse and edit ELF, PE, and MachO formats. However, the tool does not support some ELF manipulations which were handled by our custom script. In the following, we review the steps to edit ELF executable.

1) Parse the ELF executable using *lief*
2) Find the GradCAM position within the *.text* section
3) Convert the semantic NOP to little endian if EI_DATA field in ELF header is "ELFDATA2LSB"
4) Shift the instructions in *.text* section and the later sections and add the 8 byte semantic NOP
5) Update the section size of *.text* section and handle alignment properties.
6) Update the offset and virtual address of the sections after the *.text* section
7) Rebuild the segments
8) Update the jump and branch addresses that refer to the instructions after the semantic NOP insertion point
9) Finally, write the modified binary

The target malware detector predicts whether the generated adversarial example is malware or benign and thus determines the evasion capability. We further analyze the adversarial transferability of samples against VirusTotal [31]. Next, we will discuss the experiments and the results achieved.

V. Experiments and Results

A. Dataset

In our experiments, we have used the ARM binaries of the IoT malware dataset, IoTPoT [32]. We crawled the GitHub repositories and Linux directories */bin*, */sbin*, */usr/bin*, and */usr/local/bin* of Raspberry Pi and Jetson Orin Nano to prepare the benign dataset.

B. Experimental Setup and Results

We conducted the experiments on an Intel® Core™ i7-10700 CPU @ 2.90GHz × 16 processor running Ubuntu 20.04.6 LTS with 32GB RAM and Python 3.8.10. Lief 0.12.3 was used to edit the ELF executable and chainerrl 0.8.0 was used to build the A3C network. We have further utilized the OpenAI gym framework [33] that enables us to generate custom malware evasion environments for training and evaluating our RL algorithms. First, we trained our target malware detector ResNet18 for 50 epochs and achieved a test accuracy

Algorithm 1 Adversarial Malware Generation

Input: Malicious ELF samples set *M*, *MAX_TURNS*, Agent *A3C*, Action Table *A*, Target Malware Detector *Mal_Detect*, True labels *y_true*
Output: Adversarial malware samples set *Adv_M*

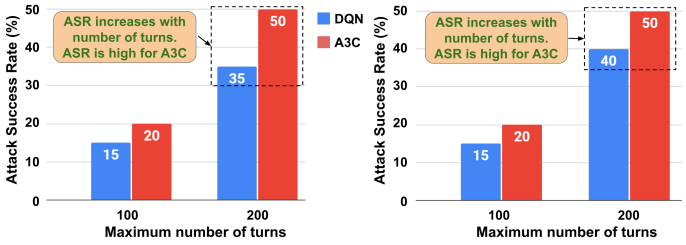
Initialize *A3C*, *gym_malenv*
reward = 0, *Adv_M* = ϕ
for *m* in *M* **do**
  $label_{orig}$ = PREDICT_LABEL(m)
  if $label_{orig}$!= *y_true*[*m*] **then**
    "Already Benign"
    break
  for i=1 to *MAX_TURNS* **do**
    Choose $a_t \in A$
    m_{t+1} = ADD_SEMANTIC_NOP(m_t, a_t)
    $label_{t+1}$ = PREDICT_LABEL(m_{t+1})
    if $label_{t+1}$!= *y_true*[*m*] **then**
      reward = 10
      Adv_M = *Adv_M* $\cup$ m_{t+1}
    else
      reward = 0
return *Adv_M*

Function PREDICT_LABEL(episode)

s = IMAGE_FEATURE_EXTRACT(episode)
label = *Mal_Detect*.predict(s)
return label

of 99.69%. However, our ADVeRL-ELF framework focuses on generating adversarial malware, and therefore, we consider the malware samples for our further experiments.

Initial set of experiments were carried out to determine the RL algorithm that works well on our problem space. Deep Q-learning (DQN) and A3C algorithms were initially trained on a sample malware dataset of 100 training samples and 20 test samples. We have applied Adam optimizer. For DQN, we have adopted the LinearDecayEpsilonGreedy explorer and prioritized experience replay. Both the label and score rewards given, respectively, in equations 1 and 2 were considered for our experiments. Results are shown in Fig. 7 for 10000 steps and 100 semantic NOPs. We have chosen the attack success rate metric that depicts the percentage of adversarial malware that successfully evaded the target malware detector out of the total adversarial samples to evaluate our framework.



(a) Based on malicious score (b) Based on label

Fig. 7: Attack success rate for DQN and A3C RL algorithms

We have further evaluated the training time of DQN and

A3C. We could see that DQN takes around 43 hours for 50000 steps, whereas A3C takes only 16 hours. We have further conducted experiments by increasing the number of samples and training steps while keeping the number of actions as 100 randomly. Table III summarizes the ASR achieved for 50000 steps and 200 maximum turns.

TABLE III: Attack success rate for increasing samples

Dataset	Reward Function	
	Label (%)	Malicious Score (%)
1102 training and 179 holdout	53.07	54.19
3784 training and 687 holdout	44.69	42.94

From Fig. 7 and Table III, we can infer that the achieved ASR is around 40-55%. When it comes to malware detection, missing out on even a single malware sample can be risky, and hence this evasion rate is problematic for any malware detector. Further, we could see that the ASR is reducing as the number of samples increases. This is because the number of steps must be increased to give all samples enough chance of evasion. Fig. 8 shows the effect of increasing turns on attack success rate. When the number of turns per episode increases, the perturbation increases, resulting in increased ASR.

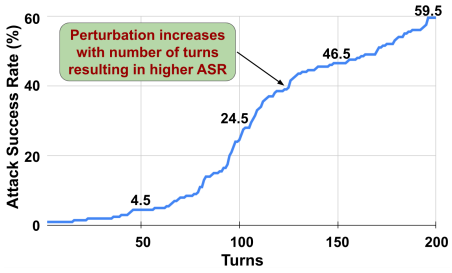


Fig. 8: Number of turns v/s Attack Success Rate

Each turn will add 8 byte semantic NOP and hence the perturbation of a sample for 200 turns will be 1600 (200*8) bytes. This trade-off between ASR and perturbation rate is imperative. A higher perturbation rate can significantly increase the file size that can cause the sample to lose its concealment ability. Thus, we keep the maximum allowed number of turns as 200. We perform our final set of experiments on a dataset of 1000 training and 200 test samples for 1,00,000 steps. Further, we modify the action space to include the 120 semantic NOPs discussed in Section IV-B. Results are shown in Table IV. Out of 200, 119 samples successfully evaded ResNet18.

TABLE IV: Final attack success rate for 200 turns

Reward	Based on label	Based on score
Attack Success Rate (%)	58	59.5

We have submitted these 119 original and adversarial malware to VirusTotal to analyze adversarial transferability against commercial antivirus. Fig. 9 shows the number of antiviruses out of total 64 that can correctly detect maliciousness of the 119 samples on VirusTotal before and after applying our adversarial attack. We could see that on average the number of antiviruses that can detect maliciousness is reduced from 39 to 29 with the maximum reduction in the number of antiviruses from 37 to 8. Finally, we have compared our ADVeRL-ELF framework against the state-of-the-art approach for adversarial

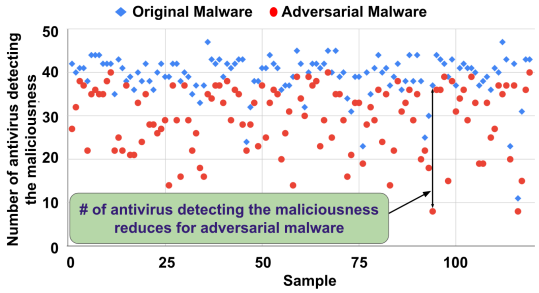


Fig. 9: VirusTotal results on original and adversarial malware

ELF attacks. Table V shows the results. Here, comprehensive disturbance adds arbitrary bytes generated by FGSM attack at the end and middle of an ELF file. Our ADVeRL-ELF framework achieved a better attack success rate of 59.5%.

TABLE V: Comparison with state-of-the-art

Approach	Attack success rate (%)
Comprehensive Disturbance [23]	26.2
ADVeRL-ELF	**59.5**

Based on the experimental findings, we could determine that ADVeRL-ELF can generate adversarial ELF malware by adding perturbations in the executable region. This is in contrast to the works that add perturbations in the non-executable areas of binary. Thus, ADVeRL-ELF generates AEs that maintain malicious and executable functionality, and these AEs can be used to strengthen existing malware detection systems. Moreover, this framework can be easily extended to state-of-the-art neural networks such as vision transformers and MobileNet and other CPU architectures. For instance, we can utilize the semantic NOPs for MIPS and x86 from the same github repository [28] and ADVeRL-ELF to generate adversarial malware for these architectures.

VI. Conclusion and Future Work

In this work, we proposed a novel framework ADVeRL-ELF that addresses the existing problem of adversarial ELF malware generation. ADVeRL-ELF generates adversarial malware by adding semantic NOPs in the GradCAM position within the executable region. We have achieved an attack success rate of 59.5% against our target malware detector ResNet18. Further, the average number of virus detection tools that could detect the sample's maliciousness on VirusTotal is reduced from 39 to 29. This work can be further extended to apply binary diversification techniques such as instruction substitution, code displacement, etc. in the executable region. Further, adversarial malware that can evade dynamic malware analysis techniques can be an interesting and challenging avenue to explore.

Acknowledgements

This work was supported in part by the NYUAD Center for Cyber Security (CCS), funded by Tamkeen under the NYUAD Research Institute Award G1104. The authors acknowledge Dr. Amira Guesmi for her contributions, discussions, and feedback on the concept development and methodology based on reinforcement learning, action space refinement, and training optimizations used in this work.

REFERENCES

[1] James Coker, "Malware Attacks Surge 30% in First Half of 2024," Infosecurity Magazine. Available: https://www.infosecurity-magazine.com/news/malware-attacks-surge-30-per-cent/. Accessed: October, 2024.

[2] AV-Atlas, "Total amount of malware and PUA under Linux," Available: https://portal.av-atlas.org/malware/statistics. Accessed: October, 2024.

[3] Satyajit Sinha, "State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally," IoT Analytics. Available: https://iot-analytics.com/number-connected-iot-devices/. Accessed: October, 2024.

[4] Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). "Explaining and harnessing adversarial examples". arXiv preprint arXiv:1412.6572.

[5] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). "Towards deep learning models resistant to adversarial attacks". stat, 1050(9).

[6] Zhan, D., Duan, Y., Hu, Y., Li, W., Guo, S., & Pan, Z. (2023). "MalPatch: Evading DNN-Based Malware Detection With Adversarial Patches". IEEE Transactions on Information Forensics and Security, 19, 1183-1198.

[7] Li, K., Zhang, F., & Guo, W. (2023). "FGAM: Fast Adversarial Malware Generation Method Based on Gradient Sign". arXiv preprint arXiv:2305.12770.

[8] Song, W., Li, X., Afroz, S., Garg, D., Kuznetsov, D., & Yin, H. (2022, May). "Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware". In Proceedings of the 2022 ACM on Asia conference on computer and communications security (pp. 990-1003).

[9] Zhang, L., Liu, P., Choi, Y. H., & Chen, P. (2022). "Semantics-preserving reinforcement learning attack against graph neural networks for malware detection". IEEE Transactions on Dependable and Secure Computing, 20(2), 1390-1402.

[10] Shahzad, F., & Farooq, M. (2012). "Elf-miner: Using structural knowledge and data mining methods to detect new (linux) malicious executables". Knowledge and information systems, 30, 589-612.

[11] Kim, C. W. (2018). "Ntmaldetect: A machine learning approach to malware detection using native api system calls". arXiv preprint arXiv:1802.05412.

[12] Park, D., Khan, H., & Yener, B. (2019, December). "Generation & evaluation of adversarial examples for malware obfuscation". In 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA) (pp. 1283-1290). IEEE.

[13] Khormali, A., Abusnaina, A., Chen, S., Nyang, D., & Mohaisen, A. (2019). "COPYCAT: practical adversarial attacks on visualization-based malware detection". arXiv preprint arXiv:1909.09735.

[14] Fang, Y., Zeng, Y., Li, B., Liu, L., & Zhang, L. (2020). "DeepDetectNet vs RLAttackNet: An adversarial method to improve deep learning-based static malware detection model". Plos one, 15(4), e0231626.

[15] Zhong, F., Hu, P., Zhang, G., Li, H., & Cheng, X. (2022). "Reinforcement learning based adversarial malware example generation against black-box detectors". Computers & Security, 121, 102869.

[16] Etter, B., Hu, J. L., Ebrahimi, M., Li, W., Li, X., & Chen, H. (2023, December). "Evading Deep Learning-Based Malware Detectors via Obfuscation: A Deep Reinforcement Learning Approach". In 2023 IEEE International Conference on Data Mining (ICDM) (pp. 101-109). IEEE.

[17] Fang, Z., Wang, J., Geng, J., Zhou, Y., & Kan, X. (2021). "A3CMal: Generating adversarial samples to force targeted misclassification by reinforcement learning". Applied Soft Computing, 109, 107505.

[18] Do Thi Thu, H., Phan, T. D., Le Anh, H., Nguyen Duy, L., Nghi Hoang, K., & Pham, V. H. (2022, December). "A Method of Mutating Windows Malwares using Reinforcement Learning with Functionality Preservation". In Proceedings of the 11th International Symposium on Information and Communication Technology (pp. 142-149).

[19] Wu, C., Shi, J., Yang, Y., & Li, W. (2018, November). "Enhancing machine learning based malware detection model by reinforcement learning". In Proceedings of the 8th International Conference on Communication and Network Security (pp. 74-78).

[20] Gibert, D., Fredrikson, M., Mateu, C., Planes, J., & Le, Q. (2022). "Enhancing the insertion of NOP instructions to obfuscate malware via deep reinforcement learning". Computers & Security, 113, 102543.

[21] Kosikowski, A., Cho, D., Ninan, M., Ralescu, A., & Wang, B. (2023, December). "EvilELF: Evasion Attacks on Deep-Learning Malware Detection over ELF Files". In 2023 International Conference on Machine Learning and Applications (ICMLA) (pp. 1702-1709). IEEE.

[22] Xue, M., Fu, J., Li, Z., Ni, S., Wu, H., Zhang, L. Y., ... & Liu, W. (2024). "A Reinforcement Learning-based ELF Adversarial Malicious Sample Generation Method". IEEE Journal on Emerging and Selected Topics in Circuits and Systems.

[23] Qiao, Y., Zhang, W., Tian, Z., Yang, L. T., Liu, Y., & Alazab, M. (2022). "Adversarial ELF malware detection method using model interpretation". IEEE Transactions on Industrial Informatics, 19(1), 605-615.

[24] Ravi, A., Chaturvedi, V., & Shafique, M. (2023). "ViT4Mal: Lightweight Vision Transformer for Malware Detection on Edge Devices". ACM Transactions on Embedded Computing Systems, 22(5s), 1-26.

[25] Khan, R. U., Zhang, X., & Kumar, R. (2019). "Analysis of ResNet and GoogleNet models for malware detection". Journal of Computer Virology and Hacking Techniques, 15, 29-37.

[26] Manavi, F., & Hamzeh, A. (2020, September). "A new method for ransomware detection based on PE header using convolutional neural networks". In 2020 17th international ISC conference on information security and cryptology (ISCISC) (pp. 82-87). IEEE.

[27] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2017). "Grad-cam: Visual explanations from deep networks via gradient-based localization". In Proceedings of the IEEE international conference on computer vision (pp. 618-626).

[28] bogwonch, "Semantic-NOPs," GitHub repository, 2011. [Online]. Available: https://github.com/bogwonch/Semantic-NOPs.

[29] AlMahamid, F., & Grolinger, K. (2021, September). "Reinforcement learning algorithms: An overview and classification". In 2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) (pp. 1-7). IEEE.

[30] Romain Thomas, "LIEF-Library to Instrument Executable Formats." Available: https://lief.quarkslab.com/. Accessed: April, 2024.

[31] VirusTotal, Available: https://www.virustotal.com/. Accessed: August, 2024.

[32] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., & Rossow, C. (2016). "IoTPOT: A novel honeypot for revealing current IoT threats". Journal of Information Processing, 24(3), 522-533.

[33] Anderson, H. S., Kharkar, A., Filar, B., Evans, D., & Roth, P. (2018). "Learning to evade static pe machine learning malware models via reinforcement learning". arXiv preprint arXiv:1801.08917.

GraphFI: An Efficient Fault Injection Framework for Graph Processing on GPGPUs

Nan Jiang*, Hengshan Yue*, Jingweijia Tan*, Mengting Zhou*, Xiaonan Wang*,
Yuchun Wang*, Wenda Wei*, Meikang Qiu†, and Xiaohui Wei*
*College of Computer Science and Technology, Jilin University
†School of Computer and Cyber Science, Augusta University

***Abstract*—As graph tasks become pervasive in real-time and safety-critical domains (e.g., financial fraud detection and electrical power systems), it is also essential to guarantee their reliable execution beyond pursuing extraordinary performance. However, due to the neglect of consideration for graph-specific execution paradigm, existing Fault Injection (FI) reliability analysis methods typically incur inaccurate system error resilience characterization, making it challenging to provide helpful guidance for efficient and reliable graph processing paradigm design.**

This paper proposes *GraphFI*, an efficient *Graph Fault Injection* framework on the universal parallel tasks acceleration platform (i.e., GPGPUs). Our key insight is progressively excavating the graph-specific error propagation and effect mechanisms, thereby avoiding blind FI trials. Firstly, observing that iterations with similar active vertex set exhibit similar error behavior, we propose *iteration-driven GraphFI* (*ID-GraphFI*) to solely select representative iterations for fast error resilience profile assessment. Secondly, by detecting resilience-similarity communities in graph topology, we propose *topology-driven GraphFI* (*TD-GraphFI*) that only selects representative vertices for community overall reliability evaluation. Thirdly, by exploring the graph-specific fault monotonic property, we propose the *monotonicity-driven GraphFI* (*MD-GraphFI*) to granularly draw system severe error boundaries for predictable/unnecessary fault injection avoidance. Merging them all, GraphFI can reduce system fault site space by up to two orders of magnitude, which achieves 2.1∼15.2× speedup compared to SOTA methods while providing better reliability assessment accuracy.

I. Introduction

As a flexible data structure for representing complex relationships between objects, graphs are widely utilized in important real-world scenarios. Due to the inherent parallelism of graph processing, General-Purpose GPUs (i.e., GPGPUs) have established their prevalence in accelerating graph processing [1]–[4]. While delivering performance improvement, the increasing hardware fault rate (e.g., transient bit-flip faults [5], [6]) of highly integrated GPGPU platforms bring severe reliability challenges to graph processing. Even worse, the extensive graph data interaction enables abundant fault propagation, further exacerbating this challenge. The hardware imperfections may cause graph system to suffer from *obvious system crashes* or *silent but severe output corruptions*. As graph processing becomes pervasive in safety-critical real-world domains (e.g., financial fraud detection and electrical power system), the above severe fault-incurred consequences may lead to non-negligible system state recovery overhead [7] or financial loss/penalty [8], [9]. As a result, beyond pursuing extraordinary performance, it is also essential to guarantee their reliable execution.

To design an efficient and reliable graph processing paradigm, the first and vital step is to systematically analyze the reliability profile of the graph processing system. Fault Injection (FI) is the classic method which can identify the system's fault-sensitive area by randomly inserting errors during program runtime and aggregating the corrupted results. Recently, multiple FI-analysis tools have been developed by both industries (e.g., SASSIFI [5] and NvbitFI [10] proposed by NVIDIA) and academia [6], [11], [12]. However, abundant FI trials are necessary to obtain statistically significant analytical conclusions, which typically incur a non-negligible time overhead in GPGPU as each FI trial indicates a complete program execution.

To bridge the inefficiency of traditional FI, a limited set of studies aim to accelerate the FI-based methodology by exploring the program fault impact similarity/monotonicity property [13]–[18]. For example, Hari et al. [13] observe threads with similar dynamic instruction sequences (i.e., similar execution patterns) show similar error behavior, thus only conducting reliability assessment for representative threads. Observing more significant fault-incurred deviations typically produce more severe output corruptions, Li et al. [17] preferentially search for the minimum deviation that can cause severe corruptions, thus avoiding predictable FI trials. Although showing superiority for fast system reliability assessment, directly applying theses methods may lead to inaccurate graph reliability evaluation due to the specific data topology and execution paradigm. For instance, the graph topology may decouple the relationship between thread dynamic instruction sequence and their error behavior, and the graph-specific updating rules may also violate the error monotonic property.

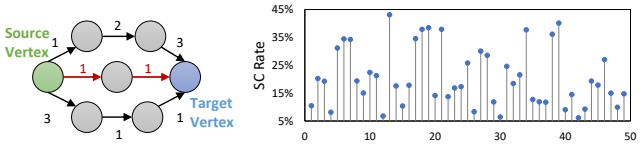
Considering the urgent demand for the reliability and performance co-optimization of graph processing systems, we build *GraphFI*, an efficient *Graph Fault Injection* method on GPGPUs. *The core idea and main novelty of our approach is that we progressively excavate the fault propagation and effect mechanisms hidden in the execution paradigm and data topology of graph processing, thereby guiding targeted and efficient FI trials.* Specifically, observing that iterations with similar active vertex set exhibit similar error behavior, we propose *iteration-driven GraphFI* (*ID-GraphFI*) to solely selects representative iterations for fast error resilience profiling. Secondly, by detecting resilience-similarity communities in graph topology, we propose *topology-driven GraphFI* (*TD-GraphFI*) that only select representative vertices for community overall reliability evaluation. Thirdly, by exploring the graph-specific fault monotonic property, we propose the *monotonicity-driven GraphFI* (*MD-GraphFI*) to draw error boundaries for predictable/unnecessary fault site injection avoidance. Merging them all, GraphFI can reduce the fault site space of graph programs for up to two orders of magnitude and achieve 2.1∼15.2× speedup compared to SOTA methods while providing better reliability assessment accuracy.

To the best of our knowledge, GraphFI is the first work that can perform fast reliability assessment for GPGPU-based graph processing. In summary, we make the following contributions:

- Explore the graph-specific error propagation and effect mechanisms and observe the (1) *fault similarity property* among iterations/vertices and the (2) *fault monotonicity property* in graph execution process, which provide opportunities for fault space reduction.
- Propose GraphFI that progressively reduce the fault space to efficiently and accurately evaluate the reliability profile of GPGPU-

This work is supported by the National Key Research and Development Program of China under Grant No. 2023YFB4502304, the National Natural Science Foundation of China (NSFC) under Grants No. 62272190, No. 62302190 and No. 62372207, and the Graduate Innovation Fund of Jilin University under Grant No. 2024CX091.

Corresponding authors: Hengshan Yue (yuehs@jlu.edu.cn) and Xiaohui Wei (weixh@jlu.edu.cn).



(a) (b)

Fig. 1. (a) An example of critical edges in Single Source Shortest Path (SSSP) and (b) The SC rate of 50 vertices which have similar neighbor numbers from Pagerank (PR).

bases graph processing.

- Evaluate the efficiency and accuracy of GraphFI on reliability profile estimating of four popular graph programs. We also demonstrate GraphFI is superior than other SOTA methods.
- Demonstrate how we can leverage GraphFI to guide design fault tolerance strategy GPGPU-based graph processing.

II. Background and Motivation

A. Graph Processing on GPGPUs

Graphs are typically represented in the form of $G = (V, E)$, where V represents the vertices set, and E denotes the edge set that represent the relationships between vertices. The purpose of graph processing is to explore complex vertices relationships by iteratively updating vertices states based on the adjacency relationships. In each iteration, vertices that receive new states in the last iteration are marked as active, which need to conduct the update operation. Due to the specific data topology, graph algorithms can be decomposed into a series of homogeneous vertex/edge tasks, exhibiting inherent compatibility to parallel computation.

Packed with thousands of computing units, GPGPUs have established their prevalence in accelerating parallel tasks. To fully utilizing the parallel potential of GPGPUs, *vertex-centric* [1], [19], [20] and *edge-centric* [21], [22] graph programming model become pervasive in recent years. In vertex-centric model, a thread serves as the computation carrier of individual vertex, including neighbor information aggregation, vertex state update, and state propagation. However, as the workload of each thread is proportional to its neighbor number, the vertex power-law distribution (i.e., a few vertices have numerous neighbors) in graph may lead to thread load imbalance in vertex-centric model. Then, edge-centric model is proposed to mitigate this imbalance. In this model, a thread operates on single edge to atomically update its destination vertex state using the source vertex state, resulting in similar workload among threads. However, compared to vertex-centric model, much more atomic operations are required as multiple threads may simultaneously write to the same vertex, which may become system performance bottleneck.

In above models, all vertices perform state updates and propagate synchronously in each iteration. To pursue a more efficient graph processing paradigm in GPGPU, recent studies [4], [23], [24] propose the asynchronous execution model that allows immediate vertex state propagation by relaxing the strictly synchronous barriers. For example, AsynGraph [4] considers the graph sketch constructed with high-degree vertices typically plays an important role in convergence speed, thus conducting multiple round state updating for the graph sketch in each iteration to accelerate graph convergence[1].

B. Reliability Challenge in Graph Processing

Although delivering performance improvement, the increasing hardware fault rate (e.g., transient bit-flip faults) of highly integrated GPU platforms bring severe challenges to graph processing. Even worse, the extensive graph data interaction enables abundant fault propagation, further exacerbating this challenge. Under fault perturbation, graph processing may suffer from serious behaviors, e.g., *system*

[1]In this paper, we mainly focus on fast reliability assessment under the vertex-centric model. We also discuss the adaptiveness of GraphFI to edge-centric and asynchronous models in Section VI.

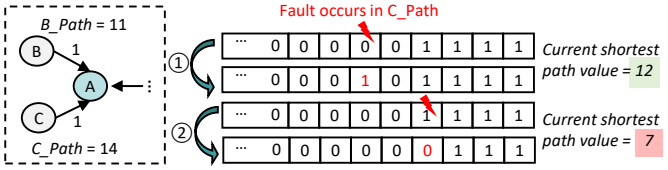


Fig. 2. An example of the un-monotonic fault impact to vertex state updating in Single Source Shortest Path (SSSP).

Detected symptoms (Detected) and *silent but Severe output Corruption (SC)*. Detected cases typically include segmentation faults, fatal traps, application aborts, etc. SC cases indicate graph application produces unacceptable output quality degradation without obvious system anomaly.

Typically, Checkpoint and Recovery (CR) mechanisms [7], [25] can be employed to restore system states when receiving crash signals. However, this may incur non-negligible overhead as heavy memory is required to store abundant immediate graph states. More seriously, CR becomes invalid for silent SCs because there are no obvious system symptoms. Therefore, modular redundancy-based mechanisms [26], [27] (e.g., DMR or TMR) are typically needed for system error detection and correction. However, such redundancy will seriously affect system parallelism, leading to considerable performance degradation. As graph-enabled relevant techniques become pervasive in real-time and safety-critical domains (e.g., financial fraud detection and electrical power systems), it is essential to guarantee reliable graph processing while maintaining computational efficiency. *To this end, the first and vital step is to systematically analyze the reliability bottleneck of graph processing, thereby providing helpful guidance for efficient fault-tolerance strategy design, which is the focus of this study.*

C. Existing Efforts on Program Reliability Analysis

Statistical Fault Injection (SFI) is a traditional approach for program reliability analysis. By randomly inserting *transient bit-flip errors*[2] during program runtime and aggregating the corrupted results, researchers can identify the system's fault-sensitive areas (i.e., reliability bottleneck). Recently, multiple FI-based reliability analysis tools have been developed by both industries (e.g., SASSIFI [5] and NvbitFI [10] proposed by NVIDIA) and academia [6], [11], [12]. However, due to the enormous fault space of GPGPU programs (up to billions [14]), traditional SFI needs abundant trials to achieve statistically significant analytical conclusions, which may incur unacceptable resource- and time-overhead.

Recent efforts attempt to improve the reliability analysis efficiency [13]–[18], [26]. The first category explores program areas that have similar behavior under faults impact, thus only performing FI trials on representative areas to avoid blind FIs. For example, Nie et al. [14] observe threads sharing similar dynamic instruction count often exhibit similar error behavior in GPGPU programs, thus accordingly cluster fault-similar thread groups. FI trials on only one thread are enough to understand the error behavior of the corresponding thread group. More rigorously, Anwer et al. [13] utilize dynamic instruction sequence as the criteria for generating fault-similar thread groups. Observing faults occurring in the operand definition and its first use will result in the same impact (i.e., def-use equivalence pruning), Hari and Pusz et al. [15], [18] spare operand-level FI experiments.

The second category utilizes the fault impact monotonicity property to guide efficient FI methodology [16], [17], [26]. The key insight behind the monotonicity is that larger fault-incurred deviations typically produce more severe output corruptions, and vice versa. For example, Chen et al. [16] notice the fault impact monotonicity in

[2]In this paper, we mainly focus on the transient bit-flip faults in line with previous studies [5], [10], [28].

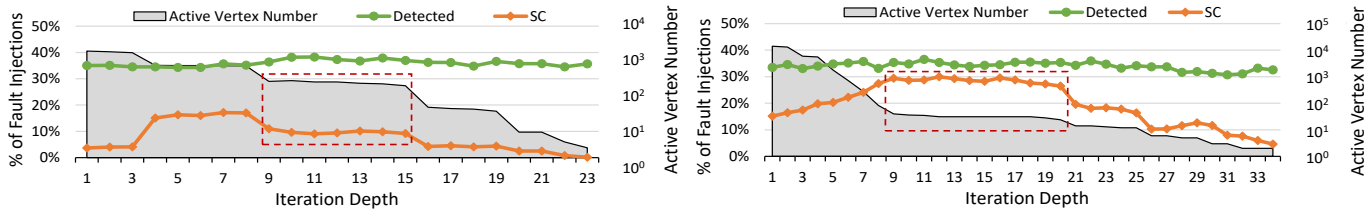


Fig. 3. The relationship between iteration error behavior and its active vertex number of (a) Strongly Connected Component (SCC) and (b) Pagerank (PR).

some AI applications, thus preferentially performing FIs on bits that may incur large deviations to efficiently identify fault-sensitive bits. Also leveraging this monotonicity, in some HPC applications (e.g., fast Fourier transform), Li et al. [17] perform progressively FIs to search the minimum deviation that can cause SCs, thus identifying the fault-tolerant/sensitive area.

D. Motivation

The above efforts show superiority in improving FI efficiency of traditional programs. However, due to the specific data topology and execution paradigm, directly applying these methods to graph applications may lead to inaccurate reliability evaluation. For example, previous studies [13], [14] consider threads with similar dynamic instruction sequences (DIS) to have similar error behavior in GPGPUs. However, this phenomenon does not always exist in GPGPU-based graph processing. For example, all threads have similar DIS in edge-centric model. However, some edges (i.e., threads) may play important roles in graph state updating, thus are more sensitive to faults. Fig. 1(a) illustrates an example of Single Source Shortest Path (SSSP) that uses *min()* to aggregate in-edge vertices' path and calculate the shortest path of each vertex to the source vertex. Obviously, faults occurring on the shortest path (marked with red) are more likely to cause SCs. In vertex-centric model, similar DIS only indicates that vertices have similar neighbor numbers. However, these vertices may have different significance in graph processing (e.g., the influencers of social networks), exhibiting different error sensitivity. Fig. 1(b) shows the SC rate of 50 vertices with similar neighbor numbers in PageRank. The significant fluctuation further reminds us to carefully explore the fault impact similarity existing in graph processing on GPGPUs.

Besides, considering larger fault-incurred deviation typically results in more severe output corruption, previous studies [16], [17] efficiently explore the error boundary that separates the error sensitive and error robust area. However, this monotonic property does not always hold in graph processing due to the specific graph updating rules. For example, as shown in Fig. 2, the current shortest path of A is 12 obtained from B. The fault in low order bit of C_Path will change A's shortest path from 12 to 7. While, fault in high order bit of C_Path will not lead to path corruption, even though the fault introduces larger data deviation. Thus, the specific updating rules of graph processing bring challenges for fault behavior exploration.

In conclusion, it is difficult to efficiently and accurately estimate the reliability of graph processing utilizing existing solutions. In this paper, we first explore the hidden reliability patterns within the specific data topology and execution paradigm of graph processing (Section III). Then we introduce the overall workflow of our proposed GraphFI (Section IV).

III. Error Resilience Characterization for GPGPU-Based Graph Processing

In this section, we progressively excavate the graph error behavior characteristics from the perspective of iterations, data topology, and graph execution to explore fault space reduction opportunities.

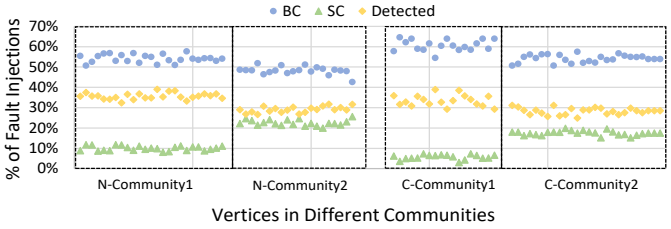


Fig. 4. Resilience similarity among vertices (1) with similar neighborhood (N-Community) and (2) in subgraph with high clustering coefficient (C-Community). BC indicates *Benign Corruption* in the output, which can be accepted by users.

A. Iteration Perspective

Graph processing iteratively updates vertex states until convergence. In an iteration, only active vertices need to conduct the update operation. With the dynamic evolving active vertex set, the error behavior may vary throughout the iterations. Thus, we first focus on the reliability regularity among iterations.

Fig. 3 shows the relationship between iteration error behavior and its active vertex number. We first observe that the Detected trend remains smooth with the changing active vertex number. Detected cases typically arise in state propagation operation (i.e., neighbor state loading). The ratio of loading operation generally remains stable although more active vertices may produce more loading operations. In contrast, the SC rate irregularly fluctuates in investigated benchmarks. Thus, additional effort is required for fast SC proneness estimation. Fortunately, we notice that iterations with similar active vertex sets demonstrate similar SC rates (e.g., iteration 9-15 and iteration 9-20 in the dashed boxes in Fig. 3(a) and Fig. 3(b)). Similar active vertex sets indicate similar data topologies that participate in computation, thus resulting in similar fault propagation and impact behavior. Finally, for iterations with obviously varying active vertex sets, there is no obvious criteria for fault similarity judgment. Thorough FI is suggested to ensure the accuracy of reliability analysis.

Heuristic#1*: We first suggest to construct iterations with similar neighbor loading operation ratio into a Detected-case similarity group. As the loading operation ratio generally stays stable, only individual iteration is enough for overall Detected rate estimation. Then, the similar SC rate of iterations with similar active motivates us to accordingly generate SC-case similarity groups, where one iteration can represent the SC proneness of the whole group.*

B. Topology Perspective

In graph processing, vertices conduct massive state interaction, which provide plentiful opportunities for fault propagation. Thus, we attempt to explore opportunities for efficient reliability analysis from data topology perspective.

A reasonable hypothesis is that vertices with similar neighbors share similar error manifestation, as they exhibit similar fault propagation behavior. Fig. 4 verifies this assumption by illustrating the fault result distribution of vertices with similar neighborhoods (N-Community). We use the Jaccard Coefficient as the proxy of vertex neighborhood similarity which is defined as:

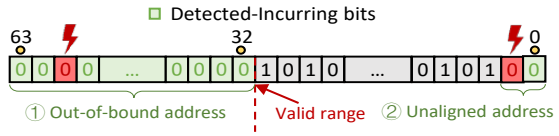


Fig. 5. An example of address operand.

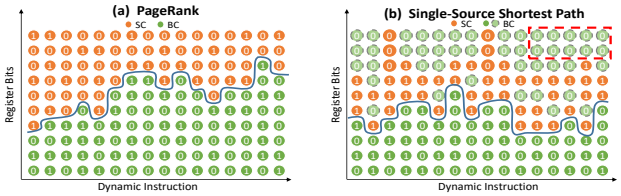


Fig. 6. The error behavior and the error boundaries in (a) PageRank (PR) and (b) Single Source Shortest Path (SSSP).

$$JC\left(v_i, v_j\right) = \frac{\left|N\left(v_i\right) \cap N\left(v_j\right)\right|}{\left|N\left(v_i\right) \cup N\left(v_j\right)\right|} \tag{1}$$

where v_i and v_j represent two vertices with the neighbor sets of $N\left(v_i\right)$ and $N\left(v_j\right)$. Beyond strictly similar neighbors, we observe that vertices in highly clustered groups also exhibit similar error behavior due to their frequent intra-group fault propagation. The C-Communities also illustrate the similar vertices fault distribution.

Heuristic#2*: Utilizing the correlation between vertices' error resilience and their neighborhood-similarity/clustering-property, we identify fault equivalent vertex communities for fast reliability assessment.*

C. Execution Perspective

In this section, we attempt to improve the Monotonicity-guided method by profiling the graph execution to help speed up graph error resilience analysis.

1) Detected Cases: As mentioned in Section III-A, almost all Detected cases originate from the neighbor state loading operation. Fig. 5 demonstrates an example of an address operator, which is typically 64-bits. Actually, we can directly foresee the Detected-incurring bits. Firstly, as the system will allocate a valid address range to the program, errors beyond a certain high order bit of the address will generate an out-of-bound memory. Beyond this, other Detected cases originate from the errors in the lower 2 bits, which will cause unaligned memory accesses.

Heuristic#3*: By checking the valid address range allocated to the program and identifying the Detected-incurring bits, we can directly estimate the Detected rate of the program without FI.*

2) SC Cases: In contrast to Detected cases, SC cases are typically caused by neighbor information aggregating and vertex state updating. Fig. 6(a) illustrates the FI results of Pagerank, where the x-axis represents the instruction sequence and the y-axis represents fault sites (i.e., different bits of operand). We observe larger fault-incurred deviation (i.e., higher bits) are more likely to result in SCs and there is generally a clear SC-BC boundary (blue line in Fig. 6(a)). This motivates us to preferentially search fault boundary to spare FI experiments.

However, we observe complicated error distribution of some graph applications. As shown in Fig. 6(b), several BC cases appear in operand higher bits (the dashed circles). We observe these "abnormal" BCs can be attributed to the specific updating rules of graph algorithm[3]. For example, SSSP uses *min()* to select the shortest path of all in-edge neighbors and accordingly update the target vertex path. Faults in higher bits of non-shortest path operands (e.g., the bits in the dashed box) may be eliminated by the *min()* function (e.g., the example in Fig. 2 in Section II-D). Apart from these special bits, the previously mentioned "boundary" is still observed in the remaining bits. Thus, excluding the abnormal bits, we can also perform fast reliability assessment by finding the fault boundary.

Heuristic#4*: Based on the specific graph updating rules, we can first determine the operand bits whose fault behaviors obey the monotonicity. Then a fast reliability assessment can be achieved by preferentially searching the fault boundary.*

[3]This phenomenon appears widely in many graph applications, such as the *min()/max()* in SSSP, SCC, WCC, MST, SSWP, etc.

IV. GraphFI Architecture

In this section, we first explain the high-level design of GraphFI, then present the details of the implementation steps, and finally show how to utilize GraphFI to help design fault tolerance strategies.

A. Design Overview

Guided by the heuristics proposed in Section III, GraphFI conducts progressive fault site reduction to improve FI efficiency. Fig. 7 depicts the workflow of GraphFI. GraphFI first performs program profiling to obtain (1) *Golden output* (i.e., the fault-free output), which can be utilized to judge whether FIs introduce SCs under the user-defined *Quality metrics*, (2) *Active vertex set of each iteration* and *Neighbor loading operation ratio*, (3) *Graph topology*, and (4) *Valid address profile and Updating operation* as the input of iteration-driven GraphFI (ID-GraphFI), topology-driven GraphFI (TD-GraphFI), and monotonicity-driven GraphFI (MD-GraphFI), respectively. In ID-GraphFI, we identify the fault similarity iteration groups and select one iteration in each group for fast reliability assessment. In each selected iteration, TD-GraphFI identifies fault equivalent vertex communities and also chooses one vertex from each community for community resilience estimation. In each selected vertex, MD-GraphFI draws error boundaries for predictable/unnecessary FI avoidance. Finally, GraphFI accumulates the FI results to obtain the resilience profile of different granularities of graph processing on GPGPUs.

B. Design Details

1) ID-GraphFI: Based on *Heuristic#1*, we first check the neighbor state loading operation ratio of different iterations. As this ratio typically remains stable, we randomly select individual iteration to represent the overall Detected rate of the program. Then we divide consecutive iterations with similar active vertex sets into an iteration group, and select a representative iteration from each group as FI candidate to efficiently evaluate the program SC rate.

2) TD-GraphFI: Within each selected iteration, based on *Heuristic#2*, GraphFI identifies vertex communities in which vertices (a) share similar neighbors (N-Communities) or (b) in highly clustered subgraphs (C-Communities). To identify N-Community, we first screen out vertex pairs with high Jaccard Coefficient (JC). Then a depth-first traverse is conducted towards these pairs to generate communities where vertices have pairwise similar neighbors. For C-Community, we calculate each vertex's Clustering Coefficient (CC). Vertices whose CC surpass the threshold are then gathered into distinct communities based on their connectivity. Then we select one vertex from each community as FI candidate. Vertices belonging to multiple communities are preferentially selected for maximizing FI efficiency improvement.

3) MD-GraphFI: We record the dynamic instruction instances (including the operand values) of each thread as the whole fault space. Within each selected vertex (i.e., thread), based on *Heuristic#3*, we first check the dynamic memory profile and accordingly determine the Detected-incurring bits [15]. Then we calculate the ratio of Detected-incurring bits, which is actually the Detected rate of the program. Secondly, based on *Heuristic#4*, we first exclude the "abnormal" bits whose error-incurred outcome is surely BC according to the fault masking property of the graph updating operation (if there is). Then, a

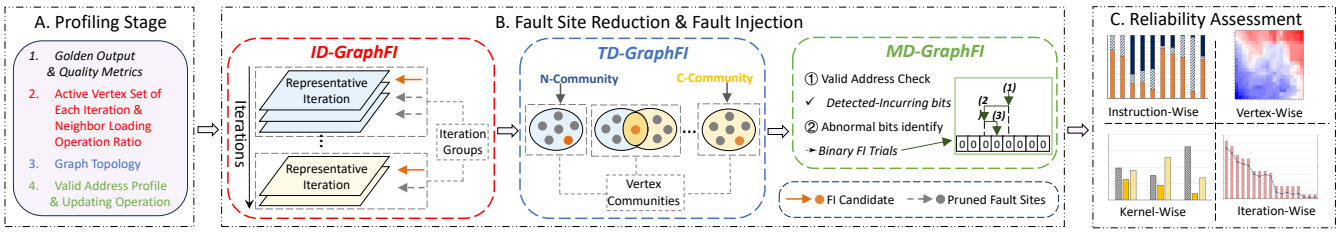


Fig. 7. Overall workflow of GraphFI.

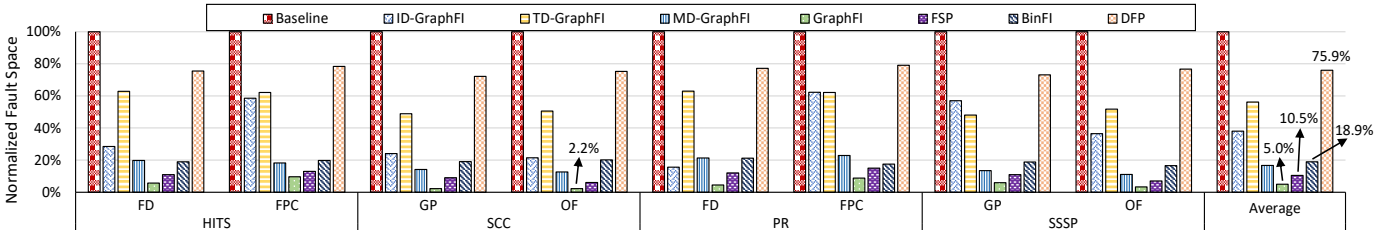


Fig. 8. The speedup of GraphFI and SOTA methods compared to baseline FI, where *ID-GraphFI*, *TD-GraphFI*, and *MD-GraphFI* represent only corresponding mechanism is adopted.

binary search FI is conducted to efficiently identify the fault boundary thus obtaining the instruction level SC rate. Specifically, if the FI result of a bit is SC, then the next FI trial moves to the lower bits until a certain bit is found, which separates the SC and BC cases.

4) Error Resilience Estimation and Guidance for Fault Tolerance Design: After conducting progressive fault site pruning and binary FI experiment, we obtain the instruction-wise error behavior. On this basis, we can accumulate the instruction-level FI results to obtain the error resilience of other system granularity (e.g., thread, and iteration level). For example, the SC rate of individual thread (i.e., vertex) in an iteration can be calculated by:

$$R_{SC} = \sum_{i \subset I_V} A(i) \tag{2}$$

where $A(i)$ is the SC rate of individual instruction and I_V is the instruction set in the target thread. Similarly, iteration-wise SC rate is the accumulation of its thread set.

Next we introduce how to leverage the analysis results of GraphFI to help design efficient fault tolerance strategies for GPGPU-based graph processing. Among the three outcome categories of graph processing (i.e., BC, SC, and Detected), SC is typically the most challenging to detect due to its silent property. Selective thread duplication (ThdDup) and instruction duplication (InsDup) are classical mechanisms to detect silent SCs in GPGPUs by double executing SC-Prone threads/instructions and comparing their results at runtime. We can use GraphFI to efficiently obtain instruction- and thread-level SC rates. Then, GraphFI preferentially duplicates threads/instructions with high SC proneness to pursue efficient SC detection capability.

V. Evaluation

We evaluate the proposed GraphFI by performing the following research questions (RQs):

- RQ_1 What is the performance of GraphFI in evaluating reliability of GPGPU-based graph processing?
- RQ_2 What is the accuracy of GraphFI in evaluating reliability of GPGPU-based graph processing?
- RQ_3 What is the performance of GraphFI-guided fault tolerance mechanism?

A. Experimental Setup

GraphFI settings. We deploy GraphFI on a host Intel i7-7700 CPU (2.60 GHz) with 16 GB memory and an NVIDIA GeForce RTX 3090 GPU. We use a cycle-accurate, open-source simulator GPGPU-sim [29] to obtain the dynamic instruction instances of each thread

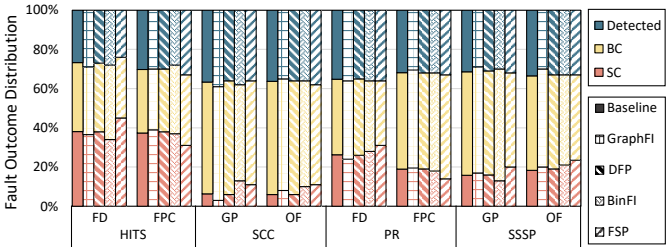


Fig. 9. Error resilience profile comparison of GraphFI and SOTA methods against the baseline FI.

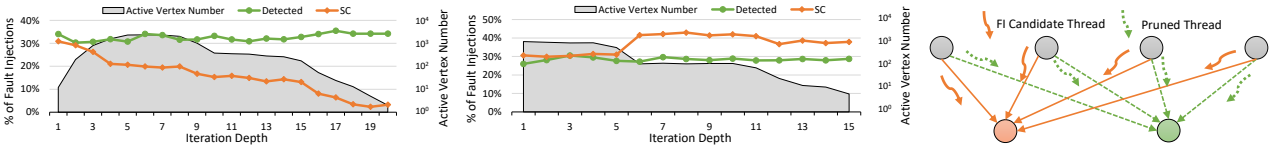
(i.e., the whole fault space). We perform FI trials following the basic FI method in [10], which injects single bit-flip errors into general-purpose registers of dynamic instructions.

Dataset, benchmarks, and output quality metrics. Four popular graph algorithms are considered in our experiment, including *Single Source Shortest Path* (SSSP), *Strongly Connected Component* (SCC), *Pagerank* (PR), and *Hypertext Induced Topic Search* (HITS). We perform evaluations on four real-world graphs (i.e., FD [30], GP [31], FBP [31], and OF [31]). Incorporating the domain-specific knowledge [32], [33] and the reference to previous error resilience studies [34], [35], we adopt *Top-k ranking* for PR and HITS as users typically only care about the correctness of the top-ranked items. For SSSP and SCC, we use *Relative L2-norm* to estimate the deviation of the overall graph's paths/connectivity from the ground truth.

Comparison methods and baseline. We select three SOTA reliability estimation methods for comparison, i.e., (1) FSP [14], (2) DFP [18], and (3) BinFI [16]. FSP clusters thread equivalent groups using dynamic instruction count as criteria and select representative threads for FI. DFP considers instruction semantics and tracks the information flow of each bit to construct fault equivalent bit sets for fault space reduction. Guided by the monotonic property, BinFI binary searches 0 bit and 1 bit separately of each operand to find the SC boundary. All these methods and GraphFI take FI to the whole fault site space as the baseline for comparison. As the faults occurring in the operand definition and its first use will result in same impact (as illustrated in Section II-C), the def-use pruning method has become a popular approach to reduce the fault space. In the following evaluation, we perform a preliminary fault space reduction using the def-use pruning for GraphFI and SOTA solutions.

B. Speedup of GraphFI (RQ_1)

Fig. 8 shows the FI efficiency of all methods by depicting the normalized reduced fault space compared to the baseline. ID-



(a) (b) (c)

Fig. 10. The iteration error behavior of (a) SSSP in edge-centric model and (b) HITS in AsynGraph model. (c) is an example of representative threads selection for edge-centric model.

TABLE I
THE ERROR DETECTION CAPABILITY (I.E., ERROR COVERAGE) OF GRAPHFI- AND FSP-GUIDED INSTRUCTION DUPLICATION (INSDUP) AND THREAD DUPLICATION (THDDUP) METHOD.

Overhead		25%	50%	75%
InsDup Detection	GraphFI	70.3%±2.8%	90.1%±1.2%	95.6%±0.9%
	FSP	65.5%±3.3%	82.9%±2.1%	93.5%±1.2%
ThdDup Detection	GraphFI	80.2%±4.5%	89.9%±2.9%	96.6%±1.8%
	FSP	57.1%±7.2%	77.1%±5.4%	90.6%±2.2%

GraphFI, TD-GraphFI, and MD-GraphFI achieve an average fault space reduction of 62.0%, 43.8%, and 83.3%, respectively. Merging them all, GraphFI can compress the fault space to 5.0% of its original size on average, achieving a speedup of up to two magnitudes to baseline FI. In general, GraphFI exhibits 2.1∼15.2× speedup to other SOTA methods. For FSP, as vertices' degrees exhibit significant variation (i.e., more thread groups), FSP has to select abundant threads to perform FIs. The efficiency of DFP is mainly proportional to the ratio of logical/data-movement instructions (e.g., MOV, SHL, AND, etc.), while these instructions usually occupy a small portion of the program. To ensure accurate reliability evaluation, BinFI needs to respectively perform binary search FI for 0 bits and 1 bits of each operand, which could incur non-negligible overhead (e.g., up to 8 FIs are necessary for a 32-bit operand).

C. Overall Error Resilience Evaluation (RQ_2)

Fig. 9 shows the reliability profile obtained by GraphFI and other SOTA methods. On average, the difference of GraphFI compared to baseline FI in terms of Detected, SC, and BC rates are 0.96%, 1.36%, and 1.22%, respectively. The delicate deviations indicate that GraphFI achieves comparable accuracy with the ground truth when evaluating the error profile of graph processing. Compared to GraphFI, the results of FSP exhibit more obvious discrepancy and volatility to the ground truth. Because the graph data topology decouples the relationship between threads' dynamic instruction count and their error resilience. DFP and BinFI also achieve promising accuracy in estimating error resilience as they adopt relatively conservative principles to reduce fault space. However, this leads to unsatisfactory efficiency due to the lack of consideration of graph-specific characteristics. In general, GraphFI achieves a better tradeoff between resilience evaluation efficiency and accuracy than SOTA solutions.

D. GraphFI-Guided Fault Tolerance (RQ_3)

Table. I shows the SC detection rate at different protection overhead, which means the increasing dynamic instruction or thread ratio for InsDup and ThdDup. In general, we observe that GraphFI-guided duplication exhibits better SC mitigation efficiency and stability compared to FSP, especially for ThdDup. This further verifies the accuracy of GraphFI in terms of graph error resilience estimating in different system granularities.

VI. DISCUSSION: THE EFFECTIVENESS OF GRAPHFI TO OTHER GRAPH PROCESSING PARADIGM

In addition to the synchronous vertex-centric model (VC), we further discuss the adaptiveness of GraphFI to synchronous edge-centric (EC) and asynchronous graph processing models (taking AsynGraph [4] as an example) illustrated in Section II-A.

Iteration perspective. In VC, our key insight is that iterations with similar active vertex sets show similar error behavior. Such a phenomenon is observed in other models. Fig. 10(a) shows the error behavior of different iterations in EC model. Here, similar active vertex sets imply similar thread number and graph topology that participate in computation, also exhibiting similar fault propagation patterns and error behaviors. AsynGraph considers the graph sketch constructed with high-degree vertices play an important role in convergence speed, thus conducting multiple round state updating for this graph sketch in each iteration. In fact, we observe these high-degree vertices (less than 20% of total vertices) contribute over 80% SCs of the program, thus analysis for these vertices can generally understand the program resilience. Fig. 10(b) illustrates the error behavior of the high-degree graph sketch with iterations going on. We observe that high-degree vertices gradually converge with iterations, meanwhile also exhibiting phase-wise similar error resilience.

Vertex perspective. By analyzing the fault propagation and impact, we find the resilience similarity in vertices within highly clustered subgraphs or having similar neighbor. Such similarity also exist in other models because these models generally do not change the fault propagation behavior in data topology. Note that, we should pay more attention to FI candidate thread selection. For example, as shown in Fig. 10(c), a vertex updating task is divide into four threads in EC model. As the orange vertex and the green vertex have the same neighborhood, we can only select all four orange threads as FI candidate.

Execution perspective. Diverse graph processing models on GPUs essentially modify the granularity of task division (e.g., VC and EC models) or the way state updates are propagated (e.g., synchronous and asynchronous models), rather than changing the graph update algorithm. Our approach focuses on exploring fault space reduction opportunities based on graph algorithms and the impact of faults, which remain unaffected with different graph processing models.

In conclusion, the proposed heuristics for fault space reduction are generally appropriate for the popular graph processing models.

VII. CONCLUSION

In this paper, we propose GraphFI, an FI framework that can efficiently and accurately estimate the reliability profile of GPGPU-based graph processing. The main effort is that we reveal the error propagation and effect mechanisms hidden in the execution paradigm and data topology of graph processing on GPGPUs. Specifically, we observe (1) iteration-wise error behavior similarity exists in iterations sharing similar active vertex set, (2) vertex-wise error behavior similarity among vertices with similar neighbors or in highly clustered subgraphs, and (3) instruction-wise graph algorithm-specific fault impact monotonicity property. Incorporating these characteristics, GraphFI progressively prunes the fault space and performs targeted and efficient FI trials. Experimental results show that, in estimating the reliability of GPGPU-based graph processing, GraphFI can achieve a 2.1∼15.2× speedup compared to SOTA methods, while providing more accurate evaluation results.

REFERENCES

[1] A. H. N. Sabet, Z. Zhao, and R. Gupta, "Subway: Minimizing data transfer during out-of-gpu-memory graph processing," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[2] Y. Zhang, D. Peng, X. Liao, H. Jin, H. Liu, L. Gu, and B. He, "Large-graph: An efficient dependency-aware gpu-accelerated large-scale graph processing," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 18, no. 4, pp. 1–24, 2021.

[3] Z. Fu, M. Personick, and B. Thompson, "Mapgraph: A high level api for fast development of high performance graph analytics on gpus," in *Proceedings of workshop on GRAph data management experiences and systems*, 2014, pp. 1–6.

[4] Y. Zhang, X. Liao, L. Gu, H. Jin, K. Hu, H. Liu, and B. He, "Asyngraph: Maximizing data parallelism for efficient iterative graph processing on gpus," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 17, no. 4, pp. 1–21, 2020.

[5] S. K. S. Hari, T. Tsai, M. Stephenson, S. W. Keckler, and J. Emer, "Sassifi: An architecture-level fault injection tool for gpu application resilience evaluation," in *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2017, pp. 249–258.

[6] D. Sartzetakis, G. Papadimitriou, and D. Gizopoulos, "gpufi-4: A microarchitecture-level framework for assessing the cross-layer resilience of nvidia gpus," in *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2022, pp. 35–45.

[7] B. Nicolae, A. Moody, E. Gonsiorowski, K. Mohror, and F. Cappello, "Veloc: Towards high performance adaptive asynchronous checkpointing at large scale," in *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2019, pp. 911–920.

[8] P. H. Hochschild, P. Turner, J. C. Mogul, R. Govindaraju, P. Ranganathan, D. E. Culler, and A. Vahdat, "Cores that don't count," in *Proceedings of the Workshop on Hot Topics in Operating Systems*, ser. HotOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 9–16. [Online]. Available: https://doi.org/10.1145/3458336.3465297

[9] Q. Lu, G. Li, K. Pattabiraman, M. S. Gupta, and J. A. Rivers, "Configurable detection of sdc-causing errors in programs," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 3, pp. 1–25, 2017.

[10] T. Tsai, S. K. S. Hari, M. Sullivan, O. Villa, and S. W. Keckler, "Nvbitfi: Dynamic fault injection for gpus," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2021, pp. 284–291.

[11] Q. Lu, M. Farahani, J. Wei, A. Thomas, and K. Pattabiraman, "Llfi: An intermediate code-level fault injection tool for hardware faults," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 11–16.

[12] G. Li, K. Pattabiraman, and N. DeBardeleben, "Tensorfi: A configurable fault injector for tensorflow applications," in *2018 IEEE International symposium on software reliability engineering workshops (ISSREW)*. IEEE, 2018, pp. 313–320.

[13] A. R. Anwer, G. Li, K. Pattabiraman, M. Sullivan, T. Tsai, and S. K. S. Hari, "Gpu-trident: Efficient modeling of error propagation in gpu programs," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2020, pp. 1–15.

[14] B. Nie, L. Yang, A. Jog, and E. Smirni, "Fault site pruning for practical reliability analysis of gpgpu applications," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 749–761.

[15] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1, pp. 123–134, 2012.

[16] Z. Chen, G. Li, K. Pattabiraman, and N. DeBardeleben, "Binfi: An efficient fault injector for safety-critical machine learning systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356177

[17] Z. Li, H. Menon, K. Mohror, P.-T. Bremer, Y. Livant, and V. Pascucci, "Understanding a program's resiliency through error propagation," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 362–373.

[18] O. Pusz, C. Dietrich, and D. Lohmann, "Data-flow-sensitive fault-space pruning for the injection of transient hardware faults," in *Proceedings of the 22nd ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, 2021, pp. 97–109.

[19] F. Khorasani, K. Vora, R. Gupta, and L. N. Bhuyan, "Cusha: vertex-centric graph processing on gpus," in *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, 2014, pp. 239–252.

[20] A. H. Nodehi Sabet, J. Qiu, and Z. Zhao, "Tigr: Transforming irregular graphs for gpu-friendly graph processing," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 622–636, 2018.

[21] A. Roy, I. Mihailovic, and W. Zwaenepoel, "X-stream: Edge-centric graph processing using streaming partitions," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013, pp. 472–488.

[22] S. Zhou, R. Kannan, V. K. Prasanna, G. Seetharaman, and Q. Wu, "Hitgraph: High-throughput graph processing framework on fpga," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 10, pp. 2249–2264, 2019.

[23] W. Han, D. Mawhirter, B. Wu, and M. Buland, "Graphie: Large-scale asynchronous graph traversals on just a gpu," in *2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT)*. IEEE, 2017, pp. 233–245.

[24] T. Ben-Nun, M. Sutton, S. Pai, and K. Pingali, "Groute: Asynchronous multi-gpu programming model with applications to large-scale graph processing," *ACM Transactions on Parallel Computing (TOPC)*, vol. 7, no. 3, pp. 1–27, 2020.

[25] B. Fang, Q. Guan, N. Debardeleben, K. Pattabiraman, and M. Ripeanu, "Letgo: A lightweight continuous framework for hpc applications under failures," in *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, 2017, pp. 117–130.

[26] X. Wei, N. Jiang, H. Yue, X. Wang, J. Zhao, G. Li, and M. Qiu, "Approxdup: Developing an approximate instruction duplication mechanism for efficient sdc detection in gpgpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 4, pp. 1051–1064, 2024.

[27] A. Mahmoud, S. K. S. Hari, M. B. Sullivan, T. Tsai, and S. W. Keckler, "Optimizing software-directed instruction replication for gpu error detection," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2018, pp. 842–854.

[28] H. Yue, X. Wei, G. Li, J. Zhao, N. Jiang, and J. Tan, "G-sepm: Building an accurate and efficient soft error prediction model for gpgpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3458817.3476170

[29] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing cuda workloads using a detailed gpu simulator," in *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 2009, pp. 163–174.

[30] "Foldoc," http://vlado.fmf.uni-lj.si/pub/networks/data/dic/foldoc, 2024.

[31] "Grid plant, fb pages, and open flights," https://networkrepository.com/, 2024.

[32] L. Song, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "Graphr: Accelerating graph processing using reram," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 531–543.

[33] K. Vora, C. Tian, R. Gupta, and Z. Hu, "Coral: Confined recovery in distributed asynchronous graph processing," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 223–236. [Online]. Available: https://doi.org/10.1145/3037697.3037747

[34] R. Venkatagiri, A. Mahmoud, S. K. S. Hari, and S. V. Adve, "Approxilyzer: Towards a systematic framework for instruction-level approximate computing and its application to hardware resiliency," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–14.

[35] B. Nie, A. Jog, and E. Smirni, "Characterizing accuracy-aware resilience of gpgpu applications," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 111–120.

YOCO: A Hybrid In-Memory Computing Architecture with 8-bit Sub-PetaOps/W In-Situ Multiply Arithmetic for Large-Scale AI

Zihao Xuan[1], Yuxuan Yang[2], Wei Xuan[1], Zijia Su[2], Song Chen[2], Yi Kang[2*]
[1]The Hong Kong University of Science and Technology, Hong Kong SAR, China
[2]School of Microelectronics, University of Science and Technology of China, Hefei, China
*Corresponding author: ykang@ustc.edu.cn

***Abstract*—In this paper, we further explore the potential of analog in-memory computing (AiMC) and introduce an innovative artificial intelligence (AI) accelerator architecture named YOCO, featuring three key proposals: (1) YOCO proposes a novel 8-bit in-situ multiply arithmetic (IMA) achieving 123.8 TOPS/W energy-efficiency and 34.9 TOPS throughput through efficient charge-domain computation and time-domain accumulation mechanism. (2) YOCO employs a hybrid ReRAM-SRAM memory structure to balance computational efficiency and storage density. (3) YOCO tailors an IMC-friendly attention computing flow with an efficient pipeline to accelerate the inference of transformer-based AI models. Compared to three SOTA baselines, YOCO on average improves energy efficiency by up to 3.9×~19.9× and throughput by up to 6.8×~33.6× across 10 CNN/transformer models.**

Index Terms—in-memory computing, analog computing, hybrid SRAM-ReRAM memory, DNN accelerator.

I. Introduction

In recent years, artificial intelligence (AI) algorithms, exemplified by convolutional neural networks (CNNs) and large language models (LLMs), have exhibited tremendous potential to revolutionize natural language processing (NLP) and computer vision tasks [1]. Driven by the scaling laws of AI, the demand for powerful computing hardware has increased dramatically to ensure the efficient execution of these algorithms [2]. For now, various high-performance AI accelerators, such as GPUs and TPUs, have been developed to meet this demand. However, these devices, based on von Neumann architecture, face significant energy efficiency bottlenecks due to "transistor scaling" limitation and the "memory wall" problem, particularly in the context of large-scale AI.

In-memory computing (IMC) technology has emerged as a promising solution by merging memory and computation units, thereby effectively reducing the cost of data movement and maximizing both bandwidth and energy-efficiency of computing [3]. This approach shows great potential for addressing the inefficiencies of traditional accelerators, especially for memory-bound tasks. Nevertheless, applying IMC to accelerate the inference of large-scale AI models presents several specific challenges, particularly given the diverse requirements of different models.

One primary challenge arises from costly analog-to-digital converters and digital-to-analog converters (ADCs/DACs). IMC architectures typically adopt in-situ arithmetic, where multiply-accumulate (MAC) operations are performed in analog form. This process requires DACs to generate analog inputs and ADCs to produce digital outputs. However, these ADCs and DACs are energy-intensive, consuming up to 85% of power in architectures like ISAAC [4], limiting system efficiency.

Additionally, most IMCs rely on a single type of memory, which restricts their flexibility to handle both static and dynamic matrix computations commonly found in transformer models [5]. For example, SRAM-based IMCs provide high computational speed but are constrained by limited memory size, making them inadequate for storing large-scale AI models. In contrast, ReRAM-based IMCs offer high-density storage but suffer from low-endurance and energy-intensive write operations, which hampers their ability to efficiently execute dynamic matrix computations.

Moreover, neural network algorithms often include complex computational patterns, such as *softmax* and *normalization*, which further constrain IMC performance. It is essential to account for these patterns when designing efficient computing dataflows to fully unlock the potential of IMC.

To address these challenges, this paper highlights the opportunities in analog computing, hybrid memory design, and pipelined computing to achieve energy-efficient acceleration of large-scale AI inference. The main contributions of this paper are as follows:

- We propose a novel in-charge computing mechanism to push *input data conversion, bit-wise multiplication, parallel accumulation, and weighted summation* all-in analog domain for energy-saving multi-bit MAC with low precision loss.
- We introduce an in time-domain accumulating approach to increase the size of the analog computing array with *high signal margin*, further reducing the energy consumption of both ADCs and partial sum (*Psum*) movements.
- We employ a hybrid ReRAM-SRAM memory design to trade-off computational flexibility and storage density. This design supports both *dynamic* and *static* matrix multiplication. Additionally, we design an IMC-friendly attention computing pipeline to accelerate transformer models.
- We develop a new AiMC architecture, named YOCO, with a hierarchical structure spanning *Array, IMA, Tile, and Chip* levels. YOCO integrates the aforementioned key features to enhance computing flexibility, efficiency, and performance.

Finally, we conducted a comprehensive evaluation of the YOCO architecture. At the circuit level, YOCO's in-situ multiply arithmetic (IMA) achieves an average energy efficiency of 123.8 TOPS/W, a throughput of 34.9 TOPS, and a computation error of less than 0.98%, for 8-bit 1024×256 vector-matrix multiplications (VMM). At the architecture level, YOCO outperforms three baseline models across 10 benchmarks, improving energy efficiency by 3.9× to 19.9× and throughput by 6.8× to 33.6×.

II. Background And Motivation

A. DNNs and Transformer Models

Deep Neural Networks (DNNs) are a class of advanced AI algorithms characterized by their deep layered architectures, which consist of multiple artificial neurons that enable the capture of intricate patterns in data. As DNNs have evolved, various types of network layers have emerged, including fully connected layers, convolutional layers, attention layers, and more. These layers share a fundamental requirement: matrix computation is the core operation. A prominent example of this is the widely used attention computing in transformer (see Fig. 1(a)), which can be represented by the following equation [5]:

$$\text{Attention}(Q,K,V) = \text{Softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

Certainly, attention computation is resource-intensive, requiring high memory bandwidth and numerous matrix multiplications (over 90%), especially in large-language models (LLMs) like GPT and BERT, which may contain billions of parameters.

Data reuse is crucial for optimizing attention computation. Effective data reuse minimizes memory access and computation redundancy, thereby improving both speed and energy efficiency. Techniques such as caching frequently accessed matrices, reusing KV matrices across multiple attention heads, and optimizing matrix multiplication dataflow are commonly employed. In IMC, data reuse is important to overcome limited memory bandwidth, particularly in handling the high volume of intermediate data generated during attention calculations.

B. Analog Processing-in-Memory Basic

The scaling law of DNN necessitates the urgent development of hardware acceleration systems with high computational efficiency to address the constraints of limited energy resources. Leveraging the inherent tolerance of DNNs to computational noise, there is increasing emphasis on utilizing energy-efficient AiMC to enhance matrix multiplication performance, particularly for DNN inference. Fig. 1(b) demonstrates the basic principle of AiMC for QK^T computing. The transpose of K matrix is pre-stored in the memory array, the Q vector ($q_0, q_1, \ldots, q_n$) is encoded into analog signals by DACs and propagated horizontally along the word lines of array, parallel analog MACs are performed within memory and compute cells (MCCs), and the vertically accumulated results are summed and converted into digital output by ADCs. The dual-function MCC, combining both memory and compute, eliminates weight movement during MAC operations and facilitates AiMC achieve maximum spatial parallelism and minimal time waste.

C. ADCs/DACs Cost Analysis

In a classic AiMC architecture, ADCs and DACs dominate energy consumption, limiting the advantages of AiMC (see Fig. 1(c)). According to REALLA's analysis [6], the high overhead of ADCs/DACs is primarily due to computation performed in bit-slice and block-wise modes. In bit slicing, inputs are processed over multiple cycles, weights across multiple parallel columns of array, and fully multi-bit MACs are realized digitally. The number

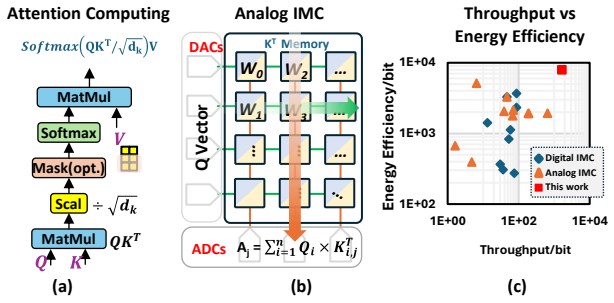


Fig. 1 (a) Scaled Dot-Product Attention. (b) AiMC topology for QK^T. (c) Normalized throughput versus energy efficiency of recent IMCs.

TABLE I. ADCs/DACs Cost Comparison

Architecture	Slice Weight	Slice Input	Block Size	ADC Cost	DAC Cost	Memory Type	Accuracy Loss
ISAAC [4]	Yes	Yes	Small	High	Low	ReRAM	High
RAELLA [6]	Yes	Yes	Mid	High	Low	ReRAM	Low
TIMELY [7]	Yes	No	Large	Low	Low	ReRAM	High
C-Ladder [8]	Yes	No	Small	High	High	DRAM	Low
C-2C [9]	No	No	Small	Low	High	SRAM	Low
Our (YOCO)	No	No	Large	Low	Low	Hybrid	Low

of ADC conversions per MAC (converts/MAC) is proportional to the product of the input and weight slices. Some works suggest that increased bit-slice enables the use of lower-resolution data converters. However, more input slices reduce DAC overhead but significantly increase ADC costs, and additional weight slices do not guarantee lower-resolution ADC. Shifting and adding in the digital domain amplify analog computation and quantization errors. In block-wise computing, weight matrix is divided into smaller blocks to match the size of analog computing array. The converts/MAC are also related to the number of blocks. Clearly, more bit slices and computing blocks mean more converts/MAC and results in more energy consumption.

D. Motivation

Some prior works have explored various bit-slice and computing block size to alleviate the cost of ADCs/DACs. Table I highlights the performance gap in recent designs [4, 6-9]. For now, modern AiMC has not yet reached its full potential. As DNNs evolve into the era of LLM, the dual demands for storage density and computational efficiency have intensified, exploring heterogeneous storage and pipeline computing in IMC is beneficial for the computation of large-scale models.

III. YOCO: You Only Charge Once

We first discuss the principle of the proposed in-charge computing (see sections III-A), followed by design of the time-domain accumulator (see section III-B). Next, we present the overall YOCO architecture in sections III-C, and finally, we introduce our pipeline design in Section III-D.

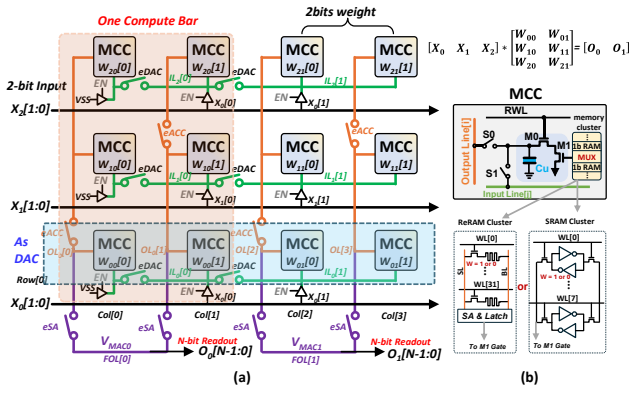


Fig. 2. (a) An example of AiMC array design for 3×2 VMM with 2bits input and weights. (b) MCC with unit capacitor and memory cluster.

A. Principle of Proposed in-Charge Computing

In a traditional charge-domain IMC circuit, each memory and computing cell (MCC) is equipped with a unit capacitor (see in Fig. 2(b)) solely for parallel accumulation. Therefore, we repurpose these unit capacitors to perform various functions through multiple charge sharing (MCS), including input data conversion, bitwise multiplication, parallel accumulation, and weighted summation. This enables fully analog multi-bit computation without input and weight slices. Fig. 2(a) shows a simple schematic of the proposed in-charge computing array, featuring 3×4 MCCs to support full parallel 3×2 VMM operations with 2-bit inputs and weights. In our simplified array, low-cost switches such as *eDAC*, *eACC*, and *eSA* are employed to reorganize the MCCs. In each row, MCCs are grouped in a 1:1:2 ratio through two eDAC switches, allowing the unit capacitors in the same row to form a 2-bit DAC for parallel 2-bit input data conversion. In the column direction, each MCC can perform only 1-bit weighted multiplication at a time. To facilitate 2-bit weight computations, two columns of MCCs are organized as a "compute bar" (CB). In each CB, *eACC* switches group the MCCs in each column, according to a unique configuration: the MCCs in the first column are grouped in a 1:2 ratio, while those in the second column are grouped in a 2:1 ratio. As a result, in a CB, only one MCC from the first column is connected to the final output line (FOL) via *eSA*, while two MCCs from the second column are connected. This establishes a 1:2 ratio in the number of unit capacitors between two columns involved in the final charge sharing, thereby inherently achieving in-situ shift and add for weighted summation.

Through additional switches, unit capacitors within MCCs can be dynamically reconfigured and grouped to perform different tasks by directing charge-sharing along specified paths, color-coded in the diagram to indicate stages of operation—green for DAC, orange for analog accumulation, and purple for weighted summation. Fig. 3 provides examples of specific switch configurations for each function, highlighting the array's flexibility for various analog computation needs. It includes four phases: **(1) DAC-Less Input Conversion by Row Charge Sharing:** When the *eDAC* switches are open and the tri-state gate is enabled (*EN=1*), only the first group's input is fixed at V_{SS}, while the capacitors inside the other groups charge or discharge according to

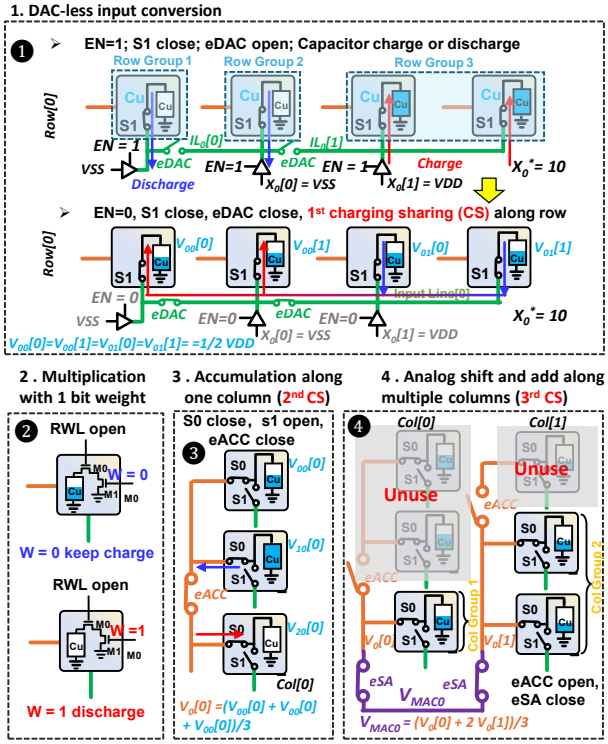


Fig. 3. Four times charge-sharing for in-situ multi-bit computing.

the input binary code $X_i[n]$, here i is row index and n is bit index. After this step, the tri-state gate is disabled (*EN=0*), and *eDAC* switches are closed, all capacitors in each row to charge sharing until reach equal voltage. If we set X_0 [1:0] is "10", the input conversion voltage is 1/2 V_{DD} (see ❶ in Fig. 3). The row capacitors are naturally organized as DAC via *eDAC* switches, and each row of the array requires only additional N tri-state gates and switches, resulting in negligible area and power consumption. **(2) Multiplication with 1-bit Weight.** In an MCC, analog multiplication is performed by two transistors *M0* and *M1*, whose gates are controlled by read word line (RWL) and memory cell, respectively. When RWL is activated, each MCC individual determines whether to release the charge of the capacitor based on 1-bit weight ("1" means keep and "0" means released). The final voltage within capacitor represents the result of multiplying between N-bit input and 1-bit weight (see ❷ in Fig. 3). **(3) Parallel Accumulation by Column Charge Sharing.** After the multiplication is completed, the S0 and *eACC* switches are closed (see ❸ in Fig. 3, allowing the capacitors of the MCCs in the same column to share charges, enabling full parallel accumulation. **(4) Weighted Summation by Multi-Column Charge Sharing.** After the completion of the third phase, each column's output represents an analog MAC result based on 1-bit weight. To achieve multi-bit weight MAC, as shown in ❹ of Fig. 3, we control the *eACC* switches to open and the *eSA* to close, so that only a unit capacitor in the first column participates in charge sharing, while two participate in the second column do so. The final shared voltage represents the bit-shifted sum of the two-column voltages, averaged accordingly.

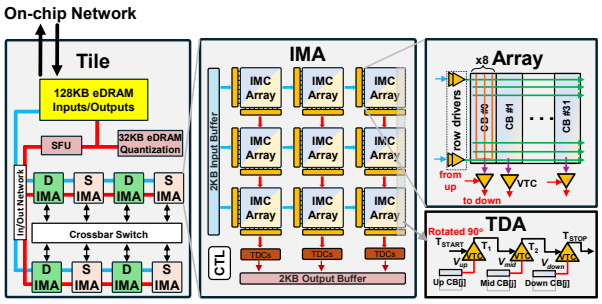


Fig. 4. Hierarchical YOCO architecture with in-charge computing array, time-domain IMA, and hybrid-memory based tile.

Overall, our design showcases a highly flexible and power-efficient approach for analog computation, enabling multiple functions to be performed sequentially by configuring the charge transfer pathways within each MCC, and the fully multi-bit computing process only requires charging once.

B. Time-Domain Accumulation

In our array, each CB generates an MAC voltage, which represents an analog partial sum. We can utilize time-domain accumulators (TDAs) to vertically accumulate these partial sums. A TDA (see the right bottom in Fig. 4) consists of serial head-to-tail voltage-to-time converters (VTCs), where the voltages from multiple CBs are converted into time signal and accumulated in the time-domain with high signal margin. When the previous VTC completes a signal transition, it releases a pulse signal to trigger the next VTC. The start signal is input from the top VTC, and the stop signal is collected from the bottom VTC. Ultimately, the time-to-digital converter (TDC) converts the time difference between the start and stop signal into digital output. To account for the inherent delay in the circuit, we use a redundant column of CBs that can be shared across the entire macro as a reference. The output of this reference is then fed into the TDC as the actual start signal.

C. Hierarchical YOCO Architecture

Memory and Compute Cell and Array: As shown in Fig. 3(b), each MCC includes a unit capacitor *Cu*, two switches *S1* and *S0*, two transistors *M0* and *M1* (as an analog 1-bit multiplier), and a memory cluster containing multiple 1-bit RAMs and a multiplexer (MUX). The 1-bit RAM can be implemented with either SRAM for performance-prioritized computation or 1b ReRAM for storage-density-prioritized computation. For 8-bit input resolution, each row of array contains at least 256 MCCs, divided into 9 groups by an *eDAC* with ratios of 1: 1: 2: …: 128. For 8-bit weight, each column includes at least 128 MCCs, with every 8 columns forming a CB. Within each CB, the output lines are split via *eACC* and *eSA* with ratios of 1: 2: …: 128. The array performs fully parallel VMM and each CB's output voltage representing the dot product of the weight and input vectors.

In-Situ Multiply Accumulate: To reduce data movement and minimize ADC usage, we integrate AiMC arrays in local domain to form an in-situ multiply accumulate (IMA) (Fig. 4). Arrays are interconnected directly through row drivers and column TDAs. Inputs are multicast across arrays horizontally via row drivers (indicated by blue arrows), while partial sum results from multiple arrays are aggregated vertically through TDAs (red arrows) and read out by an 8-bit TDC. Each array is controlled by power gating, allowing the computational scale of IMA to be reconfigurable and energy-saving. Both the input and output of IMA are equipped with 2KB buffers for temporary data storage and reuse.

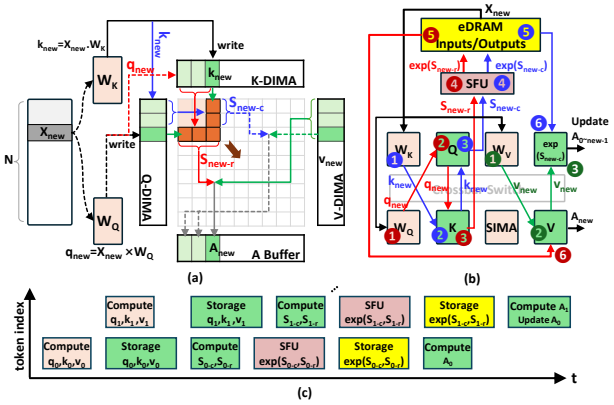


Fig. 5. IMC-friendly attention computation: (a) algorithm dataflow, (b) hardware dataflow, and (c) pipeline.

Tile and Accelerator with Hybrid Memory: A tile comprises eight IMAs, with half using SRAM for dynamic data processing and the other half using RRAM for static data processing. Dynamic and static IMAs (DIMA and SIMA) are interconnected through an internal crossbar switch, facilitating pipelined computing. Each tile includes a 128KB eDRAM as 8-bit inputs and outputs cache, a specialized function unit (SFU), and a quantization circuit includes 32KB memory. Like TIMELY, all tiles are interconnected via Hyper-Transport link. Upon completing a set of outputs, a tile can transfer data to other IMAs within the same tile via the internal I/O network or to IMAs of other tiles via on-chip network. Depending on the weight size, the accelerator can allocate one or multiple tiles to match the size of DNN layers.

D. IMC-Friendly Attention Computing Pipeline

As transformer models gain prominence in the AI community, optimizing their computational flow becomes increasingly important. Fig. 5 illustrates an IMC-friendly attention computing flow in algorithm level. In this flow, static weights W_Q, W_K, and W_V are pre-mapped into one or more SIMAs. The embedded word vector from eDRAM serves as the input vector for SIMA to generate *Q*, *K*, and *V* values for the current token, which are then loaded into DIMAs via an intra-tile crossbar switch. In *K-DIMA*, *K* is stored as weights in memory array, while *Q* is loaded into the input buffer and multiplied with all previously stored *K* values to get the score matrix. Conversely, in *Q-DIMA*, *Q* acts as the weights, and *K* is loaded into the input buffer to compute the scores with historical *Q* values. For each new input token, *K-DIMA* and *Q-DIMA* yields a $1\times n$ and $n\times 1$ score result ($S_{new\text{-}r}$ and $S_{new\text{-}c}$), respectively, where *n* denotes the current token position. Inspired by the flash attention approach, the newly generated score values undergo an exponential function transformation ($\exp(S_{new\text{-}r})$ and $\exp(S_{new\text{-}c})$) through the SFU and, along with the

updated l_{max} and m_{ij}, are stored in eDRAM. In subsequent cycles, these updated scores are multiplied by V to refine the attention output. Once the last token has been processed, the complete attention output can be obtained immediately. Fig. 5(b) shows the hardware dataflow between components within a tile during attention computation, and finally, the computations between tokens can be organized in a pipelined manner to enhance the inference speed of the transformer (Fig. 5 (c)).

IV. Evaluation

A. Experiment Setup

TABLE II. SUMMARY OF YOCO PARAMETERS

	Compo.	Num. &Size	Energy (pJ)	Latency (ns)	Area/comp. (μm^2)
MCC	Capacitor	2fF	1.62fJ/act.	-	0.8
	SRAM/1T1R	8/32	-	-	0.096
Array	MCC array	128×256	[a]26.5	13	26214
	Row driver	128	9.36fJ	<30ps	0.18
	Time Acc.	32	58.5fJ	113ps	5.3
IMA	Array	8×8	[a]29.6	<14.1	26406
	[b]TDC (8bits)	32×8	7.7	0.9	6865
	I/O Buffer	4KB	2.9/256b	0.112/256b	4656
Tile	IMA	8	4325	<15/VMM	3.45mm^2
	SFU	128	0.6	0.1	1398
	eDRAM	160KB	0.1pJ/bit	128GB/s	*0.2*mm^2
Chip	Tile	4	-	-	27.8 mm^2
Total	-	-	-	-	111.2 mm^2
Hyper Link	links/freq Line bw	1/1.6GHz 6.4GB/s	-	-	5.7 mm^2

[a]50% MCC activate; [b] Silicon-Verified by [8]

YOCO Configuration. All YOCO parameters, along with their energy, latency, and area, are summarized in Table II. Following the prior work [9], each MCC employs a 2 fF MOM capacitor for 8-bit computational resolution. The MOM capacitor can be stacked with memory cells, thus avoiding additional area in the layout. Multiple RAM units are organized as a memory cluster in MCC. To match the area of the MOM capacitor, each memory cluster in DIMA consists of 8 SRAMs, while in SIMA, each cluster contains 32 1T1R cells. Each array comprises 128×256 MCCs for 8-bit input and weight resolution, with each row driven by eight row drivers. Every eight columns of AiMC array form a compute-bar (CB), and the output of CB is connected to a column time accumulator. Each IMA consists of an 8×8 array, where analog computing results are read out in parallel by 32×8 TDCs. Both input and output data buffers for each IMA are set at 2 KB to enhance data reusability. Similar to ISAAC, YOCO integrates a special function unit (SFU) within each tile for complex function expansion and a 128 KB eDRAM for data caching.

Methodology. The YOCO model is constructed from multiple simulators or open-source data. Analog circuits, such as in-charge computing array and time-domain accumulator, are simulated in Cadence Virtuoso. TDC parameters are derived from previous report [10]. On-chip interconnects, SFU, buffer, and eDRAM are modeled using CACTI [11]. Parameters for ReRAM, featuring

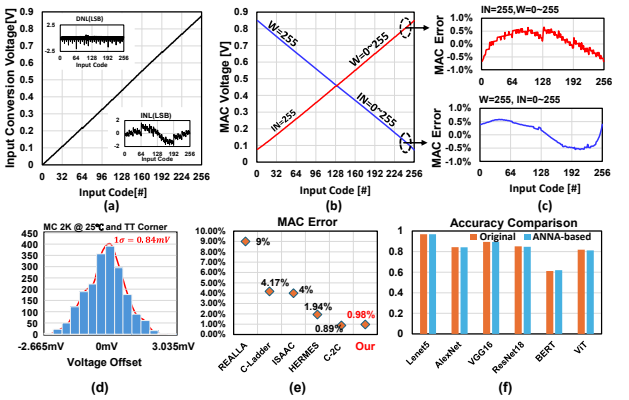


Fig. 6. An illustration of the (a) Transfer curve (TC) between input code and output voltage and correspond INL and DNL. (b) two TCs of 8-bit MAC with 128 input channels; and (c) correspond MAC error. (d) voltage offset under 2K Monte-Carlo simulation;(e) MAC error comparison; (f) DNNs inference accuracy comparison under 6 benchmarks.

1KΩ/20KΩ on/off resistance with 1-bit precision, are sourced from TIMELY [7]. Hyper-Transport links are based on ISAAC specifications [4]. All simulation data are then consolidated in the timeloop/accelergy framework, as referenced in REALLA [6], to perform overall architectural simulation. Given the 15ns computation latency of the IMA, the clock frequency for YOCO system simulation is set to 50MHz. Models of three baseline accelerators, ISAAC, TIMELY, and REALLA, are also developed in timeloop/accelergy [12]. For fairness, all accelerators are modeled at the 28nm process node, with shared components across designs.

Benchmarks. The overall YOCO architecture is evaluated over 10 DNN models, includes 5 CNN models (AlexNet, VGG16, ResNet18, MobileNetV3, and DenseNet201) and 5 transformed-based AI models (MobileBERT, QDQBERT, Vision Transformer, and LLaMA3-7B).

B. Experiment Results

Accuracy. Fig. 6(a) shows the transfer curve (TC) between input binary code and conversion voltage, along with integral (INL) and differential nonlinearity (DNL). Conversion errors are within two least significant bits (LSB) and typically under one LSB. Fig. 6(b) provides two TCs during the 8-bit MAC operation with 128 channels: the blue curve represents the MAC voltage with weights scanned from 0 to 255 while input is fixed at 255, and the red curve shows input scanning from 0 to 255 with weight fixed at 255. The corresponding MAC error (less than 0.68%) is shown in Fig. 6(c). PVT variation analysis using 2,000 Monte Carlo simulations at TT corner and room temperature (Fig. 6(d)) indicates a 3σ variation of 2.25 mV—less than one LSB (3.52 mV). The time accumulator's error is under 0.11%, keeping the analog MAC error below 0.79%. YOCO adopts 8-bit TDCs as the readout circuits for 8bit MAC resolution. The final end-to-end IMA computing error is less than 0.98% (see Fig. 6(e)). We also evaluated the impact of analog computing error for DNNs inference accuracy. Fig. 6(f) shows the accuracy comparison between fully-precision and YOCO-based inferences under 6 benchmarks. We observe less

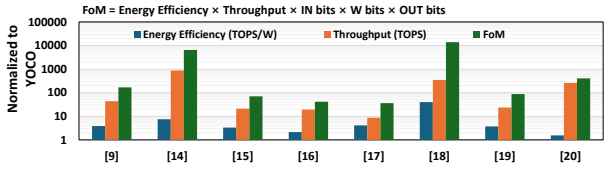


Fig. 7. The normalized VMM (a) energy efficiency and (b) throughput of YOCO's IMA over other SOTA design.

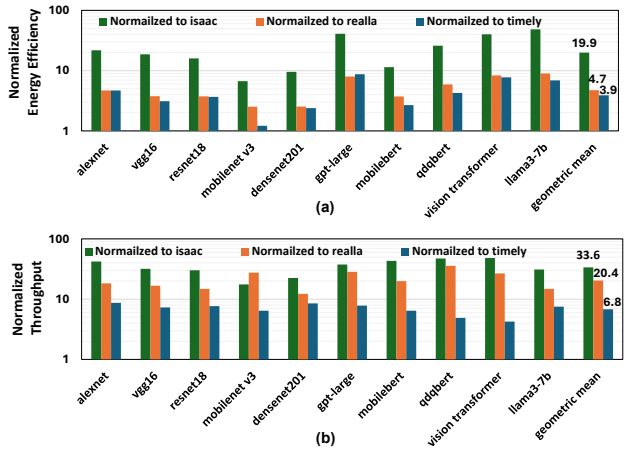


Fig. 8. (a) Efficiency and (b) throughput normalized to the ISAAC, REALLA, and TIMELY.

than 0.5% accuracy loss on 4 CNNs and less than 0.61% loss on 2 transformer-based models.

Energy Efficiency and Throughput. According to the summary in Table II, an IMA consumes approximately 4.235nJ of energy and incurs 15ns of latency to execute an 8-bit VMM with 1024×256 size. Based on discussion in [13], the average activation probability of MCCs is set to 50%. With these configurations and data, the IMA of YOCO achieves an energy efficiency of 123.8 TOPS/W and a throughput of 34.9 TOPS. Compared to 8 prior IMC circuits [9, 14-20], our design improves energy efficiency by 1.5× to 40×, throughput by 12× to 1164×, and the figure of merit (FoM) by 36× to 14,000× (Fig. 7). Furthermore, we evaluated the overall YOCO architecture on 10 CNN/transformer models to validate its superior performance. Fig. 8 shows the normalized energy efficiency and throughput over three state-of-the-art (SOTA) IMC architectures on 10 DNN models. On average, YOCO improves energy efficiency by 19.9×, 4.7×, and 3.9× and enhances throughput by 33.6×, 20.4×, and 6.8× compared to ISAAC, REALLA, and TIMELY, respectively (see the Geometric Mean in the rightmost part of Fig. 8).

Save ADCs/DACs Cost: Thanks to the innovative analog computing circuit design, YOCO significantly reduces the costs associated with expensive DACs and ADCs, while achieving ultra-high energy-efficiency and throughput. For DACs, YOCO uses grouped row capacitors to replace 8bits DAC, enabling low-cost input data conversion. Fig. 9(a) compares the DAC overhead between traditional and YOCO's methods, showing that YOCO reduces DAC area by 352×, energy consumption by 9×, and latency by 1.6×. For ADCs, YOCO adopts novel charge-domain

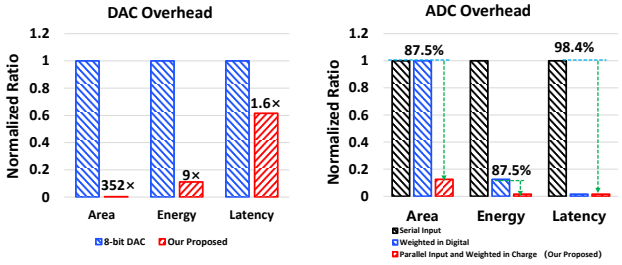


Fig. 9. (a) DACs and (b) ADCs overhead comparison with traditional data conversion strategies.

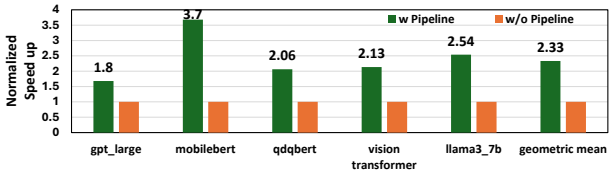


Fig. 10. The normalized speed up with or without pipeline design for 5 transformer models' inference.

computation and time-domain accumulation approaches to drive an all-analog 8-bit MAC, minimizing the number of ADC conversions while maintaining accuracy. As shown in Fig. 9(b), compared to bit-wise input , YOCO reduces the area, energy, and delay of ADC by 98.4%. When compared to weighted in digital-domain schemes, YOCO reduce ADC area and energy costs are reduced by 87.5% without delay cost.

Pipeline Performance. We also evaluated the effectiveness of the proposed IMC-based pipelined computation for accelerating attention layer. In 5 classic transformer models, compared to traditional layer-wise computation dataflow, the pipelined approach achieves speedups of up to 1.8× to 3.7× (geomean 2.3×) for attention inference (Fig. 10). The performance gains primarily stem from effectively hiding the computing latency of attention layer.

V. Conclusion

YOCO is the first work to demonstrate energy efficiency exceeding 100 TOPS/W in 8-bit fixed-point arithmetic. This breakthrough significantly alleviates the energy consumption bottleneck faced by large-scale model computations. YOCO marks the first successful demonstration of the superiority of analog-based computing. Benefits from hybrid memory architecture and optimized attention computation pipeline, YOCO is also uniquely positioned to tackle the rapidly evolving demands of large language models (LLMs). These innovations ensure that YOCO delivers both high performance and energy efficiency, making it a cutting-edge solution for future AI.

Acknowledgments

We sincerely thank Dr. Jia Chen for her valuable support and guidance throughout the manuscript submission process.

REFERENCE

[1] N. Maslej, L. Fattorini, E. Brynjolfsson, J. Etchemendy, K. Ligett, T. Lyons, J. Manyika, H. Ngo, J. C. Niebles, and V. Parli, "Artificial intelligence index report 2023," *arXiv preprint arXiv:2310.03715*, 2023.

[2] Y. Chen, Y. Xie, L. Song, F. Chen, and T. Tang, "A survey of accelerator architectures for deep neural networks," *Engineering,* vol. 6, no. 3, pp. 264-274, 2020.

[3] S. Yu, H. Jiang, S. Huang, X. Peng, and A. Lu, "Compute-in-Memory Chips for Deep Learning: Recent Trends and Prospects," *IEEE Circuits and Systems Magazine,* vol. 21, no. 3, pp. 31-56, 2021.

[4] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "ISAAC: a convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in Proceedings of the 43rd International Symposium on Computer Architecture, Seoul, Republic of Korea, 2016, pp. 14–26.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems,* vol. 30, 2017.

[6] T. Andrulis, J. S. Emer, and V. Sze, "RAELLA: Reforming the arithmetic for efficient, low-resolution, and low-loss analog PIM: No retraining required!." pp. 1-16.

[7] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, and Y. Lin, "TIMELY: pushing data movements and interfaces in PIM accelerators towards local and in time domain." pp. 832-845.

[8] S. Xie, C. Ni, A. Sayal, P. Jain, F. Hamzaoglu, and J. P. Kulkarni, "16.2 eDRAM-CIM: Compute-In-Memory Design with Reconfigurable Embedded-Dynamic-Memory Array Realizing Adaptive Data Converters and Charge-Domain Computing." pp. 248-250.

[9] H. Wang, R. Liu, R. Dorrance, D. Dasalukunte, D. Lake, and B. Carlton, "A Charge Domain SRAM Compute-in-Memory Macro With C-2C Ladder-Based 8-Bit MAC Unit in 22-nm FinFET Process for Edge Inference," *IEEE Journal of Solid-State Circuits*, pp. 1-14, 2023.

[10] Q. Chen, C. C. Boon, Q. Liu, and Y. Liang, "A Single-Channel Voltage-Scalable 8-GS/s 8-b >37.5-dB SNDR Time-Domain ADC With Asynchronous Pipeline Successive Approximation in 28-nm CMOS," *IEEE Journal of Solid-State Circuits,* vol. 58, no. 6, pp. 1610-1622, 2023.

[11] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6 . 0 : A Tool to Understand Large Caches."

[12] A. Parashar, P. Raina, Y. S. Shao, Y. H. Chen, V. A. Ying, A. Mukkara, R. Venkatesan, B. Khailany, S. W. Keckler, and J. Emer, "Timeloop: A Systematic Approach to DNN Accelerator Evaluation." pp. 304-315.

[13] Z. Xuan, and Y. Kang, "High-Efficiency Data Conversion Interface for Reconfigurable Function-in-Memory Computing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pp. 1-14, 2022.

[14] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie, and S. Yin, "A 28nm 29.2TFLOPS/W BF16 and 36.5TOPS/W INT8 Reconfigurable Digital CIM Processor with Unified FP/INT Pipeline and Bitwise In-Memory Booth Multiplication for Cloud Deep Learning Acceleration." pp. 1-3.

[15] H. Jia, M. Ozatay, Y. Tang, H. Valavi, R. Pathak, J. Lee, and N. Verma, "Scalable and Programmable Neural Network Inference Accelerator Based on In-Memory Computing," *IEEE Journal of Solid-State Circuits,* vol. 57, no. 1, pp. 198-211, 2022.

[16] P. C. Wu, J. W. Su, Y. L. Chung, L. Y. Hong, J. S. Ren, F. C. Chang, Y. Wu, H. Y. Chen, C. H. Lin, H. M. Hsiao, S. H. Li, S. S. Sheu, S. C. Chang, W. C. Lo, C. C. Lo, R. S. Liu, C. C. Hsieh, K. T. Tang, C. I. Wu, and M. F. Chang, "A 28nm 1Mb Time-Domain Computing-in-Memory 6T-SRAM Macro with a 6.6ns Latency, 1241GOPS and 37.01TOPS/W for 8b-MAC Operations for Edge-AI Devices." pp. 1-3.

[17] X. Si, Y. N. Tu, W. H. Huang, J. W. Su, P. J. Lu, J. H. Wang, T. W. Liu, S. Y. Wu, R. Liu, Y. C. Chou, Y. L. Chung, W. Shih, C. C. Lo, R. S. Liu, C. C. Hsieh, K. T. Tang, N. C. Lien, W. C. Shih, Y. He, Q. Li, and M. F. Chang, "A Local Computing Cell and 6T SRAM-Based Computing-in-Memory Macro With 8-b MAC Operation for Edge AI Chips," *IEEE Journal of Solid-State Circuits,* vol. 56, no. 9, pp. 2817-2831, 2021.

[18] Z. Chen, Z. Yu, Q. Jin, Y. He, J. Wang, S. Lin, D. Li, Y. Wang, and K. Yang, "CAP-RAM: A Charge-Domain In-Memory Computing 6T-SRAM for Accurate and Precision-Programmable CNN Inference," *IEEE Journal of Solid-State Circuits,* vol. 56, no. 6, pp. 1924-1935, 2021.

[19] B. Wang, C. Xue, Z. Feng, Z. Zhang, H. Liu, L. Ren, X. Li, A. Yin, T. Xiong, Y. Xue, S. He, Y. Kong, Y. Zhou, A. Guo, X. Si, and J. Yang, "A 28nm Horizontal-Weight-Shift and Vertical-feature-Shift-Based Separate-WL 6T-SRAM Computation-in-Memory Unit-Macro for Edge Depthwise Neural-Networks." pp. 134-136.

[20] S. E. Hsieh, C. H. Wei, C. X. Xue, H. W. Lin, W. H. Tu, E. J. Chang, K. T. Yang, P. H. Chen, W. N. Liao, L. L. Low, C. D. Lee, A. C. Lu, J. Liang, C. C. Cheng, and T. H. Kang, "7.6 A 70.85-86.27TOPS/W PVT-Insensitive 8b Word-Wise ACIM with Post-Processing Relaxation." pp. 136-138.

Computational Advantage in Hybrid Quantum Neural Networks: Myth or Reality?

Muhammad Kashif *†, Alberto Marchisio*†, Muhammad Shafique*†
* eBrain Lab, Division of Engineering, New York University Abu Dhabi, PO Box 129188, Abu Dhabi, UAE
† Center for Quantum and Topological Systems, NYUAD Research Institute, New York University Abu Dhabi, UAE
Emails: {muhammadkashif, alberto.marchisio, muhammad.shafique}@nyu.edu

Abstract*—Hybrid Quantum Neural Networks (HQNNs), under the umbrella of Quantum Machine Learning (QML), have garnered significant attention due to their potential to enhance computational performance by integrating quantum layers within traditional neural network (NN) architectures. Despite numerous state-of-the-art applications, a fundamental question remains: *Does the inclusion of quantum layers offer any computational advantage over purely classical models? If yes/no, how and why?

In this paper, we analyze how classical and hybrid models adapt their architectural complexity in response to increasing problem complexity. To this end, we select a multiclass classification problem and perform comprehensive benchmarking of classical models for increasing problem complexity, identifying those that optimize both accuracy and computational efficiency to establish a robust baseline for comparison. These baseline models are then systematically compared with HQNNs by evaluating the rate of increase in floating-point operations (FLOPs) and number of parameters, providing insights into how architectural complexity scales with problem complexity in both classical and hybrid networks. We utilize classical machines to simulate the quantum layers in HQNNs, a common practice in Noisy Intermediate-Scale Quantum (NISQ) era. Our analysis reveals that, as problem complexity increases, the architectural complexity of HQNNs, and consequently their FLOPs consumption, despite the simulation overhead associated with quantum layer's simulation on classical hardware, scales more efficiently (53.1% increase in FLOPs from 10 features (low problem complexity) to 110 features (high problem complexity)), compared to classical networks (88.1%). Moreover, as the problem complexity increases, classical networks consistently exhibit a need for a larger number of parameters to accommodate the increasing problem complexity. Additionally, the rate of increase in number of parameters is also slower in HQNNs (81.4%) than classical NNs (88.5%). These findings suggest that HQNNs provide a more scalable and resource-efficient solution, positioning them as a promising alternative for tackling complex computational problems.

I. Introduction

Quantum Machine Learning (QML) is an emerging research area that integrates quantum computing with machine learning (ML) techniques, aiming to enhance computational power and efficiency in processing complex datasets [1]–[4]. Quantum Neural Networks (QNNs) are a core component of QML, employing quantum circuits/layers to mimic the structure and function of classical neural networks (NNs) [5], [6]. Hybrid Quantum Neural Networks (HQNNs) advance this concept by combining quantum and classical NN, leveraging the strengths of both quantum and classical computation [7], [8]. HQNNs are particularly advantageous in the Noisy Intermediate-Scale Quantum (NISQ) era, where they can capitalize on quantum capabilities for specific sub-tasks while relying on classical resources to manage error-prone operations [9], [10]. In a typical HQNN architecture, one or more hidden layers of a classical NN are replaced by a trainable quantum layer [11], as shown in Fig. 1.

A growing body of state-of-the-art research employs HQNNs across a wide range of applications [12]–[20]. Despite the notable computational speedups demonstrated by quantum computing in various domains [21]–[24], no prior study has systematically investigated whether the integration of *quantum* components in NNs offers any advantages in learning performance or computational efficiency [25]. Additionally, demonstrating quantum advantage for QML is challenging due to several reasons, as critically discussed in [26].

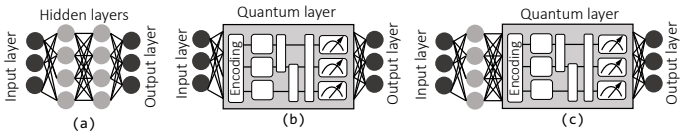


Fig. 1: Illustration of (a) classical NN, (b) HQNN with only quantum hidden layer and (c) HQNN with classical and quantum hidden layers.

A. Open Research Questions

Comparing classical and quantum models is not straight-forward due to their fundamentally different computational paradigms. In the NISQ era, where quantum computers are limited by noise and scale, most research relies on simulating quantum algorithms using classical hardware[1] [25], which is computationally intensive due to the exponential scaling of quantum states [27]. *By examining the increase in architectural complexity (required to address growing problem complexity) and the corresponding increase in computational demands of QML models simulated on classical machines, potential quantum advantages over classical models can be evaluated.* The argument is that, if QML models exhibit superior performance despite the overhead associated with simulating quantum circuits on classical hardware, this suggests that the observed superiority/advantage is likely inherent to the quantum nature of the algorithms and could become more significant with the deployment of universal fault-tolerant quantum computers. *This approach provides valuable insights into the scalability and impact of quantum computing, offering a pragmatic pathway to evaluate its future role in solving complex problems beyond classical capabilities.*

To this end, recently, there have been some efforts on benchmarking QML algorithms, primarily focusing on analyzing which algorithms perform better in specific scenarios [25], [28]. However, the question of whether incorporating the quantum component in NNs framework provides any advantage remains unexplored and is an *open and important* research problem. **In this paper, we raise several key questions to investigate this gap:**

- **Q1:** *What metrics will be appropriate to evaluate and compare the computational complexity of classical and hybrid networks?*
- **Q2:** *Does the quantum part in HQNNs add anything qualitatively different or important?*
- **Q3:** *If the quantum part gives any advantage in HQNNs, and can we demonstrate if it actually is computational advantage?*

B. Our Contributions

We attempt to answer the above questions through a rigorous empirical analysis. **Our main contributions are:**

- We propose a systematic methodology for comparing the computational complexity of hybrid and classical networks. For this purpose we generate a synthetic dataset in such a way that we can increase the complexity of the problem by increasing the number of features. Our novel investigation unleash how both classical and hybrid networks adapt to the complexity of the problem. (**Section III**)

[1]List of quantum simulators (Accessed 15-11-2024): https://www.quantiki.org/wiki/list-qc-simulators

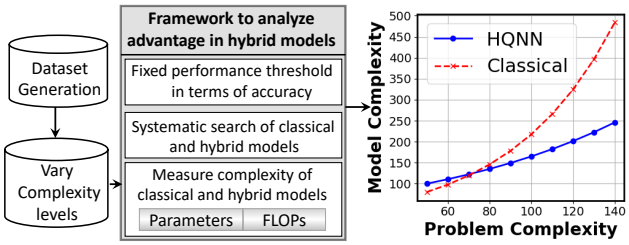


Fig. 2: An overview of our contributions.

- We perform grid search to identify the classical models that meet a predefined accuracy threshold across varying levels of problem complexity to serve as benchmarks. We then identify the most efficient hybrid network architectures (models that achieve the predefined accuracy threshold) through a similar grid search process. (**Section III-B and III-C**)
- A key finding from our analysis is that *as the problem complexity increases, classical NNs require more sophisticated and resource-intensive architectures to reach a certain accuracy, resulting in higher FLOPs consumption*. In contrast, *HQNNs demonstrate superior scalability and efficiency*. A well-designed hybrid architecture, particularly with a more *expressive quantum layer*, remains largely unaffected by the increasing complexity of the problem. An architecture that effectively solves a simpler problem can often be adapted, to solve a more complex problem. (**Section IV**)
- Another important finding of our analysis is that *classical models consistently require a greater number of parameters across all levels of problem complexity, even at lower complexity levels*. In contrast, *HQNNs generally need fewer parameters, highlighting their efficiency in parameter utilization.* (**Section IV**)

Summary of key results: This study demonstrates the superior scalability of HQNNs compared to classical NNs as problem complexity increases. In our analysis, we find that despite simulation overhead, HQNNs with expressive quantum layers require approximately 7.5% fewer FLOPs and 42.3% fewer parameters to solve complex problems (110 features in our case) than classical networks. Additionally, HQNNs scale more efficiently, with a 53.1% increase in FLOPs consumption while going from simple to complex tasks, compared to 88.5% increase in FLOPs for classical models. Similarly, the rate of increase in number of parameters for classical models is 88.5%, while HQNNs show a slower rate of increase in parameters that accounts to 81.4%.

II. Complexity of Neural Networks

There is no single, definitive measure for assessing the NN's complexity, however, several approximate metrics exist [29]–[34]. Therefore, we argue that relying on a single measure may not provide a comprehensive understanding of the network's complexity. Hence, it is advisable to use multiple methods to better estimate the network's complexity for a given task. We use FLOPs and the number of parameters as indicators to assess the network's complexity.

A. Floating Point Operations (FLOPs)

FLOPs are a widely recognized measure of computational complexity in NNs [35]–[38]. FLOPs provide an estimate of the total number of arithmetic operations required to execute a model on a given input. This metric serves as a proxy for understanding the computational resources, such as processing power and energy consumption that are needed to train and evaluate NNs. By quantifying FLOPs, one can objectively compare the efficiency of different architectures, enabling more informed decisions about model design and scalability.

B. Number of Parameters

The number of parameters represent a total number of trainable weights in a NN and is often used as a measure of model complexity, with larger networks (with more parameters) generally being more expressive and capable of capturing complex data patterns [39]. In NNs, the parameter count typically scales with the number of layers and neurons per layer, and is hence useful for evaluating model complexity [40].

III. Our Methodology

In this paper, we argue that two models are comparable if they achieve similar performance (accuracy) on a given task, regardless of differences in their size. Accordingly, we set an accuracy threshold of $\geq 90\%$ for both training and validation, and perform an extensive grid search for models that meet this condition regardless of their size. The detailed methodology is shown in Fig. 3. Below, we explain different steps of our methodology in detail.

A. Synthetic Dataset Generation

We generate synthetic datasets as these datasets play a crucial role in benchmarking ML models, offering precise control to systematically increase task complexity and evaluate model performance under varying conditions [25], [41], [42]. We generate a spiral dataset with 1500 data points distributed across 3 classes.

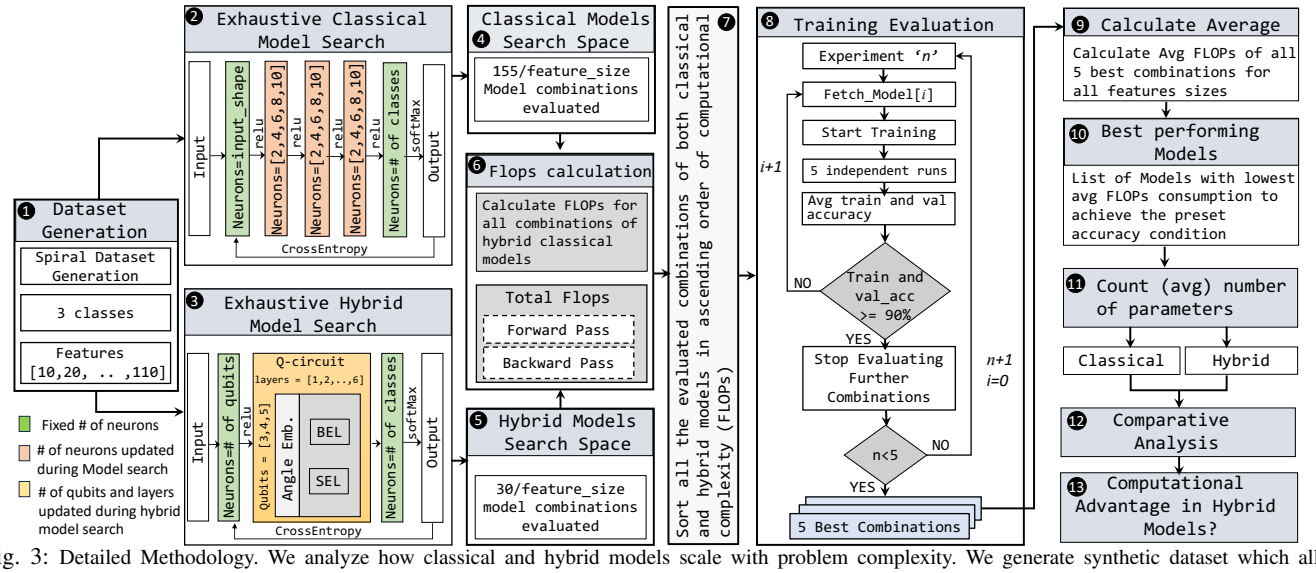


Fig. 3: Detailed Methodology. We analyze how classical and hybrid models scale with problem complexity. We generate synthetic dataset which allows to progressively increase the problem complexity. We then search for classical and hybrid models using grid search approach and compare the FLOPs and parameters. To account for randomness in NNs training, the results are averaged over 5 experiments. Small black boxes show the order of methodology steps.

The dataset's core structure is a spiral, as shown in Fig. 4(a) with each class represented by a distinct arm, offering a challenging yet learnable pattern for models. To investigate the effect of task complexity, we maintain a fixed number of data points and classes while systematically increasing complexity by augmenting the number of features. The increase in feature dimensions is accompanied by a corresponding increase in noise, further enhancing the complexity of the dataset. Specifically, the additional features introduce subtle variations through non-linear transformations of the existing features. Controlled noise is introduced, scaled with the number of features ($noise = 0.1 + 0.003 \times num_features$), ensuring a progressive rise in task difficulty. We utilize feature sizes ranging from 10 to 110 in increments of 10. From this point onward, we refer to feature size as complexity level, with the associated noise implicitly applied unless explicitly stated otherwise. This incremental increase in feature dimensions results in a progressively challenging classification task, as illustrated in Fig. 4(b).

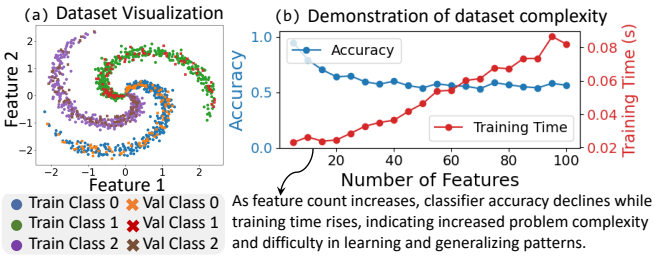


Fig. 4: (a) Visualization of first two features of generated dataset, and (b) Demonstration of increasing problem complexity.

B. Exhaustive Classical Model Search

We use grid search algorithm to systematically search for the models that reach the preset accuracy condition ($\geq 90\%$).

***Working of Grid Search Algorithm*:** For n layers, and m neuron options, our grid search algorithm evaluates all the possible combinations based on formula: Total no. of combinations $= m\frac{m^n-1}{m-1}$. For instance, if $m = [2, 3]$ (i.e., there can be 2 or 3 neurons in each layer, resulting in neuron options of $m = 2$), and $n = 2$, this means we can have minimum 1 and maximum 2 layers. Using the general formula, there are 6 possible combinations, i.e., $[2], [3], [2, 2], [2, 3], [3, 2], [3, 3]$.

Classical Model Search Space: We restrict the classical models to have a maximum of $n = 3$ layers, with the number of neurons in each layer chosen from the set $m = \{2, 4, 6, 8, 10\}$ ($m = 5$), resulting in a search space of total 155 model combinations for classical models for each complexity level.

C. Exhaustive Hybrid Model Search

For hybrid models, the number of neurons in the first and last classical layers are fixed. The number of neurons in the input layer are equal to the number of qubits because we use angle encoding that requires one qubit to encode one feature [43]. The neurons in the output layer are equal to the number of classes (3) in the dataset.

Hybrid Model Search Space: During the hybrid model search, only the quantum layers are varied. We use $[3, 4, 5]$ qubits quantum layers, and for each qubit size, quantum layers of depth $[1, 2, 3, ..., 10]$ are tested, yielding 30 model combinations per feature size.

Quantum Layer Design: For the underlying quantum layers in HQNNs, we use two different types designs which are separately evaluated: the Strongly Entangling Layer (SEL) and the Basic Entangling Layer (BEL), as shown in Fig. 5. For both layer types, each qubit number is experimented with all the layer depths, to identify the optimal configuration, i.e., that meets the preset accuracy threshold of $\geq 90\%$.

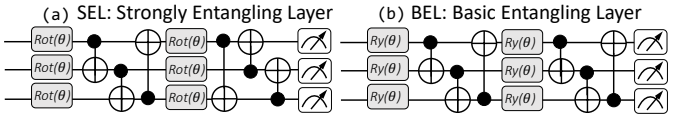


Fig. 5: Example of (a) SEL and (b) BEL quantum layer design, both having 3 qubits and depth of 2 layers. Same structure is repeated for more layers.

D. FLOPs Calculation

We use built-in functions from Tensorflow to compute the FLOPs for both and forward and backward pass. The procedure to compute FLOPs involves converting the `TensorFlow/Keras` models (both classical and HQNNs[2]) into a concrete function using TensorFlow's `convert_variables_to_constants_v2` utility. The frozen computation graph of the model is then analyzed using the `TensorFlow Profiler` with options set to compute the total number of floating-point operations.

For the forward pass, the profiler directly computes FLOPs based on the operations in the model graph. For the backward pass, we define a dummy loss function and use TensorFlow's `GradientTape` to compute gradients, which are also converted to a frozen computation graph. FLOPs for the backward pass are then calculated in a similar manner. The total FLOPs are obtained by summing the FLOPs of the forward and backward passes, providing a comprehensive measure of the model's computational complexity.

E. Models Sorting based on FLOPs consumption

Once the FLOPs are calculated, the model combinations are sorted in ascending order based on computational complexity (higher FLOPs means complex model and vice versa). This sorting ensures that we avoid training all models unnecessarily as the first model in the list that satisfies the accuracy condition is the model with the lowest computational complexity.

F. Models Training

The sorted list of models are trained sequentially for various levels of problem complexity. To account for the inherent randomness in parameter initialization and to ensure the statistical reliability of the results, each model configuration is evaluated over five independent runs. This approach mitigates the effects of stochastic variations and provides a more robust assessment of model performance across different initialization instances. Each run involves training for 100 epochs, and the highest training and validation accuracy across epochs is recorded. The maximum accuracies from all runs are averaged to provide a robust performance measure for each model combination for a given complexity level. If any model achieves an average training and validation accuracy of $\geq 90\%$, further evaluation of additional model combinations for that complexity level is stopped. This entire process is repeated five times to address the stochastic nature of NNs training, where different model configurations might result as optimal at different time instances. By repeating the procedure, a more accurate estimate of the average complexity required to solve the problem is obtained. Once five top-performing combinations are identified for a given complexity level, the evaluation continues with the next complexity level.

G. Models with Lowest Flops Consumption

At the end of the experimentation process, the average computational complexity required by both classical and hybrid networks is obtained for varying levels of problem complexity. These average complexities were then compared to determine whether hybrid networks provide any computational advantage over classical networks as the complexity of the problem increases. This comparative analysis is crucial in evaluating the scalability and efficiency of hybrid models, particularly in scenarios where the problem's complexity poses significant computational challenges for classical approaches.

[2]We convert quantum layers in HQNNs into Tensorflow/keras layer using pennylane, more details at: https://pennylane.ai/qml/demos/tutorial_qnn_module_tf

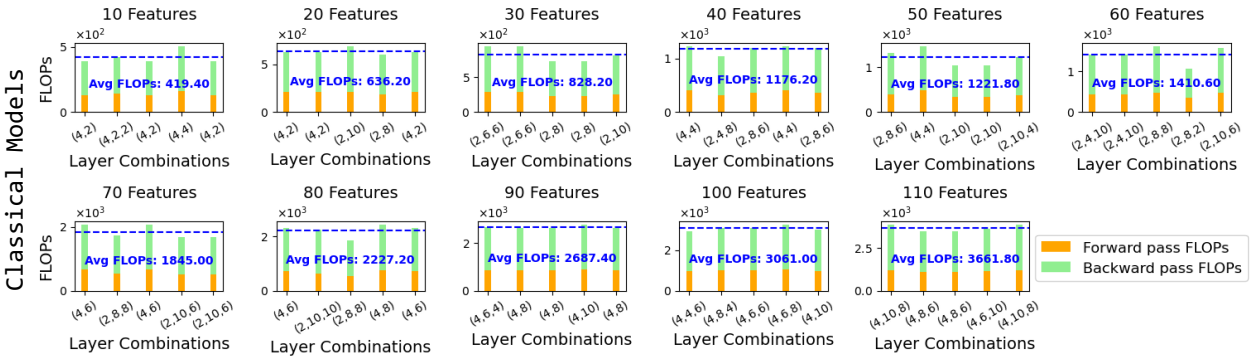


Fig. 6: FLOPs consumption of best-performing classical models for different complexity levels of the problem. Each subplot displays the FLOPs consumption of the top five performing models from five independent runs of model search, corresponding to different feature sizes.

IV. Results and Discussion

For the experiments, we utilized PennyLane, a Python-based framework designed for differentiable quantum programming [44]. During the training phase, the learning rate was set to 0.001, and a batch size of 8 was employed for both classical and hybrid models, with a total of 100 training epochs.

A. Classical Models: Analysis of FLOPs Consumption

In this section, we present our findings on the computational complexity (in terms of FLOPs consumption) required by classical models to solve the problem under predefined accuracy conditions, as shown in Fig. 6. By varying the problem complexity (increasing features and noise, see Section III-A), we evaluate the scalability and computational efficiency of classical models. We observe that at lower complexity, simpler models achieve the desired accuracy. However, as the problem complexity increases (e.g., up to 110 and associated high noise level), more sophisticated models, i.e., with additional layers and neurons are needed, resulting in a significant increase in FLOPs and hardware resources.

B. Hybrid Models: Analysis of FLOPs Consumption

We now present the results illustrating how both the hybrid models adapt to increasing problem complexity.

Discussion for BEL-based HQNNs: The results in Fig. 7 show the computational complexity of the BEL-based hybrid model for varying problem complexities. For problems with up to 40 features, the same quantum circuit architecture (3 qubits, 2 layers) is sufficient. The increase in FLOPs is mainly due to the increase in classical input layer size as feature count grows, while the quantum layer remains unchanged in depth (layer repetitions) and width (no. of qubits). However, further increase in problem complexity necessitates making the underlying quantum layer more expressive to effectively handle the increased complexity of the problem. This involves adding more qubits and increase quantum layer's depth. These changes in the architecture lead to a significant rise in FLOPs, reflecting an increase in the overall computational demands and greater hardware resource consumption.

Discussion for SEL-based HQNNs: The SEL quantum layer has a more intricate entanglement design than the BEL, enhancing its expressiveness and ability to capture complex patterns, as shown in Fig. 5(b). Unlike BEL, the SEL structure remains fixed at 3 qubits and 2 layers, effectively solving problems with up to 110 features without altering the quantum architecture, as shown in Fig. 8. The increase in FLOPs is attributed to the classical component's growth, particularly in the input layer to accomodate higher feature size, while the quantum layer remains unchanged, providing an efficient framework for handling complex tasks without additional quantum resources.

To make the argument tighter, we count the number of parameters of the best-performing models for both classical and hybrid approaches. This analysis is crucial, as it is possible that hybrid models have more parameters. This would potentially explain why their complexity in terms of rate of increase in FLOPs consumption does not increase as rapidly as classical models—they may be sufficiently expressive from the start. Due to page limitations, we present parameter counts for selective feature sizes $(10, 40, 80, 110)$ *for both classical and hybrid models.*

C. Classical Models: Analysis of # of Parameters

For classical models, we observed a substantial increase in the number of trainable parameters as problem complexity is increased. This is primarily due to the need for more sophisticated architectures, such as additional layers and neurons, to maintain the desired accuracy for increased complexity (Fig. 9, top panel). The trend suggests that classical models require enhanced expressiveness to manage the increased input dimensionality and noise. The increase in parameters correlates closely with the rise in FLOPs, further reinforcing the notion that classical models demand greater computational resources as problem complexity increases.

D. Hybrid Models: Analysis of # of Parameters

For BEL-based hybrid models, the model complexity remains constant for problem complexity up to 40 features, with an average

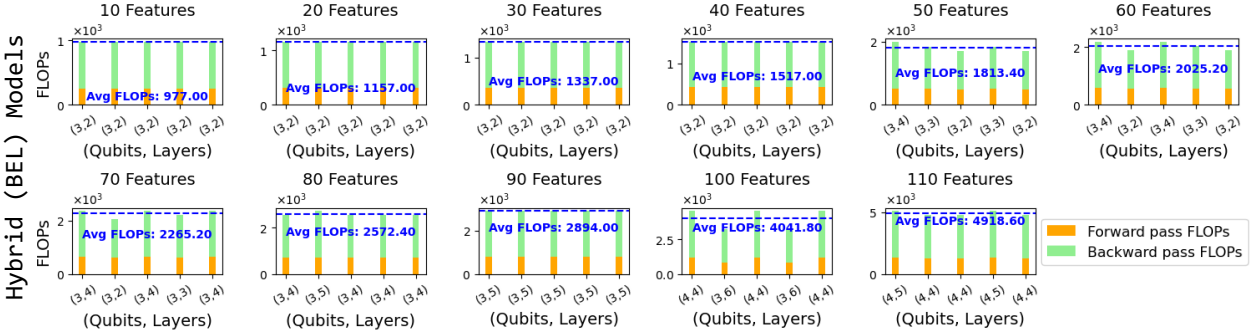


Fig. 7: FLOPs consumption of best-performing hybrid (BEL) models for different complexity levels of the problem. Each subplot displays the FLOPs consumption of the top five performing models from five independent runs of model search, corresponding to different feature sizes.

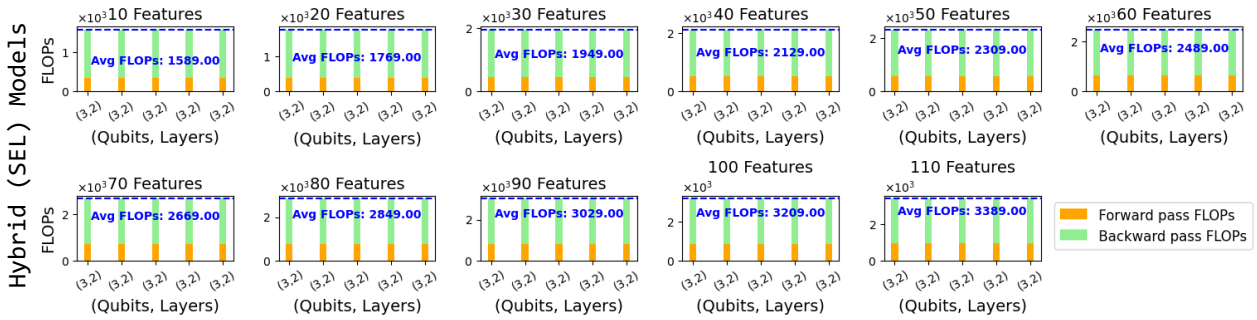


Fig. 8: FLOPs consumption of best-performing hybrid (SEL) models for different complexity levels of the problem. Each subplot displays the FLOPs consumption of the top five performing models from five independent runs of model search, corresponding to different feature sizes.

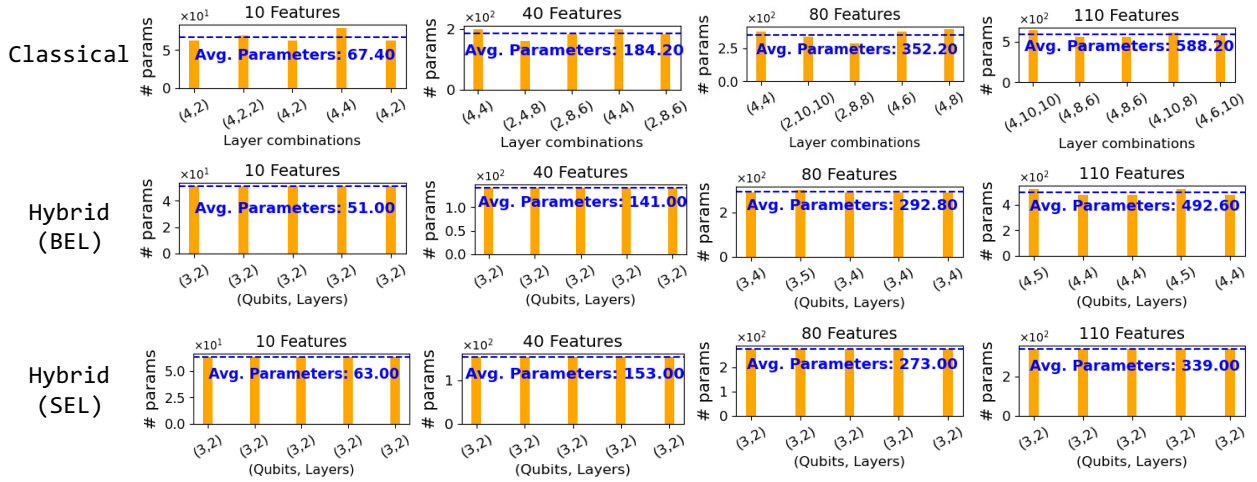


Fig. 9: Number for parameters for five top-performing classical models (top pannel), Hybrid (BEL) Models (middle pannel) and Hybrid (SEL) Models (bottom pannel) for different level of problem complexity.

parameter count lower than that of classical models. Beyond 40 features, however, an increase in the number of qubits and quantum circuit depth is necessary, leading to a moderate rise in parameter count (Fig. 9, middle panel). This increase is mainly due to the larger input layer resulting from the bigger feature size. Despite this, BEL-based hybrid models maintain a lower parameter count compared to their classical counterparts. In contrast, hybrid models with the SEL quantum layer exhibit minimal changes in parameter count across all complexity levels. The same quantum layer configuration is sufficient regardless of the problem complexity, with only a slight increase in parameters due to additional neurons in the input layer. Overall, SEL-based hybrid models demonstrate superior parameter efficiency compared to both classical and BEL-based hybrid models.

E. Classical Vs. Hybrid - Comparative Analysis of FLOPs and # of Parameters

We now compare the classical and hybrid models in terms of FLOPs consumption and the number of parameters. The comparative analysis is aimed to provide a clear picture of how the complexity of classical and hybrid models scale with problem complexity. As discussed in Section III-F, five independent experiments are conducted for each complexity level to identify the best-performing models. For this comparative analysis, we select the smallest model from the set of five best-performing configurations for both classical and hybrid approaches.

a) Comparison of FLOPs Consumption

FLOPs comparison of best-performing classical and both the hybrid models, as a function of the problem complexity is presented in Fig. 10(a). The results demonstrate how these models scale in terms of computational complexity as the problem complexity increases. The classical model exhibited a consistent linear rate of

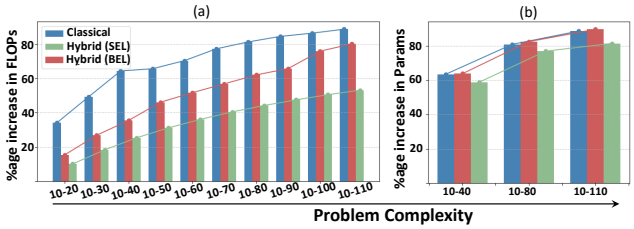


Fig. 10: Comparison of rate of increase in FLOPs and # of parameters of classical and HQNNs with increasing probem complexity. As the problem complexity increases, the rate of increase in FLOPs is more pronounced in classical models than hybrid models. Similarly, the rate of increase of parameter is also better in HQNNs especially with strongly entangling layers.

increase in FLOPs consumption with increasing problem complexity. Specifically, there was an absolute increase of 3285 FLOPs (obtained by subtracting the avg. FLOPs of 10 features from FLOPs of 110 features in Fig. 6), representing an 88.5% rise in computational demands when scaling from 10 to 110 features.

BEL-based hybrid models also exhibit a linear rate of increase in FLOPs but at a slightly lower rate than classical models, across the same complexity levels, with an absolute rise of 3941.6 FLOPs (obtained by subtracting the avg. FLOPs of 10 features from FLOPs of 110 features in Fig. 7). representing an increase of 80.13% computational demands when scaling from 10 to 110 features.. SEL-based hybrid models also showed a linear increase in computational demand, but at a significantly lower rate than both classical models and BEL-based hybrid models. The absolute increase was only 1800 FLOPs (obtained by subtracting the avg. FLOPs of 10 features from FLOPs of 110 features in Fig. 8), when the number of features increased from 10 to 110, reflecting 53.1% rate of increase in FLOPs. This is substantially lower than the rates observed for classical

and BEL-based models, despite the overhead of simulating quantum layers on classical hardware.

An important observation is that the rate of increase in computational demands, measured in terms of FLOPs consumption, is significantly lower in hybrid models compared to classical models. This underscores the superior adaptability of HQNNs to increasing problem complexity, particularly in HQNNs with more expressive underlying quantum layers, such as those utilizing SEL architectures.

b) Comparison of # of Parameters

Fig. 10(b) compares the average number of parameters for classical and both hybrid models as a function of problem complexity. Classical models exhibit a steady rise in parameter count with increasing feature size, reflecting the need for greater number of parameters and expressiveness to maintain a certain performance as problem complexity grows. Specifically, an absolute increase of 520.8 parameters is observed (obtained by subtracting the avg. parameters of 10 features from avg. params of 110 features in Fig. 9(top panel)), corresponding to an 88.5% rate of increase in parameter count, as the problem complexity increases from 10 to 110 features.

BEL-based hybrid models also exhibit an increase in parameter count with growing problem complexity. In particular, an absolute increase of 441 parameters (obtained by subtracting the avg. parameters of 10 features from avg. params of 110 features in Fig. 9(middle panel)), at a rate of 89.6% is observed, which is very close to the rate of increase of parameters in classical models. SEL-based hybrid models demonstrate the slowest parameter growth, with incrementally increased problem complexity. While going from low to high problem complexity an absolute increase of 276 parameters (obtained by subtracting the avg. parameters of 10 features from avg. params of 110 features in Fig. 9(bottom panel)) at a rate of 81.4% is observed, which is significantly lower than both classical and BEL-based hybrid models.

These results indicate that the rate of increase in parameter count as a function of problem complexity aligns closely with the rate of increase in FLOPs consumption. Notably, HQNNs employing more expressive quantum layers, such as SEL, demonstrate exceptional adaptability to increasing problem complexity, requiring minimal modifications to their architectural complexity. This observation suggests a potential computational advantage of incorporating quantum components into neural networks, highlighting the inherent benefits of quantumness in addressing complex problems.

F. Ablation Study: Determining the FLOPs of Quantum Layers

The HQNN framework incorporates classical layers for input preprocessing and postprocessing, as well as data encoding routines, which are computationally intensive when exploiting classical machines to run HQNN models. Once the universal fault-tolerant quantum computers and the development of proper quantum backpropagation techniques for QML algorithms would be developed, the reliance on classical layers and classical optimization routines is expected to diminish. Furthermore, the availability of quantum-native datasets would eliminate the need for data encoding, as the data would inherently exist in a quantum-compatible format. Therefore, we now compute the FLOPs consumption for classical and encoding components of HQNNs, to estimate the FLOPs consumption of actual trainable quantum layers. Table I provides a detailed breakdown. Understandably, as feature size increases, total FLOPs consumption increases for both hybrid models, due to the higher input dimensionality. The hybrid (BEL) models exhibit a more rapid increase, indicating greater sensitivity to input complexity. A substantial portion of the total FLOPs is consumed by classical layers and encoding. For instance, at a feature size of 110, the classical and encoding components in BEL-based hybrid model consumes 3901 FLOPs, representing the majority of the 4797 total FLOPs, whereas, the quantum layer only requires 896 FLOPs, about 18.7% of the total. On the other hand, for more sophisticated HQNN designs, particularly those employing SEL, the FLOPs consumption of the quantum layer remains constant, demonstrating the efficient handling of increasingly complex problems by more expressive quantum networks. This stability underscores the capability of these quantum layers to adapt to problem complexity without a proportional increase in computational demands.

TABLE I: Ablation study Results: Breakdown of FLOPs consumption for different across the different stages of the hybrid networks (in FLOPs). FS=feature size, BC=best combination, TF=total flops, Enc=Encoding, CL=classical layers, QL=quantum layer

Model	FS/BC	TF	Enc+CL	CL	Enc	QL
Hybrid (BEL)	10/(3,2)	977	749	283	749-283=466	228
	40/(3,2)	1517	1289	823	1289-823=466	228
	80/(3,4)	2537	2009	1543	2009-1543=466	528
	110/(4,4)	4797	3901	2769	3901-2769=1132	896
Hybrid (SEL)	10/(3,2)	1589	749	283	749-283=466	840
	40/(3,2)	2129	1289	823	1289-823=466	840
	80/(3,2)	2849	2009	1543	2009-1543=466	840
	110/(3,2)	3389	2549	2083	2549-2083=466	840

V. Conclusion

In this paper, we conducted a comprehensive benchmarking study to evaluate the computational efficiency of Hybrid Quantum Neural Networks (HQNNs) compared to classical neural networks (NNs). Our key findings aim at providing valuable insights to the research questions raised in the introduction as summarized below:

(A1) Since there exists no definitive measure of computational complexity in NNs, relying solely on a single metric to draw conclusions about computational complexity may lead to limited insights. Therefore, we employed two independent metrics, i.e., floating-point operations (FLOPs) and parameter count to evaluate computational complexity. We argue that if the trends observed in the results of these distinct metrics align, the conclusions derived from the analysis are more robust and meaningful.

(A2) Our analysis reveals that as the problem complexity increases, classical NNs demonstrate a substantial rise in both FLOPs and the number of parameters due to the requirement of more sophisticated architecture to maintain a certain accuracy. This indicates limited adaptability of classical NNs to the increasing complexity of the problem. In contrast, HQNNs exhibit a more efficient computational footprint. Specifically, quantum layers within HQNNs consume significantly fewer FLOPs and require fewer parameters as problem complexity grows, compared to classical models. Additionally, HQNNs adapt well to the growing problem complexity since the rate of increase in FLOPs and parameter count (while going from low to high complexity of the problem) is significantly lower in HQNNs compared to classical models. These findings indicate that HQNNs exhibit superior adaptability to complex problem scenarios, highlighting their potential advantages over classical models in managing computational complexity.

(A3) Given that both independent measures of complexity, i.e., FLOPs and parameter count demonstrate better adaptability in HQNNs to problem complexity over classical networks, we posit that there is strong evidence suggesting that the inclusion of quantum layers offers a computational advantage in NNs. We also advocate that further investigations using additional complexity measures are needed to fully understand the QML benefits over classical ML. Thus, we leave this question partially open for future research to explore with a broader set of metrics.

In summary, HQNNs offer a scalable, resource-efficient alternative to classical models, positioning them as a promising solution for complex tasks in machine learning.

Acknowledgements

This work was supported in part by the NYUAD Center for Quantum and Topological Systems (CQTS), funded by Tamkeen under the NYUAD Research Institute grant CG008.

REFERENCES

[1] M. Schuld *et al.*, "An introduction to quantum machine learning," *Contemporary Physics*, vol. 56, no. 2, pp. 172–185, 2015.

[2] Y. Zhang and Q. Ni, "Recent advances in quantum machine learning," *Quantum Engineering*, vol. 2, no. 1, p. e34, 2020.

[3] D. Peral-García, J. Cruz-Benito, and F. J. García-Peñalvo, "Systematic literature review: Quantum machine learning and its applications," *Computer Science Review*, vol. 51, p. 100619, 2024.

[4] K. Zaman, A. Marchisio, M. A. Hanif, and M. Shafique, "A survey on quantum machine learning: Current trends, challenges, opportunities, and the road ahead," *CoRR*, vol. abs/2310.10315, 2024. [Online]. Available: https://arxiv.org/abs/2310.10315

[5] Y. Kwak, W. J. Yun, S. Jung, and J. Kim, "Quantum neural networks: Concepts, applications, and challenges," in *2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 2021, pp. 413–416.

[6] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, "The power of quantum neural networks," *Nature Computational Science*, vol. 1, no. 6, pp. 403–409, 2021.

[7] M. Kashif *et al.*, "Design space exploration of hybrid quantum–classical neural networks," *Electronics*, vol. 10, no. 23, p. 2980, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/23/2980

[8] K. Zaman, T. Ahmed, M. Kashif, M. A. Hanif, A. Marchisio, and M. Shafique, "Studying the impact of quantum-specific hyperparameters on hybrid quantum-classical neural networks," *arXiv:2402.10605*, 2024. [Online]. Available: https://arxiv.org/abs/2402.10605

[9] M. Kashif and S. Al-Kuwari, "The unified effect of data encoding, ansatz expressibility and entanglement on the trainability of hqnns," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 38, no. 5, pp. 362–400, 2023. [Online]. Available: https://doi.org/10.1080/17445760.2023.2231163

[10] D. Bokhan, A. S. Mastiukova, A. S. Boev, D. N. Trubnikov, and A. K. Fedorov, "Multiclass classification using quantum convolutional neural networks with hybrid quantum-classical learning," *Frontiers in Physics*, vol. 10, p. 1069985, 2022.

[11] M. Kashif and S. Al-Kuwari, "Demonstrating quantum advantage in hybrid quantum neural networks for model capacity," in *2022 IEEE International Conference on Rebooting Computing (ICRC)*, 2022.

[12] L. Domingo, M. Djukic, C. Johnson, and F. Borondo, "Binding affinity predictions with hybrid quantum-classical convolutional neural networks," *Scientific Reports*, vol. 13, no. 1, p. 17951, 2023.

[13] Y. Dong, Y. Fu, H. Liu, X. Che, L. Sun, and Y. Luo, "An improved hybrid quantum-classical convolutional neural network for multi-class brain tumor mri classification," *Journal of Applied Physics*, vol. 133, no. 6, 2023.

[14] L. Domingo, M. Chehimi, S. Banerjee, S. H. Yuxun, S. Konakanchi, L. Ogunfowora, S. Roy, S. Selvaras, M. Djukic, and C. Johnson, "A hybrid quantum-classical fusion neural network to improve protein-ligand binding affinity predictions for drug discovery," *arXiv preprint arXiv:2309.03919*, 2023.

[15] S. Shi, Z. Wang, R. Shang, Y. Li, J. Li, G. Zhong, and Y. Gu, "Hybrid quantum-classical convolutional neural network for phytoplankton classification," *Frontiers in Marine Science*, vol. 10, p. 1158548, 2023.

[16] N. Innan, A. Marchisio, M. Shafique, and M. Bennai, "Qfnn-ffd: Quantum federated neural network for financial fraud detection," 2024. [Online]. Available: https://arxiv.org/abs/2404.02595

[17] Z. Kaseb, M. Möller, G. T. Balducci, P. Palensky, and P. P. Vergara, "Quantum neural networks for power flow analysis," *Electric Power Systems Research*, vol. 235, p. 110677, 2024.

[18] H.-Y. Chen, Y.-J. Chang, S.-W. Liao, and C.-R. Chang, "Deep q-learning with hybrid quantum neural network on solving maze problems," *Quantum Machine Intelligence*, vol. 6, no. 1, p. 2, 2024.

[19] A. Wang, J. Hu, S. Zhang, and L. Li, "Shallow hybrid quantum-classical convolutional neural network model for image classification," *Quantum Information Processing*, vol. 23, no. 1, p. 17, 2024.

[20] E. Paquet, F. Soleymani, G. St-Pierre-Lemieux, H. L. Viktor, and W. Michalowski, "Quantumbound–interactive protein generation with one-shot learning and hybrid quantum neural networks," *Artificial Intelligence Chemistry*, vol. 2, no. 1, p. 100030, 2024.

[21] F. Arute *et al.*, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[22] H.-S. Zhong *et al.*, "Quantum computational advantage using photons," *Science*, vol. 370, no. 6523, p. 1460–1463, Dec 2020. [Online]. Available: http://dx.doi.org/10.1126/science.abe8770

[23] Y. Wu *et al.*, "Strong quantum computational advantage using a superconducting quantum processor," *Phys. Rev. Lett.*, vol. 127, 2021. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.127.180501

[24] L. S. Madsen and F. e. Laudenbach, "Quantum computational advantage with a programmable photonic processor," *Nature*, vol. 606, no. 7912, pp. 75–81, 2022. [Online]. Available: https://doi.org/10.1038/s41586-022-04725-x

[25] J. Bowles, S. Ahmed, and M. Schuld, "Better than classical? the subtle art of benchmarking quantum machine learning models," *arXiv:2403.07059*, 2024. [Online]. Available: https://arxiv.org/abs/2403.07059

[26] M. Schuld and N. Killoran, "Is quantum advantage the right goal for quantum machine learning?" *PRX Quantum*, vol. 3, p. 030101, Jul 2022. [Online]. Available: https://link.aps.org/doi/10.1103/PRXQuantum.3.030101

[27] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, aug 2018. [Online]. Available: https://doi.org/10.22331\%2Fq-2018-08-06-79

[28] J. Schnabel and M. Roth, "Quantum kernel methods under scrutiny: A benchmarking study," *arXiv:2409.04406*, 2024.

[29] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[30] Z. Lu *et al.*, "The expressive power of neural networks: A view from the width," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.

[31] Y. Lee *et al.*, "Neural complexity measures," in *Advances in Neural Information Processing Systems*, vol. 33. Curran Associates, Inc., 2020, pp. 9713–9724.

[32] L. Szymanski *et al.*, "Conceptual complexity of neural networks," *Neurocomputing*, vol. 469, pp. 52–64, 2022.

[33] J. Holstermann, "On the expressive power of neural networks," *arXiv preprint arXiv:2306.00145*, 2023.

[34] D. A. Ehrlich *et al.*, "A measure of the complexity of neural representations based on partial information decomposition," *Transactions on Machine Learning Research*, 2023.

[35] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the International Conference on Machine Learning*, 2019, p. 6105–6114.

[36] Z. Guo *et al.*, "Flops-efficient filter pruning via transfer scale for neural network acceleration," *Journal of Computational Science*, vol. 55, p. 101459, 2021.

[37] S.-C. Hsia *et al.*, "Convolution neural network with low operation flops and high accuracy for image recognition," *Journal of Real-Time Image Processing*, vol. 18, no. 4, pp. 1309–1319, 2021.

[38] J. Chen, S.-h. Kao, H. He, W. Zhuo, S. Wen, C.-H. Lee, and S.-H. G. Chan, "Run, don't walk: Chasing higher flops for faster neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2023, pp. 12 021–12 031.

[39] I. Goodfellow *et al.*, *Deep learning*. MIT press, 2016.

[40] M. Bianchini and F. Scarselli, "On the complexity of neural network classifiers: A comparison between shallow and deep architectures," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 8, pp. 1553–1565, 2014.

[41] Y. Liu, S. Khandagale, C. White, and W. Neiswanger, "Synthetic benchmarks for scientific research in explainable machine learning," *arXiv:2106.12543*, 2021. [Online]. Available: https://arxiv.org/abs/2106.12543

[42] P. Orzechowski and J. H. Moore, "Generative and reproducible benchmarks for comprehensive evaluation of machine learning classifiers," *Science Advances*, vol. 8, no. 47, p. eabl4747, 2022.

[43] R. LaRose and B. Coyle, "Robust data encodings for quantum classifiers," *Phys. Rev. A*, vol. 102, p. 032420, Sep 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.102.032420

[44] V. Bergholm *et al.*, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv*, 2018. [Online]. Available: https://arxiv.org/abs/1811.04968

Efficient Weight Mapping and Resource Scheduling on Crossbar-based Multi-core CIM Systems

Hanjie Liu[†], Sifan Sun[†], Aifei Zhang[‡], Haiyan Qin[†], Yutong Wu[‡], Minhao Gu[‡],
Shihang Fu[‡], Shuaikai Liu[‡], Baosen Liu[‡], Wang Kang[*†]
[†]*National Key Laboratory of Spintronics, Hangzhou International Innovation Institute;*
School of Integrated Circuit Science and Engineering, Beihang University, China
[‡]*Zhicun Research Lab, Beijing, China*
[*]*Corresponding author: wang.kang@buaa.edu.cn*

***Abstract*—Crossbar-based computing-in-memory (CIM) systems facilitate large-scale parallel multiply-and-accumulate (MAC) operations, while a domain-specific compiler (DSC) plays a pivotal role in optimizing the deployment of neural network algorithms on such systems. With the development of multi-core and large-core architectures, some key compiler problems such as high parallel processing, resource utilization, and crossbar array assignment methods have not been solved. For low-latency application scenarios, we have designed a resource scheduling strategy for our hardware system based on stream data processing to reduce the latency caused by intra-core and inter-core communication. Additionally, a weight mapping strategy has been developed to maximize the potential of crossbar arrays in convolutional neural networks (CNNs) deployment. Experimental results on our multi-core eFlash-based CIM system-on-chip (SoC) demonstrate that these two technologies help CNNs achieve a 76% reduction in latency, a 30% improvement in resource utilization, and the use rate of crossbar array that can reach up to 94.7%.**

***Index Terms*—CIM, CNN, Compiler, Weight mapping, Resource schedule**

I. Introduction

In recent years, the extensive application of convolutional neural networks (CNNs) in machine learning has generated a substantial demand for efficient architectures to manage these computationally and data-intensive tasks. In response, various neural network accelerators [1], [2] have emerged. However, these CMOS-based devices, which adhere to the von Neumann architecture, are now facing the challenge posed by "the memory wall" [3].

Computing-in-memory (CIM) technology provides a promising solution for accelerating artificial intelligence inference by performing analog computations directly in memory, potentially reducing latency and power consumption. To effectively implement CIM, various techniques have been proposed, including commercial memory types such as SRAM, DRAM, and flash memory, as well as emerging non-volatile memory (NVM) technologies like ReRAM, PCM, FeFET, and MRAM [4]. The two-dimensional crossbar arrays constructed from these devices exhibit high storage density and the capability for parallel in-situ computation, thus garnering significant academic interest.

The development of macro designs for crossbar-based CIM accelerators has advanced rapidly, yet there remains a lack of detailed consideration for the execution of CNNs, necessitating manual intervention for actual deployment. As task complexity increases and neural networks continue to deepen, the demand for computational power also grows. Consequently, macro designs are evolving towards multi-core and large-core architectures, and there is an increasing call for automated deployment solutions.

Currently, the automation of compilation in some CIM systems focuses on small-core implementation strategies [5], [6]. These strategies enhance parallelism by partitioning convolution weights and distributing them across different cores for computation. However, this splitting introduces the need for summing and concatenating multi-core output results, which can accumulate significant quantization errors during actual computations [7]. While large-core computation eliminates these issues, it presents new challenges in optimizing resource utilization and data flow.

In our work, we achieve automated optimization and deploy CNNs on a multi-core CIM system, resulting in efficient, low-latency, and highly parallel task execution. Building on multi-core collaboration and large crossbar arrays, we propose two technical improvements for resource utilization. Our specific contributions are as follows:

- A multi-level abstract architecture and execution logic for a multi-core CIM system is proposed, serving as the foundation for detailed research on CNNs execution.
- A scheduling process is designed to improve system computational resource utilization and optimize latency caused by bandwidth limitations through resource pooling, static random allocation, and iterative evaluation.
- A new weight mapping strategy is designed for lager-core, which leverages weighted expansion to exploit the potential of crossbar arrays in processing convolutions.
- Experimental results on our multi-core CIM system-on-chip (SoC) demonstrate a 76% reduction in latency, a 30% improvement in resource utilization, and a crossbar array utilization rate of up to 94.7%.

II. Background and Motivation

Fig. 1 illustrates the basic operational unit for analog vector-matrix multiplication (VMM) in CIM systems, implemented within a two-dimensional crossbar array structure. This computational strategy can be categorized into two types based on the physical properties of the devices [8].

This work was supported by the Beijing MSTC Program (Z231100007423019), Beijing Natural Science Foundation (L223004), Natural Science Foundation of China (62274008).

A. Crossbar-based CIM

Fig. 1(a) shows the floating-gate memory cells array. In the subthreshold region, the leakage current of floating-gate devices exhibits an exponential dependence on the gate voltage. At the same time, the gate voltage difference can represent the ratio of input to output current. Therefore, we can program the device by adjusting the charge at the cell intersections and the peripheral unit charge, with the output current component expressed as: $I_{\Sigma i} = W_i I_i$. In the NVM crossbar structure shown in Fig. 1(b), the conductance value of each cell is programmable, and the current at each intersection is given by: $I_{\Sigma i} = G_i V_i$. According to Kirchhoff's law, the cumulative current value in each column is the sum of its components. Thus, the crossbar array completes the VMM operation.

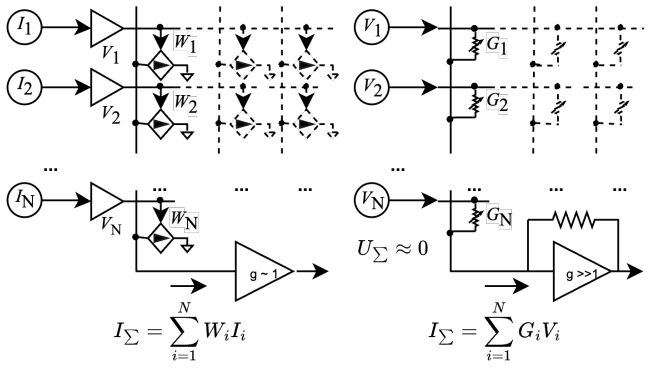


Fig. 1. Analog VMM circuits with Floating-gate crossbar and NVM crossbar. Note the difference in Op-Amp gain requirements.

NVM devices have fast write speeds and high write endurance, showcasing strong development potential. In contrast, floating-gate devices boast high density and mature processes, making them more advantageous in large-core designs [9]. Given that our SoC is built on floating-gate technology, our approach focuses on studying CIM systems based on large-scale floating-gate arrays.

B. Auto Development for CNNs

To achieve automated deployment of CNNs in CIM systems, industry and academia [10], [11] have introduced domain-specific compilers (DSC). These compilers typically consist of frontend and backend. The frontend is responsible for hardware-agnostic optimizations, including computational graph and operator optimization, often converting other operators into matrix computation operators such as convolution and fully connected. The backend emphasizes optimization strategies for the storage and computation of matrices involved in matrix-vector multiplication (MVM) as shown in Fig. 2.

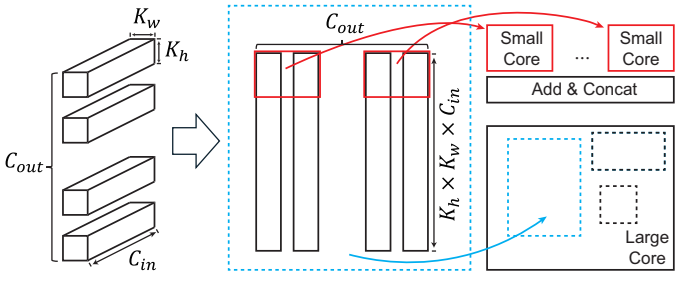


Fig. 2. Weight mapping of CNNs on different size of crossbar. K_w and K_h represent the width and height of convolution kernel, C_{in} and C_{out} represent the number of input and output channels. Weights are flattened and storage in the crossbar.

In systems based on NVM, to perform computations of larger-scale weight matrices within the limited size of crossbar arrays. Studies [5], [11] have adopted methods that split the original weights and distribute them across multiple cores for computation. This approach addresses the problem and enhances computational parallelism, Nevertheless, reorganizing the computation results introduces additional computational overhead and quantization errors.

In floating-gate and future NVM-based systems, when the weight matrix is smaller than the crossbar array, the entire matrix can be stored in a single core, thus eliminating the need for splitting computations and subsequent summation and stitching operations. However, maintaining high parallelism becomes a new challenge. Additionally, with the increase in single-core computational tasks, the optimization space for data flow both within and between cores also expands.

III. Architecture of CIM System

We propose an abstract architecture for a multi-core CIM system, which is compatible with widely adopted multi-level structures [12]. Based on this architecture, we developed an inference execution model tailored for low-latency application scenarios, enabling comprehensive analysis and evaluation of optimization strategies in the compilation backend.

A. Hardware Abstraction

Fig. 3 illustrates the proposed multi-level architecture, which sequentially expands from the Chip level to the Core, Matrix Computing Unit (MCU) and Crossbar levels.

At the highest level, the architecture consists of a two-dimensional homogeneous array of computing modules and a global buffer, where each computing module includes a CIM core and an inter-buffer. Modules are interconnected through a Network-on-Chip (NoC). The CNN parameters are pre-stored in the cores, and data exchange between cores occurs through the inter-buffer and global buffer. Different cores execute operations asynchronously, using a trigger-off mechanism for synchronization during inter-core communication.

Each core comprises a control unit, an inner-buffer, a set of MCUs, and other computation units (OCUs). These OCUs include element-wise operations (e.g., batch normalization), vector operations (e.g., pooling), and transformation operations (e.g., sampling). The computation units within the core can only access the data stored in the inner-buffer.

Each MCU consists of input/output first-in-first-out (FIFO) queues, pre-processing and post-processing units, and a set of crossbar arrays. The pre- and post-units perform operations such as shifting, clipping, expansion, and activation to support

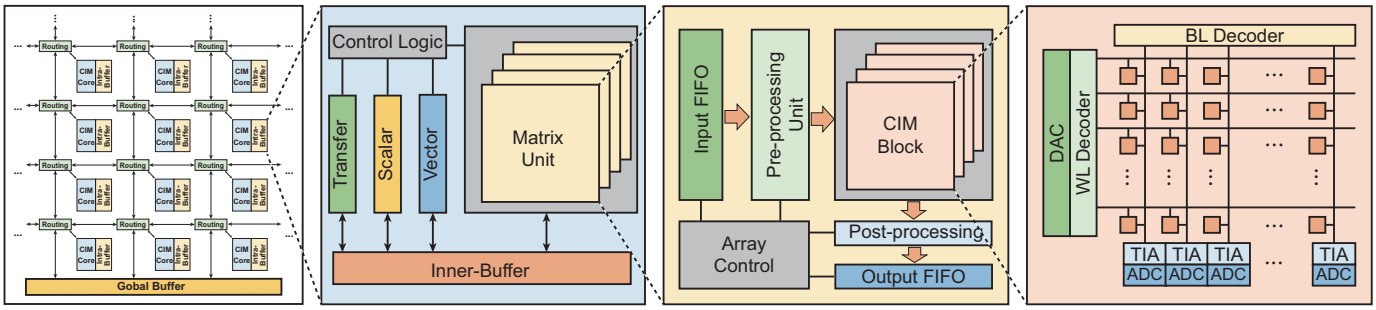


Fig. 3. Multilevel architecture of CIM system. Multiple cores interconneted through NoC in the top level. Each core contains various computing units, with MCU featuring multiple crossbars that share a common pre- and post-processing system. (a) Chip level. (b) Core level. (c) MCU level. (d) Crossbar level.

the fusion and acceleration of common operations before and after on convolution layers in CNNs.

Since the crossbar operates in the analog domain while the other circuits in the system function in the digital domain, basic peripheral circuits such as ADC/DAC and transimpedance amplifiers (TIA) are also required in the CIM Block.

B. Inference Execution

Algorithmic network tasks are divided into multiple chip pipelines for execution. Each pipeline covers a subnet, with its outputs stored in the global buffer, serving as the input for the subsequent pipelines.

Each chip pipeline is further divided into tasks, completed either within a single core or collaboratively across multiple cores. The outputs of each task, along with intermediate results across cores, are written into the inter-buffer.

Inner-core computations interact through a bank of inner-buffer. Each computation unit is allocated specific input and output spaces, facilitating parallel computation. Once a unit's designated input data is exhausted or its output space is filled, it must wait for the dependent units to process these data.

IV. Methodology

Given that the frequency of digital modules is typically much higher than that of crossbar arrays, we consider the computation time of the crossbar arrays T_{MVM} and the transfer of intermediate data T_{DT} across multiple levels as the primary components of latency T:

$$T = T_{DT} + T_{MVM} = \sum_i \frac{\text{Data}_{L_i}}{\text{Band}_{L_i}} + \sum_j \frac{F_{out_j}}{C_{out_j}} \quad (1)$$

$Data_{L_i}$ and $Band_{L_i}$ represent the data volume and bandwidth at each level, respectively. F_{out_j} and C_{out_j} represent the volume of output feature map and the output channel of the corresponding MVM layer, respectively. To reduce latency in the proposed architecture, two primary steps are involved:

- Optimize data transfer latency by allocating computational resources and intermediate data storage resources through resource scheduling.
- Optimize crossbar latency by using a weighted expansion strategy for weight mapping, which enhances resource allocation on individual crossbar arrays.

A. Resource Scheduling

Algorithm 1 outlines the overall process of scheduling. Initially, a resource management pool based on a multi-level architecture is instantiated according to the configuration parameters. Next, the computation graph of the CNN to be deployed is loaded and partitioned into multiple subgraphs. Computational and storage resources are then allocated to the operators in each layer in the subgraphs, generating the complete data flow. Finally, iterative optimization is performed on the data flow to minimize latency.

Algorithm 1 Resource Scheduling Algorithm

Require: ArchParams, CompGraph, EvalFunc
Ensure: Optimal resource allocation and data flow

```
Update resource pool by ArchParams //Resource Statistics
for each node in CompGraph do
    if random_probability < P then
        Create subgraph from current node
        for each sub_node in subgraph do
            Randomly allocate func to sub_node
        end for  //Resource Allocation
    else Update(P)
    end if
end for  //Network Partition
for each operator pair in CompGraph do
    buffer set: (diff subgraph) ? L0 : (diff core) ? L1 : L2
    buffer allocate
end for  //Data Flow Generation
for each generation do
    latency = EvalFunc(data_flow)
    Select best solutions based on latency
    Apply crossover to create new population
end for  //Evaluation Iteration
Return best solution found
```

a) Network Partition: To achieve a larger solution search space, we design the computation graph partitioning and resource allocation process to be highly stochastic. During the graph partitioning process, we determine whether to cut at the current node and generate a subgraph with a probability P,

which is expressed as:

$$P = \frac{1}{1 + e^{k(R-R_0)}}, \quad 0 \leq R \leq 1 \tag{2}$$

Where R represents the remaining resource rate. k is the scaling factor and R_0 is the threshold parameter.

b) Resource Allocation and Dataflow Generation: For computational resources, nodes' attributes are matched with the corresponding functional computation units in the resource pool without restricting the allocation to specific cores. For storage resources, only sizes are allocated, not specific addresses. The inner-buffer (L2) is allocated in banks, while the inter-buffer (L1) is assigned variable lengths, requiring a balance between resource usage and bandwidth bottlenecks. The global buffer (L0) acts as an intermediary for subgraphs, with its space partitioned based on the size of the final output of each subgraph. Following this, memory spaces at each level are matched according to the data dependencies between nodes in the subgraph, using a "proximity to computation units first" principle, ultimately generating a complete data flow.

c) Evaluation Iteration: An evolutionary algorithm is employed to search for the optimal solution within the solution space. The objective function is defined to minimize latency of data move:

$$T_{DT} = \sum_{C_{sg}} T_{DSG} + \max_{k}(T_{DCT_k}) \tag{3}$$

$$T_{DCT} = 2 * (\sum_{layer} T_{ex}^{l'} + \sum_{layer} T_{in}^{l}) \tag{4}$$

$$T_{DSG} = \frac{F_{out_{ln}}}{Band_{L_0}}, T_{ex}^{l'} = \frac{Data^{l'}}{Band_{L_1}}, T_{in}^{l} = \frac{Data^{l}}{Band_{L_2}} \tag{5}$$

The total data transfer latency T_{DT} is composed of the subgraph data latency T_{DSG} and the maximum core task data latency T_{DCT}. C_{sg} represents the number of subgraphs, and $F_{out_{ln}}$ denotes the output feature map size of the last node in each subgraph. The T_{DCT} is further divided into intra-core latency T_{in} and inter-core latency T_{ex}, which are computed as the corresponding data volumes divided by the bandwidth of the respective level buffer.

In the mutation phase, adjustments are made by altering the ownership of the endpoint nodes of subgraphs and alternating the computational units of the execution nodes. This method allows the system to explore various configurations, ultimately identifying the optimal solution that minimizes latency. Typically, this results in a data flow with very few cross points, as indicated by the red lines in Fig. 4.

B. Weight Mapping

Due to the varying scales of crossbar arrays and the weight parameters of different network layers, the utilization of crossbars significantly impacts overall computational efficiency. It is crucial to develop an optimal method to automatically allocate the entire network's parameters to the arrays, thereby facilitating the complete model deployment in CIM systems. Additionally, since each storage cell within the crossbar also functions as a computation unit, weight replication is a vital approach to enhancing computational parallelism and crossbar array utilization.

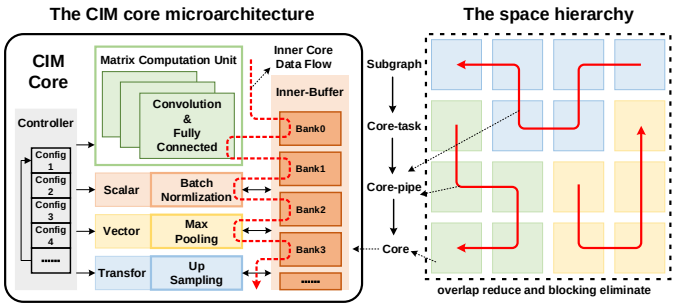


Fig. 4. Ideal smooth data flow lines within and between cores.

a) Integer Linear Programming: As shown in Fig. 6(a-c), mapping an 8-layer network onto a limited space. Different colored blocks represent the weight parameters of various MVM layers. In traditional sequential mapping methods, weight blocks are mapped to the crossbar array in the order they appear in the computation graph. When a block cannot fit into the available space, a new space is required, resulting in significant unused space. To address this issue, an Integer Linear Programming (ILP) space mapping strategy [13] has been proposed as an effective method to optimize crossbar utilization. This approach treats the boundary constraints of weight arrangement as an integer constraint problem, with the position of weight blocks as the ILP's objective. Through linear programming, a candidate set of all feasible solutions is constructed, and the optimal solution is selected.

b) Weight Replication: Replication strategies vary across different hierarchical levels. At the macro level, weight replication blocks are placed across different cores, directly enhancing parallelism by leveraging asynchronous inter-core operations. However, on a single crossbar array, to compute two inputs within the same cycle, it is necessary to concatenate the inputs. Due to the computational characteristics of crossbars, the fundamental approach involves diagonally separating the replicated weight blocks from the original ones to prevent computational interference. This approach necessitates significant zero-padding, as illustrated in Fig. 5(b). Given the sliding window nature of convolution operations, there is data reuse between adjacent input frames. Consequently, by merging

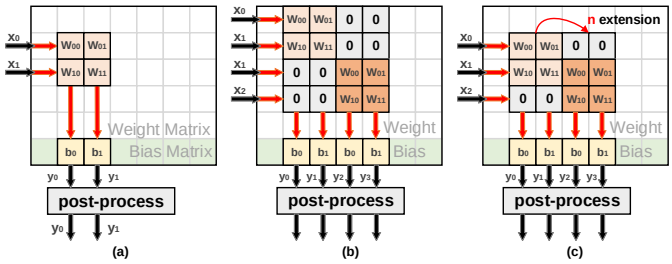


Fig. 5. (a) The original computation in crossbar array. (b) Weight replication for parallel computing and (c) OMM-like method to reduce zero padding.

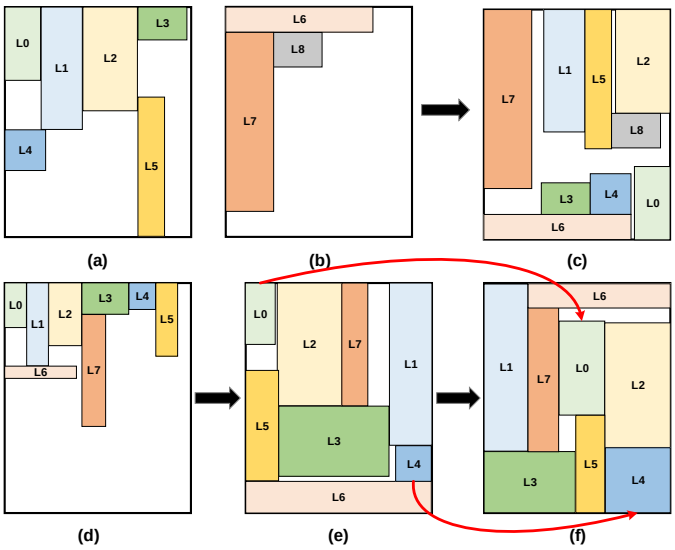


Fig. 6. Traditional sequential mapping methods (a)-(b), and ILP methods to improve the use ratio of crossbar. (d)-(e) ILP + OMM-like method for mapping and weighted skew on the bottleneck layer (L0 and L4).

input data, the zero-padding introduced by weight replication can be minimized. This method is similar in concept to the overlapped mapping method (OMM) [14].

Clearly, within a single large core, the OMM-like can help the system achieve an equivalent trade-off between crossbar area and computational efficiency. When increasing the output by M times, the crossbar utilization η (example for horizontal direction) can be determined using (6) and (7), where S_{array} denotes the area of the crossbar array.

$$T'_{MVM} = \frac{F_{out}}{M * C_{out}} \tag{6}$$

$$\eta = \frac{M * C_{out} * [K_w * K_h + (M-1)K_w]}{S_{array}} \tag{7}$$

Therefore, we have refined the original ILP by incorporating the horizontal and vertical expansion of each layer's weight parameters as new variables in the optimization process. Additionally, maximizing utilization has been introduced as one of the primary objectives. This enhancement has led to an increase in large core utilization to over 90%, and the substantial reduction in latency is directly proportional to the degree of weight expansion.

c) Weighted extension: In convolutional computations, the execution efficiency of different layers varies, often related to their sliding window operations. To mitigate the bottlenecks caused by excessive sliding window operations, we introduced a weighted scoring mechanism on top of the ILP + OMM-like approach. The reward function is designed as follows:

$$N_{w/h} = \left\lfloor \frac{IF_{w/h} - K_{w/h} + 2P_{w/h}}{S_{w/h}} \right\rfloor + 1 \tag{8}$$

$$N = N_w * N_h \tag{9}$$

$$F_{\text{Rewards}} = \frac{N}{M} x^{\frac{1}{\alpha}}, \quad \alpha > 1 \tag{10}$$

Where: N is the number of sliding window operations for the current layer. $IF_{w/h}$ represents the width and height of the input feature map. K, P, and S denote the kernel, padding, and stride, respectively. M is a normalization adjustment parameter, and F_{Rewards} is a fitted exponential function, where x is the extension factor based on the original weights. In the process of weighted extension solving in Fig. 6(d-f), array resources are prioritized for allocation to the bottleneck layers, resulting in further reduction of latency.

V. EVALUATION

A. Experiment Setup

a) Architecture Configuration: The final evaluation experiments were conducted directly on our SoC, which employs an eFlash floating-gate crossbar scheme. This SoC features a three-level buffer and consists of four large cores interconnected via a NoC. Each core contains eight MCUs and five OCUs, with MCU and OCUs executing in parallel, though only one MCU can execute at a time. The MCUs are organized in pairs to form a pipelined ping-pong structure, achieving one MVM every 120ns. The eFlash cell's precision is 8-bit, with inputs, outputs, and weights of crossbar all represented as 8-bit fixed-point numbers. Detailed configurations are in Table I.

TABLE I
METRICS OF OUR MULTI-CORE EFLASH-BASED CIM SOC

Component	***Params***	***Specific***	***Params***	***Specific***
Chip	core array	4	-	-
Global buffer	size	4MB	bandwidth	8GB/s
Core	MCU	2X4	OCU	5
	DAC	1152	ADC	320
Inter buffer	size	512KB	bandwidth	32GB/s
Inner buffer	bank num	8	bandwidth	64GB/s
	bank size	32KB		
Crossbar	weight size	1152*960	bias size	32*960
latency	MCU	240ns	OCU	4ns

b) benchmarks and baselines: We used commonly employed networks in computer visual scenarios (Yolov5, Gsr, Gvfe, Resnet50 and Unet) as benchmarks. Linear scheduling (sequential allocation in order) and OMM-like mapping strategies were selected as the baselines for our scheduling and mapping comparisons. We obtained latency measurements through direct testing on the SoC. Additionally, we developed a performance analysis simulator capable of event-level simulation in multi-level architectures, allowing us to analyze other performance metrics. These include different levels of buffer usage, computational unit utilization, and the time consumption of specific components.

B. Experiment Result

a) Resource Utilization: Fig. 8(left) illustrates the reduction in the number of chip pipelines for network tasks with varying complexities. This reduction indicates that more operators are processed within a single chip pipeline, leading to a maximum increase in computational resource utilization by up to 1.4 times. Additionally, storage resource utilization improved by up to 30%. Fig. 8(right) shows the variation

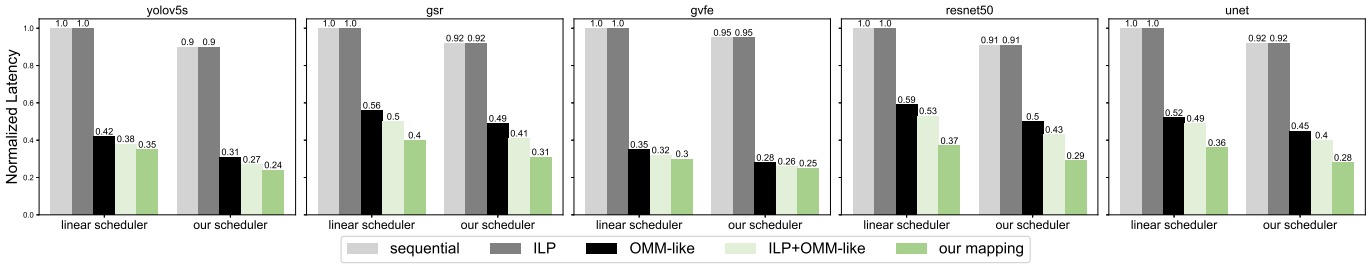


Fig. 7. The latency improvement with different strategy on benchmarks.

in data transfer latency between different levels of buffers. Although the latency of L1 increased, the higher bandwidth of L1 compared to L0 resulted in an overall 8% reduction in total latency.

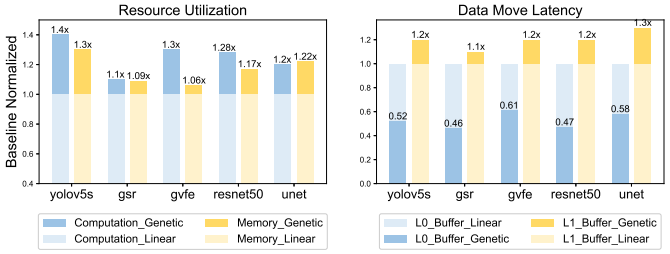


Fig. 8. The comparison of resource utilization for linear-method and our scheduler, and the latency change in different level of buffer.

b) Crossbar Utilization: Fig. 9 displays the improvements in area utilization and computational efficiency of crossbar arrays by different mapping strategies. In the case of large cores, ILP does not optimize performance without weight replication. However, with weight replication, there is a noticeable increase in area utilization, peaking at 94.7%. Although the overall area utilization decreases slightly after applying weighted skewing resources to address bottleneck layers, the latency is further improved. The final scheme achieves an average latency improvement of 28.2% compared to OMM-like strategies.

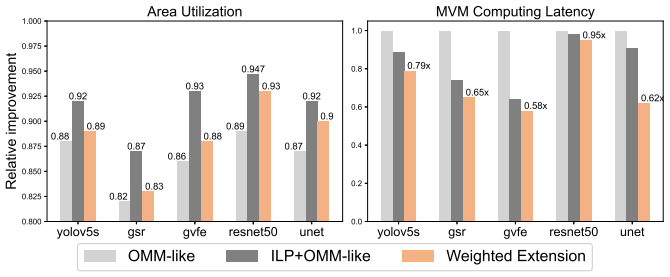


Fig. 9. Crossbar use ratio and computing latency of different mapping strategy.

c) Latency: Fig. 7 shows the latency performance of benchmark deployments on our SoC, corresponding to the combination of all optimization strategies. The data indicates that our resource scheduling and weight mapping strategies independently contribute to a maximum latency reduction of 10% and 70%, respectively. For the combined deployment on the Yolov5s network, the overall latency was reduced by 76%.

VI. Conclusion

This work pioneers the automatic optimization and deployment of multi-core, large-array CIM SoCs. Previous research has often overlooked domain-specific compilers in multi-core, large-array systems, with some efforts still confined to simulation stages. We addressed the growing demand for low-latency edge scenarios by developing optimization strategies for resource scheduling and weight mapping. Experimental results demonstrate a 30% improvement in resource utilization, a 94.7% crossbar array utilization, and a 76% reduction in latency based on actual chip measurements.

References

[1] K. H. Lee and et al., "A Low-Power Processor With Configurable Embedded Machine-Learning Accelerators for High-Order and Adaptive Analysis of Medical-Sensor Signals," JSSC, vol. 48, no. 7, pp. 1625-1637, 2013.
[2] T. Chen and et al., "DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning," in ASPLOS, pp. 269–284, 2014.
[3] K. Lee and et al., "A Charge-Sharing based 8T SRAM In-Memory Computing for Edge DNN Acceleration," DAC, pp. 739-744, 2021.
[4] C. Wolters and et al., "Memory Is All You Need: An Overview of Compute-in-Memory Architectures for Accelerating Large Language Model Inference," arXiv: 2406.08413 [cs.AR].
[5] R. Pelke and et al., "Mapping of CNNs on multi-core RRAM-based CIM architectures," VLSI-SoC, pp. 1-6, 2023.
[6] K. E. Jeon and et al., "Weight-Aware Activation Mapping for Energy-Efficient Convolution on PIM Arrays," ISLPED, pp. 1-6, 2023.
[7] J. Bai and et al., "CIMQ: A Hardware-Efficient Quantization Framework for Computing-In-Memory-Based Neural Network Accelerators," TCAD, vol. 43, no. 1, pp. 189-202, 2023.
[8] X. Guo and et al., "Fast, energy-efficient, robust, and reproducible mixed-signal neuromorphic classifier based on embedded NOR flash memory technology," IEDM, pp. 6.5.1-6.5.4, 2017.
[9] Y. Feng and et al., "A Novel Array Programming Scheme for Large Matrix Processing in Flash-Based Computing-in-Memory (CIM) With Ultrahigh Bit Density," TED, vol. 70, no. 2, pp. 461-467, 2023.
[10] C. Yang and et al., "CIMAX-Compiler: An End-to-End ANN Compiler for Heterogeneous Computing-in-Memory Platform," WCCCT, pp. 152-157, 2023.
[11] X. Sun and et al., "PIMCOMP: A Universal Compilation Framework for Crossbar-based PIM DNN Accelerators," DAC, pp. 1-6, 2023.
[12] L. Han and et al., "A Convolution Neural Network Accelerator Design with Weight Mapping and Pipeline Optimization," DAC, pp. 1-6, 2023.
[13] T. Bai and et al., "An End-to-End In-Memory Computing System Based on a 40-nm eFlash-Based IMC SoC: Circuits, Toolchains, and Systems Co-Design Framework," TCAD, vol. 43, no. 6, pp. 1729-1740, 2024.
[14] Z. Zhu and et al., "Mixed Size Crossbar based RRAM CNN Accelerator with Overlapped Mapping Method," ICCAD. pp, 1–8, 2018.

Cost-Distance Steiner Trees for Timing-Constrained Global Routing

Stephan Held and Edgar Perner

Research Institute for Discrete Mathematics and Hausdorff Center for Mathematics
University of Bonn
Bonn, Germany
held@dm.uni-bonn.de and perner@dm.uni-bonn.de

***Abstract*—The cost-distance Steiner tree problem seeks a Steiner tree that minimizes the total congestion cost plus the weighted sum of source-sink delays. This problem arises as a subroutine in timing-constrained global routing with a linear delay model, used before buffer insertion. Here, the congestion cost and the delay of an edge are essentially uncorrelated, unlike in most other algorithms for timing-driven Steiner trees.**

We present a fast algorithm for the cost-distance Steiner tree problem. Its running time is $\mathcal{O}(t(n \log n + m))$, where t, n, and m are the numbers of terminals, vertices, and edges in the global routing graph. We also prove that our algorithm guarantees an approximation factor of $\mathcal{O}(\log t)$. This matches the best-known approximation factor for this problem, but with a much faster running time.

To account for increased capacitance and delays after buffering caused by bifurcations, we incorporate a delay penalty for each bifurcation without compromising the running time or approximation factor.

In our experimental results, we show that our algorithm outperforms previous methods that first compute a Steiner topology, e.g. based on shallow-light Steiner trees or the Prim-Dijkstra algorithm, and then embed this into the global routing graph.

***Index Terms*—Cost-distance Steiner tree, timing-constrained global routing**

I. Introduction

Modern routers do not only aim for minimum total wire length, but need to consider signal timing as well [1], [8], [13], [15], [20]–[22]. Before and in preparation of buffer insertion, this is usually estimated using a linear delay model, where the delay depends linearly on the wire length, as well as on wire type and layer [4], [18]. There are many Steiner tree variants that take path lengths into account [2]–[4], [6], [14]. They do not consider independent congestion and delay costs.

The cost-distance Steiner tree problem overcomes this limitation. It asks for a Steiner tree minimizing the sum of routing congestion cost and weighted path delays through the tree. It occurs as the central sub-problem in timing-constrained global routing, where it is usually solved heuristically [13].

We are given a graph G with edge cost $c : E(G) \to \mathbb{R}_+$ and edge delays $d : E(G) \to \mathbb{R}_+$. In our application G is a 3D global routing graph, where an edge cost $c(e)$ arises from the current edge usage and $d(e)$ is the delay of e in a linear delay model. If multiple wire types (width and spacing configurations) are available G may have a parallel edge for each wire type that has an individual cost and delay.

Given a net connecting a set of sinks S to a root r, delay weights $w :\to \mathbb{R}_+$ and positions $\pi : S \cup \{r\} \to V(G)$, we seek for a Steiner tree T connecting $S \cup \{r\}$ that is embedded into G, i.e. has an extension $\pi : V(T) \setminus (S \cup \{r\}) \to V(G)$ and minimizes

$$\text{cost}(T) = \sum_{e \in T} c(e) + \sum_{t \in S} w(t) \text{delay}_T(r, t), \tag{1}$$

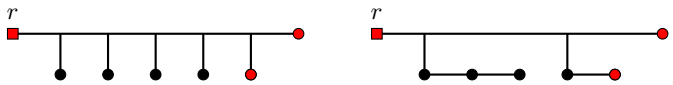


Fig. 1. Two trees for the same net, the right one with fewer bifurcations on the path from the root r to critical sinks (red dots).

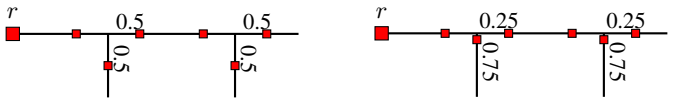


Fig. 2. Two buffering solutions (small red squares) indicating different delay tradeoffs between at the branchings.

where $\text{delay}_T(r, t) = \sum_{e \in E(T_{[r,t]})} d(e)$ in a simple linear delay model. The delay weights indicate the criticality of a sink and arise from a Lagrangean relaxation in timing-constrained global routing [13].

Throughout this paper, we use $t := |S \cup \{r\}|, n := |V(G)|$ and $m := |E(G)|$. The best known approximation factor for this problem is $\mathcal{O}(\log t)$ based on iteratively computing a matching of terminals that are served by a Steiner vertex based on a special distance function [5], [17] with a running time of $\mathcal{O}(t^2(n \log n + m))$. It does not permit an approximation factor better than $\Omega(\log \log t)$ unless NP $\subseteq$ DTIME$(t^{\mathcal{O}(\log \log \log t)})$ [7]. For the uniform case, where $c = d$, constant approximation factors are possible [9], [17].

Bifurcations on a path increase the capacitance and delay after buffering. In Figure 1, we see a critical path with many bifurcations on the left, which is unfavorable to the tree on the right after buffering. To mitigate this, we use a delay penalty for each bifurcation, as proposed in [4].

As buffering has some freedom to shield off capacitance (see Figure 2), we use a model where the bifurcation penalty can be distributed within certain limits. For technical reasons, we assume that the root and the sinks form the leaves of T and internal vertices have degree at most 3. We say that such a tree is *bifurcation compatible*. As we allow multiple vertices with the same position, any Steiner tree can be transformed into such a tree without changing the total length or any source-sink length in the tree.

We can consider T as an r-arborescence. Then, at every proper Steiner vertex u with $\delta_T^+(u) = \{x, y\}$ we distribute a delay penalty to (u, x) and (u, y) as follows. Let $d_{bif} \in \mathbb{R}_+$ be the total bifurcation penalty for both branches and $0 \le \eta \le 1/2$. For $\lambda_x \in [\eta, 1 - \eta]$, we assign a delay penalty of $\lambda_x d_{bif}$ to (u, x) and $\lambda_y d_{bif}$ to (u, y) with $\lambda_y = (1 - \lambda_x)$.

In the cost-distance objective with weighted delays, the optimum choice of λ_x depends only on the total delay weights in the two

subtrees:

$$\lambda_x = \begin{cases} \eta & \text{if } w(S \cap V(T_x)) > w(S \cap V(T_y)) \\ 0.5 & \text{if } w(S \cap V(T_x)) = w(S \cap V(T_y)) \\ (1-\eta) & \text{else.} \end{cases} \quad (2)$$

If $w(S \cap V(T_x)) = w(S \cap V(T_y))$ any choice is optimal. T_x is the sub-arborescence rooted at x and $w(X) = \sum_{x \in X} w(x)$. In [4], the special case $\eta = 0.5$ was considered. Minimizing the maximum delay would lead to different choices of λ_x.

Setting λ_x according to (2) for every edge (u, x) with $|\delta^+(u)| = 2$ and $\lambda_x = 0$ elsewhere, we have the following delay model, which we assume for (1) from now on:

$$\text{delay}_T(r, t) = \sum_{e=(u,v) \in E(T_{[r,t]})} (d(e) + \lambda_v d_{bif}) \,. \quad (3)$$

The parameter d_{bif} is computed similarly to [4] by taking an optimally spaced uniform repeater chain. Then, d_{bif} is the delay increase when adding the input capacitance in the middle of a single net, minimizing over all layers and wire types.

A. Our Contributions

Our contributions are the following:

1) A new randomized $\mathcal{O}(\log t)$-approximation algorithm for cost-distance Steiner trees with bifurcation penalties (Section II). Its running time of $\mathcal{O}(t(n \log n + m))$ improves by a factor of t over [17].
2) Multiple enhancements to improve its practical performance (Section III).
3) Experimental comparisons with the state-of the art heuristics, demonstrating a significant benefit of our method in timing-constrained global routing (Section IV).

II. A Fast Approximation Algorithm

Our algorithm (Algorithm 1) works similar to Kruskal's algorithm for spanning trees [16]. It builds a Steiner tree by iteratively merging components. In each iteration i, we propagate distance labels from all sinks simultaneously but with a special sink-specific distance function (see Theorem 1) until a permanent label reaches another terminal. This determines the vertices u and v in Line 3, which are joined through a new Steiner point s that replaces u and v in the terminal set S_{i+1} of the next iteration and has the sum of their delay weights as its own weight. Note that the component containing the root vertex r_i might also be merged. In this case, s is inserted at the position $\pi(r_i)$ and serves as the root r_{i+1} in the next iteration (line 5). If two sinks are merged, we select the position $\pi(s)$ randomly proportional to their delay weight (line 7). In the end, the inserted Steiner vertices are the bifurcations of the tree.

A key difference to [16] is the cost function. In [17], the main algorithm uses for each pair u, v of sinks an individual distance function $c(e) + 2w(u)w(v)/(w(u) + w(v)) \cdot d(e)$ $(e \in E(G))$. It combines costs and weighted delays. Computing shortest paths between each pair of terminals with an individual cost function would be prohibitively slow in a large global routing graph. We use one distance function

$$l_u(e) \coloneqq c(e) + w(u) \cdot d(e) \quad (e \in E(G)) \quad (4)$$

per sink u as in [5]. While the algorithms in [5], [17] merge iteratively based on a matching involving (a constant fraction of) all terminals, our algorithm allows the newly inserted Steiner vertex to be merged in the next iteration, before any other merges take place.

We address the bifurcation delay penalties as follows. Let

$$\beta(w, w') = d_{bif} \left(\eta \max(w, w') + (1 - \eta) \min(w, w') \right)$$

Algorithm 1: Cost-Distance Algorithm

Input: An instance $(G, S, r, w, c, d, d_{bif}, \eta)$ of the Cost-Distance Steiner tree problem
Output: An embedded Steiner tree (T, π) on S with root r.

1 $T \leftarrow (S \cup \{r\}, \emptyset)$, $i \leftarrow 0$, $S_0 \leftarrow S$, $r_0 \leftarrow r$.
2 **while** $|S_i| > 0$ **do**
3 Find the nodes $u \in S_i$, $v \in S_i \cup \{r_i\}$ minimizing $L(u, v)$ (see (5)& Theorem 1).
4 **if** $v = r_i$ **then**
5 Add a new Steiner vertex s placed at $\pi(s) = \pi(r)$, $r_{i+1} \leftarrow s$, $w(s) \leftarrow w(u)$, $S_{i+1} \leftarrow S_i \setminus \{u\}$.
6 **else**
7 Randomly select a position $p \in \{\pi(u), \pi(v)\}$ with probabilities proportional to their delay weight. Add a new Steiner vertex s at $\pi(s) = p$ with $w(s) \leftarrow w(u) + w(v)$, $S_{i+1} \leftarrow S_i \cup \{s\} \setminus \{u, v\}$.
8 Add the u-s-v connection corresponding to the path underlying $L(u, v)$ to T.
9 $i \leftarrow i + 1$.
10 Return the embedded tree (T, π).

denote the minimum possible weighted delay penalty when merging two sinks with delay weights w and w'. Then, the weighted bifurcation delay penalty for merging $u \in S_i$ and $v \in S_i \cup \{r_i\}$ is

$$b(u, v) \coloneqq \begin{cases} \beta(w(u), w(v)) & (v \in S_i) \\ \beta(w(u), w(S_i \setminus \{u\})) & (v = r_i) \end{cases}.$$

In line 3 of Algorithm 1, the pair u, v minimizing

$$L(u, v) \coloneqq \begin{cases} \text{dist}_{G, c + \min\{w(u), w(v)\} \cdot d}(u, v) + b(u, v) & (v \in S_i) \\ \text{dist}_{G, c + w(u) \cdot d}(u, r_i) + b(u, r_i) & (v = r_i) \end{cases} \quad (5)$$

is chosen. By $\text{dist}_{\Gamma, \gamma}(x, y)$ we denote the shortest path distance between x and y in the graph Γ w.r.t. edge lengths γ. $L(r_i, v) \coloneqq L(v, r_i)$ completes the definition of L. We can still prove an approximation factor of $\mathcal{O}(\log t)$. In addition, our algorithm is substantially faster than [5], [17].

A. An Example

Figure 3 visualizes the course of the algorithm for a set of 5 sinks, the black dots in the first picture. The size of the dots represents their delay weight. We call the elements of S_i in iteration i the *active* vertices (marked blue). In iteration $i = 0$ of the algorithm, $S_0 = S$, thus all sinks are active, and we start Dijkstra searches (areas inside blue circles) from all sinks using their individual cost functions. The larger the delay weight, the slower the circle grows. The terminal u is the first one to find a connection (to v).

One of $\pi(u)$ and $\pi(v)$ is now chosen randomly with probability proportional to its delay weight to place the Steiner vertex s with weight $w(s) \coloneqq w(u) + w(v)$. The random choice makes it more likely for the heavier weighted terminal to be reachable from r without a detour. Here, the more likely vertex v was chosen.

The shortest u-s-v path regarding l_u (red) is added to T. The terminals u and v are deactivated and s is activated. In the next iteration $i = 1$, we started a new Dijkstra search from s w.r.t. l_s. In this iteration the shortest connection is a root connection, so we do not make a random choice for our Steiner vertex and just deactivate the sink. We continue connecting vertices and decreasing the number of active vertices until all sinks are connected.

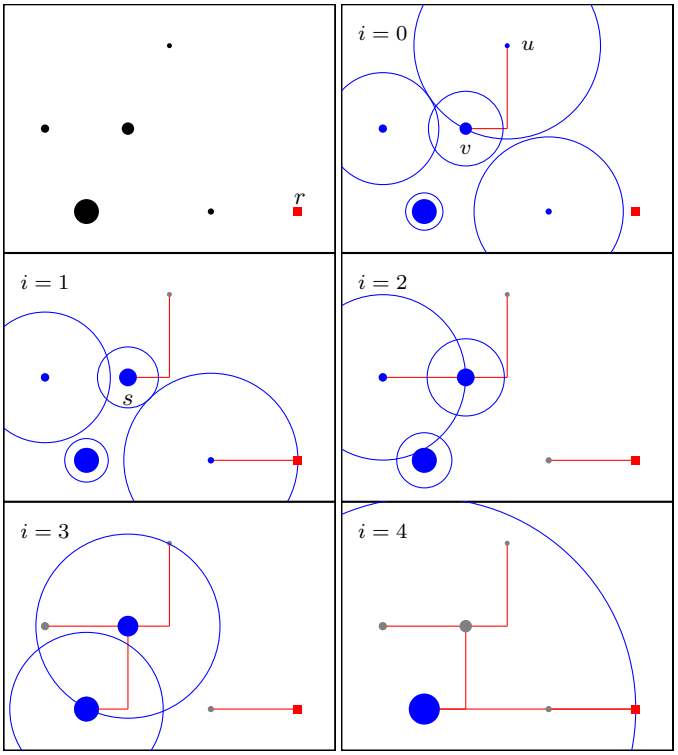


Fig. 3. Cost-distance algorithm example. Root in red, the sinks S in black. For simplicity the labeled vertices in global routing graph are visualized as Euclidean balls.

B. Theoretical Properties

We first analyze the running time of the algorithm.

Theorem 1. *The cost-distance algorithm can be implemented with a running time of $\mathcal{O}(t(n \log n + m))$.*

Proof. The total running time is dominated by the shortest path computations in Step 3. We start a Dijkstra search from each sink $u \in S_i$ simultaneously using the individual distance function l_u.

Whenever we enter a vertex $v \in S_i \cup \{r_i\} \setminus \{u\}$, we add the optimally balanced weighted node delay $b(u, v)$. If such a vertex gets permanently labeled, we have found a pair u, v minimizing $L(u, v)$. After adding the connection, we start a Dijkstra search from the newly inserted Steiner vertex s. Dijkstra labels from other terminals remain valid in the next iteration. As we insert $|S| - 1$ Steiner vertices, we have $2|S| - 1$ path searches, implementing shortest path searches with Fibonacci heaps [10] yields the claimed running time. □

We now prove the approximation guarantee of our algorithm. Throughout, let OPT denote the cost according to (1) and (3) of an optimum solution T^* to the original instance. The challenge lies in bounding the incremental cost increase from each iteration. Similar to [17], we show that there is a cheap matching of terminals w.r.t. weights L (Lemma 4) which covers roughly the price increase of $\frac{1}{4}$ of the iterations (Theorem 6).

In iteration i, consider the potential

$$D_i := \sum_{s \in S_i} w(s) \text{delay}_{T^*}(r, s).$$

As new Steiner vertices $s \in S_i$ are mapped to a position $\pi(s) = \pi(v')$, $v' \in S \cup \{r\}$, $\text{delay}_{T^*}(r, s) := \text{delay}_{T^*}(r, v')$ is well-defined for all $s \in S_i$.

Lemma 2. *In each iteration i, there is a cost-distance tree for the terminal set $S_i \cup \{r_i\}$ with expected total cost at most*

$$C(T^*) + \mathbb{E}[D_i] \leq \text{OPT}\,.$$

Proof. Suppose that two sinks $u, v \in S_i$ are matched in iteration i. They contribute a delay cost of $w(u) \cdot \text{delay}_{T^*}(r, u) + w(v) \cdot \text{delay}_{T^*}(r, v)$ to D_i in iteration i. The expected contribution of the new Steiner vertex s to D_{i+1} is

$$\sum_{x \in \{u,v\}} \frac{w(x)}{w(u) + w(v)} \cdot w(s) \cdot \text{delay}_{T^*}(r, x)$$
$$= w(u) \cdot \text{delay}_{T^*}(r, u) + w(v) \cdot \text{delay}_{T^*}(r, u).$$

Otherwise, if one of the merged nodes is the root, the new Steiner vertex does not contribute to D_{i+1}. This means that we have $\mathbb{E}[D_{i+1}] \leq \mathbb{E}[D_i]$. Thus, $C(T^*) + \mathbb{E}[D_i] \leq \text{OPT}$ by induction. As T^* can be used as a solution for S_i, we have proven the lemma. □

A matching M in a set X is called *near-perfect* if it leaves at most one element uncovered, i.e. $|X \setminus \cup_{e \in M} e| \leq 1$.

Lemma 3. *Given a bifurcation compatible tree $T = (V, E)$ and a set of nodes $S \subseteq V$, there exists a near-perfect matching of the nodes in S such that the union of the unique paths in T between matched nodes contains each edge and each vertex of the tree at most once.*

Proof. Lemma 4.2 in [17] proves that for an arbitrary tree there is such a matching using each edge at most once. If T is bifurcation compatible, this implies that each vertex is contained in at most one path. □

Lemma 4. *In each iteration i, there exists a near-perfect matching M of $S_i \cup \{r_i\}$ with $\mathbb{E}[L(M)] \leq \text{OPT}$.*

Proof. By Lemma 3, there is a near-perfect matching M in $S_i \cup \{r_i\}$ such that the union of the unique paths in T^* between matched nodes contains each edge and each vertex at most once.

For technical reasons, we extend w to $w(r_i)$ with an arbitrary constant. Given $\{u, v\} \in M$, we may assume w.l.o.g. $w(u) \leq w(v)$ or $v = r_i$. Then the cost of M is

$$\sum_{(u,v) \in M} L(u, v)$$
$$\leq \sum_{(u,v) \in M} \Big(\text{dist}_{T^*,c}(u, v) + w(u)\, \text{dist}_{T^*,d}(u, v) + b(u, v) \Big)$$
$$\leq C(T^*) + \sum_{(u,v) \in M} \Big(w(u)\, \text{dist}_{T^*,d}(r, u) + w(v)\, \text{dist}_{T^*,d}(r, v) + b(u, v) \Big) \leq C(T^*) + D_i,$$

where the first inequality holds due to Lemma 3. The next inequality holds because we added twice the distance w.r.t d from r to the u-v path in T^* and $w(u) \leq w(v)$. The last inequality holds because $b(u, v)$ uses the optimum delay penalty distribution, and the unique paths defined by the matching use each vertex exactly once. In contrast, D_i uses a penalty distribution, where penalties can be counted multiple times. Observe that the $b(u, v)$ values are accounted for in D_i either at the unique least common ancestor (w.r.t. the root r) or along the vertices of an r-u-path.

By Lemma 2, $\mathbb{E}[C(T^*) + D_i] \leq \text{OPT}$. □

Lemma 5. *When the algorithm merges $u \in S_i$ and $v \in S_i \cup \{r_i\}$ in iteration i, $L(u, v)$ reflects at least half of the expected objective cost increase in this iteration.*

Proof. $L(u,v)$ reflects the correct connection cost and bifurcation penalty $b(u,v)$. Let P be the underlying path reflecting the embedding into G. If $v \neq r_i$, the vertex $x \in \{u,v\}$ is chosen with a probability of $w(x)/(w(u)+w(v))$. Thus, the expected delay cost w.r.t. d is

$$\frac{2w(u)w(v)}{w(u)+w(v)} \operatorname{dist}_{P,d}(u,v) \leq 2\min\{w(u),w(v)\} \operatorname{dist}_{P,d}(u,v)$$

and

$$\begin{aligned} &2\min\{w(u),w(v)\} \operatorname{dist}_{P,d}(u,v) + \operatorname{dist}_{P,c}(u,v) + b(u,v) \\ \leq\ &2L(u,v) \end{aligned}$$

If $v = r_i$, the (deterministic) cost is $L(u,v)$. □

Theorem 6. *The cost-distance algorithm computes a Steiner tree with expected objective value at most $\mathcal{O}(\log t) \cdot \mathrm{OPT}$.*

Proof. Consider a minimum weight near-perfect matching M in $(S \cup \{r\})$. Order the matching edges $M = \{\{u_1,v_1\},\dots,\{u_{\lfloor t/2 \rfloor},v_{\lfloor t/2 \rfloor}\}\}$ in non-decreasing order w.r.t. L.

For the first $\lfloor t/4 \rfloor$ iterations, the i-th (shortest) path from our algorithm will have costs at most $L(u_{2i-1},v_{2i-1})$, because each path can intersect at most two matching edges. Thus, we paid at most half of the cost of an optimal near perfect matching when adding up the $\lfloor t/4 \rfloor$ path costs $L(u,v)$. By Lemmas 4 and 5, this implies a cost of at most OPT for reducing the size of our instance by a factor (almost) $3/4$.

Reducing the size repeatedly, we pay $\mathcal{O}(\log_{4/3} t)$ times OPT overall. □

III. Practical Enhancements

In this section, we present multiple practical enhancements.

A. Discounting Existing Tree Components

The variant from Section II inserts Steiner vertices only at terminal positions, which leads to unnecessary detours. To avoid this, we set $c(e) = 0$ for edges in the connected components of T where the current path search starts and ends, and allow reusing them. We add separate labels for each possible end component a path search reaches so that the zero connection cost edges can only be used to reach that particular end component. This implicitly places Steiner vertices at the points where the path leaves or enters the connected components. We still restart the next path search from one of the randomly chosen terminal positions. This change significantly improves connection costs and preserves the theoretical guarantee if $d_{bif} = 0$.

B. Two-level Heap Structure

As global routing graphs usually satisfy $m \in \mathcal{O}(n)$, we use binary heaps. We use one heap for each sink in S_i and a top-level heap that stores the minimum key from each sink heap. We operate with a single sink heap until the minimum label in the top-level heap is exceeded, and we extract the next minimum globally.

C. Goal-Oriented Path Searches

To speed up the path searches, we can use goal-oriented path searches, a.k.a. A^* searches [12]. This reduces the number of labeling steps. In our implementation, the connection/congestion costs between two vertices are lower bounded using landmark based future costs [11] and the delays are bounded based on L_1-distance and the fastest layer and wire type combination for that distance.

In a path search from a sink u, we minimize the sum of these lower bounds over all targets in $S_i \cup \{r_i\} \setminus \{u\}$. When setting the connection cost of edges in a component to zero as described in III-A, we need to make sure that the future costs stay feasible. We do this by omitting the connection future costs until every vertex of the starting component is permanently labeled and subtracting an upper bound on the connection future cost for the component containing a target $v \in S_i \cup \{r_i\} \setminus \{u\}$.

D. Better Embedding of Steiner Vertices

We can improve the position of the Steiner vertices s. After adding a path P between terminals u and v, we choose the position $\pi(s)$ on P that minimizes

$$c(E(Q)) + (w(u)+w(v))d(E(Q)) + \sum_{y \in \{u,v\}} w(y)d(E(P_{[y,s]})),$$

where Q is a cheapest s-r-path w.r.t. $c + (w(u)+w(v))d$. This corresponds to extending P to an optimum cost-distance tree for the sink set $S = \{u,v\}$. Finding the best Steiner vertex s requires an extra path search to compute Q. Instead, we use future costs to estimate it. Though, we lose the theoretical guarantees, this modification improves the quality in practice.

If the new Steiner vertex s was already permanently labeled w.r.t. l_v, we can use this path to get some further saving.

E. Encourage Root Connections

Lastly, connecting a terminal u to the root r_i early in the algorithm is very expensive because the expected node delay cost $b(u,r_i)$ reflects all delay weights. However, connecting to the root immediately reduces the cost of an optimal tree for S_{i+1}, saving a cost of at least $\eta \cdot d_{bif} \cdot w(u)$ in future iterations. We subtract these cost savings from the node delay costs for root connections to encourage merging with r_i.

IV. Experimental Results

We integrated our cost-distance Steiner tree function as a Steiner tree oracle of a timing-constrained global routing algorithm similar to [13]. Lagrangean relaxation is used to relax global timing and routing constraints. The cost-distance Steiner tree problem (1) arises as the Lagrangean subproblem.

A. Algorithms for Comparison

We compare the new cost-distance Steiner tree algorithm with three alternatives. Each first computes a Steiner topology in the plane, considering total length instead of congestion cost. Then, this tree is embedded optimally into the global routing graph minimizing the cost-distance objective (1) using a Dijkstra-style embedding as described in [13]. The embedding greatly determines the signal speed by choosing layers and wire types.

The first routine just computes a short L_1 Steiner tree and embeds it optimally into the global routing graph.

The second is based on shallow-light Steiner trees [6], [14]. These algorithms start from an approximately minimum-length tree. During a DFS traversal, sinks are reconnected to the root whenever they violate a given delay/distance bound by more than a factor $(1+\varepsilon)$ for an $\varepsilon > 0$. In a reverse DFS traversal, deleted edges may be re-activated to connect former predecessors if that saves cost. In [14] bifurcation penalties were also integrated for $\eta = 0.5$. We (re-)distribute them according to our more flexible model in the initial tree and during the reverse DFS traversal.

Another very popular method is the Prim-Dijkstra algorithm [2], [3]. Here, sinks are iteratively added into the root-component. A sink s and an edge e in the root component are chosen to insert a new Steiner vertex into e connecting s such that a weighted sum of total length and path length to s is minimized. Bifurcation penalties were incorporated into this method for the special case $\eta = 0.5$ [4]. We

TABLE I
AVERAGE COST INCREASE COMPARED TO MINIMUM, WITH $d_{bif} = 0$.

$\|S\|$	#instances	L_1	SL	PD	CD
3-5	527784	**0.40**%	0.41%	0.44%	0.67%
6-14	156053	1.62%	1.58%	**1.54**%	1.57%
15-29	57594	4.70%	3.73%	3.39%	**2.24**%
≥ 30	84033	7.09%	5.38%	4.48%	**1.73**%
all	825464	1.61%	1.37%	1.27%	**1.06**%

TABLE II
AVERAGE COST INCREASE COMPARED TO MINIMUM, WITH $d_{bif} > 0$.

$\|S\|$	#instances	L_1	SL	PD	CD
3-5	741967	0.96%	0.95%	1.20%	**0.65**%
6-14	222419	4.52%	4.32%	3.28%	**1.89**%
15-29	84034	11.37%	9.76%	4.82%	**3.03**%
≥ 30	127861	23.73%	13.39%	7.07%	**3.30**%
all	1176281	4.85%	3.57%	2.49%	**1.34**%

can distribute the delay penalty to the two branches, when selecting the edge of the root component.

Both methods use the globally optimized delay budgets that arise from the resource sharing algorithm [13].

The three methods are well established in academic and commercial design tools and form a solid foundation to compare against. They are denoted as L_1 (L_1-shortest), SL (shallow-light), and PD (Prim-Dijkstra) in the following tables. Our new cost-distance algorithm with the practical enhancements is denoted CD.

We made tests with and without bifurcation delay penalties.

B. Results on Individual Instances

First, we present an apples-to-apples comparisons on identical cost-distance Steiner tree instances as they were generated during timing-constrained global routing on several 5nm designs, where we set $d_{bif} = 0$ in the first set of tests shown in Table I. For each instance, we measured the relative increase in the objective (1) between each algorithm and the best of the 4 algorithms. On the small instances with 3-5 sinks, the L_1 heuristic gives the best results, though results don't vary much between the 4 algorithms. Note that the embedding into the 3D global routing graph is optimum w.r.t. (1) for L_1, SL, and PD. On larger instances with more than 15 sinks, the cost-distance algorithm dominates the other methods.

Table II shows the same results with bifurcation penalties, here the cost-distance algorithm dominates the other methods throughout.

C. Global Routing Results

Next, we compare the impact of the different algorithms on timing-constrained global routing results. Congestion is measured using the ACE [19]. $ACE(x)$ is the average congestion of the $x\%$ most critical global routing edges. We then use $ACE4 := \frac{1}{4}(ACE(.5) + ACE(1) + ACE(2) + ACE(5))$. Note that the most critical 0.5-percentile is implicitly contributing to all four numbers. An $ACE4$ of 93% is usually considered routable, but detailed routing will introduce significant detours and delays if it is above 90%.

Table III gives an overview of the instances. They are a mix of industrial microprocessor and ASIC units. Again, we start with experiments without bifurcation penalties ($d_{bif} = 0$) in Table IV. For each instance and each Steiner method, as well as for the sum/average of all instances it contains a line, and we report the worst slack (WS), total negative slack (TNS), the $ACE4$, the wire length (WL), the via count (Vias), and the wall times on an AMD EPYC 9684X processor using 16 threads. Best results are marked in green. For most instances our new cost-distance Steiner trees yield the best timing (WS & TNS), the best $ACE4$ and the best via count. The topology in the other routines will often force embedding path searches through congested areas, increasing their via count. However, cost-distance trees come with a higher wire length. The running time of all methods are comparable.

Table V shows results with bifurcation penalties. Note that the bifurcation penalties increase the delays and thus decrease slacks. The wire length and via count decrease compared to $d_{bif} = 0$ because the delay prices now weigh stronger compared to congestion. Again the new CD methods outperforms the others in terms of WS, TNS and via count, but has a slightly larger wire length.

TABLE III
INSTANCE PARAMETERS (ALL ARE IN 5NM TECHNOLOGY)

Chip	# nets	# layers	Chip	# nets	# layers
c1	49 734	8	c5	420 131	9
c2	66 500	9	c6	590 060	9
c3	286 619	7	c7	650 127	15
c4	305 094	15	c8	941 271	15

TABLE IV
TIMING-CONSTRAINED GLOBAL ROUTING RESULTS WITH $d_{bif} = 0$.

Chip	Run	WS [ps]	TNS [ps]	ACE4 [%]	WL [m]	Vias [#]	Walltime [h:m:s]
c1	L_1	-49	-1 633	91.21	0.6016	561 140	00:03:24
	SL	-49	-1 544	91.32	0.6073	560 678	00:03:23
	PD	-49	-1 370	91.50	0.6104	558 133	00:05:29
	CD	-49	-1 340	89.36	0.6152	547 240	00:03:30
c2	L_1	-82	-19 774	86.36	1.0730	864 387	00:04:30
	SL	-80	-19 659	85.75	1.0838	863 571	00:04:00
	PD	-80	-19 558	85.81	1.0845	862 924	00:03:34
	CD	-80	-19 512	85.56	1.1033	854 387	00:03:52
c3	L_1	-220	-105 137	92.07	2.0070	2 336 123	00:17:33
	SL	-220	-82 142	92.28	2.0305	2 327 893	00:18:28
	PD	-220	-67 763	92.39	2.0375	2 324 940	00:20:23
	CD	-220	-67 643	91.54	2.0751	2 313 590	00:16:40
c4	L_1	-67	-31 907	90.23	6.5801	4 095 449	00:36:15
	SL	-68	-31 805	90.12	6.6114	4 083 044	00:50:07
	PD	-71	-31 769	90.21	6.6270	4 067 446	00:49:31
	CD	-68	-31 682	89.77	6.8127	4 005 655	00:43:32
c5	L_1	-276	-946 089	86.26	4.4107	4 640 578	00:21:29
	SL	-286	-869 512	86.15	4.4633	4 612 457	00:26:23
	PD	-264	-803 343	85.94	4.4687	4 580 028	00:26:58
	CD	-264	-776 373	85.27	4.6067	4 522 756	00:21:25
c6	L_1	-331	-536 208	85.96	3.6925	4 699 131	00:35:08
	SL	-314	-525 484	86.03	3.7549	4 678 851	00:43:27
	PD	-311	-509 299	86.02	3.7537	4 659 440	00:39:15
	CD	-308	-507 841	85.02	3.8492	4 613 520	00:37:18
c7	L_1	-86	-57 267	87.72	13.1894	7 817 717	01:22:23
	SL	-87	-57 903	87.69	13.2282	7 800 550	01:19:05
	PD	-90	-59 139	87.61	13.2473	7 790 466	01:18:02
	CD	-88	-54 659	87.23	13.4631	7 721 547	01:38:58
c8	L_1	-102	-174 288	91.14	13.5765	12 559 176	02:00:44
	SL	-99	-175 643	91.05	13.6386	12 519 244	02:35:41
	PD	-104	-171 323	90.99	13.6962	12 501 090	02:35:44
	CD	-105	-174 831	90.78	14.3538	12 330 992	02:33:09
all	L_1	-1 214	-1 872 304	88.87	45.1308	37 573 701	05:21:26
	SL	-1 204	-1 763 692	88.80	45.4180	37 446 288	06:20:34
	PD	-1 188	-1 663 564	88.81	45.5253	37 344 467	06:18:56
	CD	-1 181	-1 633 880	88.07	46.8791	36 909 687	06:18:24

V. CONCLUSION

We presented a new and faster approximation algorithm for the cost-distance Steiner tree problem, achieving the best known theoretical

TABLE V
TIMING-CONSTRAINED GLOBAL ROUTING RESULTS WITH $d_{bif} > 0$.

Chip	Run	WS [ps]	TNS [ps]	ACE4 [%]	WL [m]	Vias [#]	Walltime [h:m:s]
c1	L_1	-84	-9 837	92.15	0.5880	536 492	00:03:14
	SL	-64	-6 325	92.33	0.6039	536 398	00:03:11
	PD	-64	-5 937	92.16	0.6051	530 699	00:03:34
	CD	-61	-6 333	91.93	0.6109	526 427	00:03:09
c2	L_1	-105	-28 844	87.76	1.0728	759 747	00:03:41
	SL	-93	-22 320	85.91	1.1032	746 116	00:03:55
	PD	-88	-22 102	86.56	1.0937	733 950	00:03:31
	CD	-97	-22 022	86.07	1.1030	728 092	00:03:50
c3	L_1	-450	-2 751 437	92.33	1.9731	2 131 502	00:17:27
	SL	-315	-2 129 700	92.77	2.0701	2 107 934	00:20:58
	PD	-315	-1 831 312	92.81	2.0624	2 094 655	00:21:02
	CD	-304	-1 864 826	91.49	2.0496	2 085 304	00:17:30
c4	L_1	-84	-147 690	90.95	6.5049	3 645 350	00:31:02
	SL	-75	-135 682	90.81	6.5351	3 612 175	00:44:12
	PD	-65	-101 879	90.73	6.6586	3 533 707	00:34:26
	CD	-64	-91 618	90.62	6.7194	3 518 335	00:32:09
c5	L_1	-533	-3 654 628	87.49	4.3901	4 300 895	00:23:04
	SL	-404	-2 358 983	87.03	4.5889	4 170 442	00:32:23
	PD	-404	-2 177 493	86.75	4.5325	4 143 083	00:25:34
	CD	-415	-2 141 300	86.48	4.5952	4 093 042	00:24:21
c6	L_1	-756	-3 978 380	88.12	3.6710	4 348 194	00:35:49
	SL	-722	-3 160 751	88.94	3.8832	4 237 414	00:46:15
	PD	-698	-2 750 994	88.54	3.8960	4 185 778	00:44:29
	CD	-685	-2 703 373	87.17	3.8637	4 169 492	00:39:26
c7	L_1	-144	-339 184	88.80	13.0465	6 779 085	01:14:58
	SL	-123	-267 361	88.92	13.0961	6 740 822	01:10:15
	PD	-119	-217 997	88.72	13.2082	6 653 426	01:11:42
	CD	-120	-194 336	89.12	13.2902	6 647 930	01:34:00
c8	L_1	-197	-1 093 449	91.46	13.4100	11 450 165	02:11:43
	SL	-154	-985 353	91.59	13.5111	11 405 990	02:54:40
	PD	-114	-804 198	91.15	13.8756	11 196 187	02:44:06
	CD	-115	-804 080	91.14	14.1136	11 138 570	02:31:19
all	L_1	-2 353	-12 003 449	89.88	44.6564	33 951 430	05:20:58
	SL	-1 949	-9 066 474	89.79	45.3916	33 557 291	06:35:49
	PD	-1 867	-7 911 912	89.68	45.9321	33 071 485	06:08:24
	CD	-1 861	-7 827 888	89.25	46.3456	32 907 192	06:05:44

approximation factor of $\mathcal{O}(\log t)$. It arises as a sub-problem in timing-constrained global routing with a linear delay model (before buffering). We adopted it for better practical performance. Our experimental results show improvements over state-of-the-art algorithms on large instances and in presence of bifurcation penalties. It leads to the overall best timing and congestion results in timing-constrained global routing.

ACKNOWLEDGEMENTS

We thank Paula Heinz who implemented a preliminary version based on [17]. We reused parts her code.

REFERENCES

[1] Albrecht, C., Kahng, A.B., Liu, B., Mandoiu, I.I. and Zelikovsky, A.Z. *On the skew-bounded minimum-buffer routing tree problem*. IEEE TCAD 22(7), 937–945, 2003.

[2] Alpert, C.J., Hu, T.C., Huang, J.H., Kahng, A.B. and Karger, D.: 1995. *Prim-Dijkstra tradeoffs for improved performance-driven routing tree design*. IEEE TCAD 14(7), 890–896, 1995.

[3] Alpert, C.J., Chow, W.K., Han, K., Kahng, A.B., Li, Z., Liu, D. and Venkatesh, S.: *Prim-Dijkstra revisited: Achieving superior timing-driven routing trees*. Proc. ISPD'18, 10–17, 2018.

[4] Bartoschek, C., Held, S., Maßberg, J., Rautenbach, D. and Vygen, J.: *Efficient generation of short and fast repeater tree topologies*. Proc. ISPD'06, 120–127, 2006.

[5] Chekuri, C., Khanna, S., and Naor, J.: *A deterministic algorithm for the cost-distance problem*. Proc. ACM-SIAM SODA'01, 232–233, 2001.

[6] Chen, G. and Young, E.F.: *Salt: provably good routing topology by a novel steiner shallow-light tree algorithm*. IEEE TCAD 39(6), 1217–1230, 2019.

[7] Chuzhoy, J., Gupta, A., Naor, J., and Sinha, A.: *On the approximability of some network design problems*. ACM Transactions on Algorithms 4 (2), article 23, 2008.

[8] Daboul, S., Held, S., Natura, B., and Rotter, D.: *Global interconnect optimization*. ACM TODAES 28(5), 1–24, 2023.

[9] Foos, J., Held, S., and Spitzley, Y.K.D.: *Tighter Approximation for the Uniform Cost-Distance Steiner Tree Problem*. Proc. APPROX/RANDOM'23, article 19, 2023.

[10] Fredman, M.L. and Tarjan, R.E.: *Fibonacci heaps and their uses in improved network optimization algorithms*. Journal of the ACM, 34(3), 596–615, 1987.

[11] Goldberg, A.V., Harrelson, C.: *Computing the shortest path: A* search meets graph theory*. Proc. ACM-SIAM SODA'05, pp. 156–165, 2005.

[12] Hart, P.E., Nilsson, N.J. and Raphael, B.: *A formal basis for the heuristic determination of minimum cost paths*. IEEE TSSC 4(2), 100–107, 1968.

[13] Held, S., Müller, D., Rotter, D., Scheifele, R., Traub, V. and Vygen, J.: *Global Routing With Timing Constraints*. IEEE TCAD 37(2), 406–419, 2018.

[14] Held, S. and Rotter, D.: *Shallow-Light Steiner Arborescences with Vertex Delays*. Proc. IPCO 2013, 229–241.

[15] Hu, J., Sapatnekar, S.S.: *A timing-constrained simultaneous global routing algorithm*. IEEE TCAD 21, 1025–1036, 2002.

[16] Kruskal, J.B.: *On the shortest spanning subtree of a graph and the traveling salesman problem*. Proc. AMS 7(1), 48–50, 1956.

[17] Meyerson, A., Munagala, K., and Plotkin, S.: *Cost-distance: Two metric network design*. SIAM Journal on Computing 38(4), 1648–1659, 2008.

[18] Wei, Y., Li, Z., Sze, C., Hu, S., Alpert, C.J. and Sapatnekar, S.S. *CATALYST: Planning layer directives for effective design closure*. Proc. DATE'13, 1873–1878, 2013.

[19] Wei, Y., Sze, C., Viswanathan, N., Li, Z., Alpert, C.J., Reddy, L., Huber, A.D., Téllez, G.E., Keller, D. and Sapatnekar, S.S.: *Techniques for scalable and effective routability evaluation*. ACM TODAES 19(2), 17:1-17:37, 2014.

[20] Yan, J.-T., Lee, C.-F., Chen, Y.-H.: *Multilevel timing- constrained full-chip routing in hierarchical quad-grid model*. Proc. ISCAS, 5439–5442,2006.

[21] Yang, J. and Kim, T.: *Improving Timing Quality Through Net Topology Optimization in Global Routing*. Proc. 37th SOCC, 1–6, 2024.

[22] Yang, L., Sun, G. and Ding, H.: *Towards Timing-Driven Routing: An Efficient Learning Based Geometric Approach*. Proc. ICCAD'23, 1–9, 2023.

ABC-FHE: A Resource-Efficient Accelerator Enabling Bootstrappable Parameters for Client-Side Fully Homomorphic Encryption

1st Given Name Surname
dept. name of organization (of Aff.)

2nd Given Name Surname
dept. name of organization (of Aff.)

3rd Given Name Surname
dept. name of organization (of Aff.)

***Abstract*—As the demand for privacy-preserving computation continues to grow, fully homomorphic encryption (FHE)—which enables continuous computation on encrypted data—has become a critical solution. However, its adoption is hindered by significant computational overhead, requiring 10000-fold more computation compared to plaintext processing. Recent advancements in FHE accelerators have successfully improved server-side performance, but client-side computations remain a bottleneck, particularly under bootstrappable parameter configurations, which involve combinations of encoding, encrypt, decoding, and decrypt for large-sized parameters. To address this challenge, we propose ABC-FHE, an area- and power-efficient FHE accelerator that supports bootstrappable parameters on the client side. ABC-FHE employs a streaming architecture to maximize performance density, minimize area usage, and reduce off-chip memory access. Key innovations include a reconfigurable Fourier engine capable of switching between NTT and FFT modes. Additionally, an on-chip pseudo-random number generator and a unified on-the-fly twiddle factor generator significantly reduce memory demands, while optimized task scheduling enhances the CKKS client-side processing, achieving reduced latency. Overall, ABC-FHE occupies a die area of 28.638 mm^2 and consumes 5.654 W of power in 28 nm technology. It delivers significant performance improvements, achieving a 1112× speed-up in encoding and encryption execution time compared to a CPU, and 214× over the state-of-the-art client-side accelerator. For decoding and decryption, it achieves a 963× speed-up over the CPU and 82× over the state-of-the-art accelerator.**

***Index Terms*—accelerator, bootstrappable parameter, client-side, fully homomorphic encryption**

I. Introduction

In recent years, personal data from edge devices, such as PCs and mobile phones, has been increasingly sent to centralized servers for advanced computation with the paradigm of cloud computing [32]. This trend is evident in real-time AI applications that have excessive computational requirements, including chatbot [6], voice assistant [18], and image recognition [23]. However, this raises privacy concerns, as users should share their personal data on cloud servers. Fully homomorphic encryption (FHE) is considered the solution by allowing computations on data while it remains encrypted, thereby preserving privacy.

While FHE provides a solution for privacy-preserving computations, it also introduces substantial computational challenges [24]. Specifically, in the CKKS [7] scheme, a popular choice for AI-based applications, the ciphertext is a high-degree polynomial (ranging from degree 2^{14} to 2^{16}) with each coefficient represented as a high-precision integer of hundreds to thousands of bits. These coefficients are decomposed in the residue number system (RNS), reducing them to approximately 50–60 bits and enabling more efficient arithmetic operations. To accelerate these computations, the number theoretic

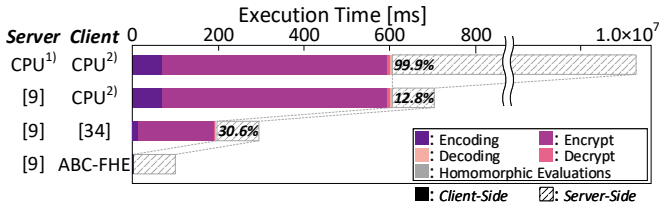


Fig. 1. Execution time breakdown of overall client-side and server-side operations.

transform (NTT) is often used, reducing the complexity of polynomial multiplication from N^2 to $Nlog(N)$. Despite this algorithmic enhancement, workloads within the FHE domain remain up to 10,000× slower than their plaintext counterparts due to the inherent computational complexity, emphasizing the need for dedicated hardware accelerators to achieve the demanding performance of modern applications [17].

Among hardware accelerators, the majority have focused on achieving advancements on the server-side [9], [19], [20], [28], [29]. However, the need for client-side accelerators has become increasingly evident for two main reasons. First, server-side computation is no longer the primary bottleneck due to drastically reduced computation times by server-side FHE accelerators. For example, as shown in Figure 1, executing FHE operations on the ResNet20 model reveals that encryption and decryption tasks on the client-side account for 69.4% of the total execution time using a state-of-the-art (SOTA) client-side accelerator [34], whereas computations on the server-side account for only 30.6% with a SOTA server-side ASIC accelerator [9]. Second, client-side computations involve distinct workload characteristics, requiring both fast Fourier transform (FFT) and NTT for processing. These computations depend on floating-point and integer operations, respectively, and are typically performed in a non-repetitive manner, presenting unique challenges compared to server-side workloads.

Several prior works have explored client-side FHE accelerators [3], [10], [22], [34], but they suffer from several limitations. First, these accelerators are constrained to small FHE parameters (e.g., $N = 2^{13}$), rendering them incapable of supporting bootstrapping operations that require larger polynomial degrees, typically at least $N = 2^{14}$ and often extending to $N = 2^{16}$. Second, prior works using non-streaming architectures encounter severe DRAM bandwidth limitations. Even if throughput is increased to accelerate computations, the resulting larger data output per cycle exceeds the available DRAM bandwidth, leading to latency overhead rather than performance gains. Third, the SOTA work [34] intensifies the DRAM bandwidth bottleneck by directly fetching parameters

from DRAM, further increasing memory bandwidth demands and exacerbating latency overhead.

To address these issues, we propose ABC-FHE, a resource-efficient FHE accelerator specifically designed to support bootstrappable parameters for client-side applications. First, it is tailored to accommodate bootstrappable parameters, aligning with the practical requirements of FHE deployment. Second, it adopts a streaming architecture, addressing memory bottlenecks associated with large bootstrappable parameters with efficient memory utilization and seamless operation. Third, it incorporates an on-chip pseudo-random number generator (PRNG) and an on-the-fly twiddle factor generator (OTF TF Gen) to generate random values and twiddle factors directly on-chip, reducing reliance on external memory. Furthermore, it introduces methods to minimize the area overhead caused by the structural characteristics of the streaming architecture.

To summarize, we make the following contributions:

- We analyze the workload of the client-side CKKS scheme, identifying distinct workload characteristics and imbalances between encryption-related tasks (IFFT + NTT) and decryption-related tasks (FFT + INTT).
- To meet latency requirements while accommodating diverse workload characteristics, we propose a streaming architecture with a reconfigurable compute engine that supports both floating-point complex-number-based I/FFT and integer-based I/NTT operations.
- To further optimize area efficiency, we introduce the unified on-the-fly twiddle factor generator (OTF TF Gen), enabling the NTT/FFT engine to share a single hardware unit for twiddle factor generation. Additionally, we provide a methodology for selecting optimal prime numbers to facilitate hardware-friendly modular multiplication.
- Overall, ABC-FHE achieves a speed-up of up to 1112× for encoding and encryption and up to 963× for decoding and decryption compared to a CPU, while delivering up to 214× and 82× improvements, respectively, over SOTA accelerators.

II. BACKGROUND

For the remainder of this paper, we use the following notations: In representation of polynomial, n denotes the degree, N represents the polynomial degree, Q refers to the prime number, and P in the FFT architecture indicates the number of parallelism lanes.

A. Fully Homomorphic Encryption

FHE achieves security by injecting error into the computation process, requiring a specialized operation called bootstrapping to reset the accumulated error during consecutive computations. In the CKKS scheme, this error is managed as "levels" which decrease as error accumulates through operations. Bootstrapping restores the levels and reduces the error; however, the process also consumes levels as part of its execution. To minimize the frequency of this computationally intensive bootstrapping, higher levels are necessary. Achieving a high level while ensuring the 128-bit security standard,

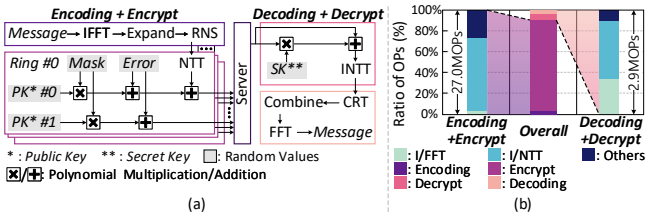


Fig. 2. Workload analysis of CKKS client-side operations: (a) Operational flow and (b) Ratio of operations for 12 level encoding/encryption and 1 level decoding/decryption supporting polynomial degree 2^{16}.

which is considered secure in modern cryptographic systems, requires large polynomial degrees ranging from 2^{14} to 2^{16} [5].

B. Fourier-like Transform

In FHE, polynomial multiplication is required, which has a complexity of N^2. By transforming from the time domain to the frequency domain using fast transforms, this complexity can be reduced to $N\log(N)$. Two types of transformations are commonly used in FHE: NTT and FFT, which perform complex-number and modular computations, respectively. Both transforms share the same fundamental equation but differ in their respective weights, known as the twiddle factors. Specifically, NTT operates within a prime modular ring (Q), while FFT operates on the unit circle in the complex plane. The equation for Fourier-like transform is as follows :

$$A[k] = \sum_{i=0}^{N-1} a(i) \times W_N^{ki} \quad \cdots (1)$$

where W_N represents the twiddle factor for each transformation.

C. Hardware for Fourier-like Transform

Even with complexity optimization through FFT, N can reach up to 2^{16} in CKKS, which still requires high computational capacity. Many prior works have sought to support the FFT algorithm through dedicated hardware [2], [12], [16]. There are two main approaches to optimizing FFT architecture.

Spatial Approach. To maximize throughput, increasing the number of computation lanes (P) is an obvious approach, as it accelerates processing at the cost of additional area. However, on the client-side, where careful external memory accesses (EMA) handling is required, generating a large amount of output data per cycle can lead to additional stalls. Therefore, the number of lanes must be chosen carefully, as a larger number is not always optimal.

Temporal Approach. The limitation on the number of lanes necessitates an architecture with multiple stages. Each stage operates in a pipelined manner, allowing the Fourier-like transform to achieve constant throughput per cycle, eliminating any latency disparities compared to non-pipelined designs. This makes a pipelined NTT architecture more suitable for client-side hardware.

D. Workload Characteristic of Client-Side FHE

Figure 2 highlights two key characteristics of CKKS client-side operations when computing a single ciphertext. First,

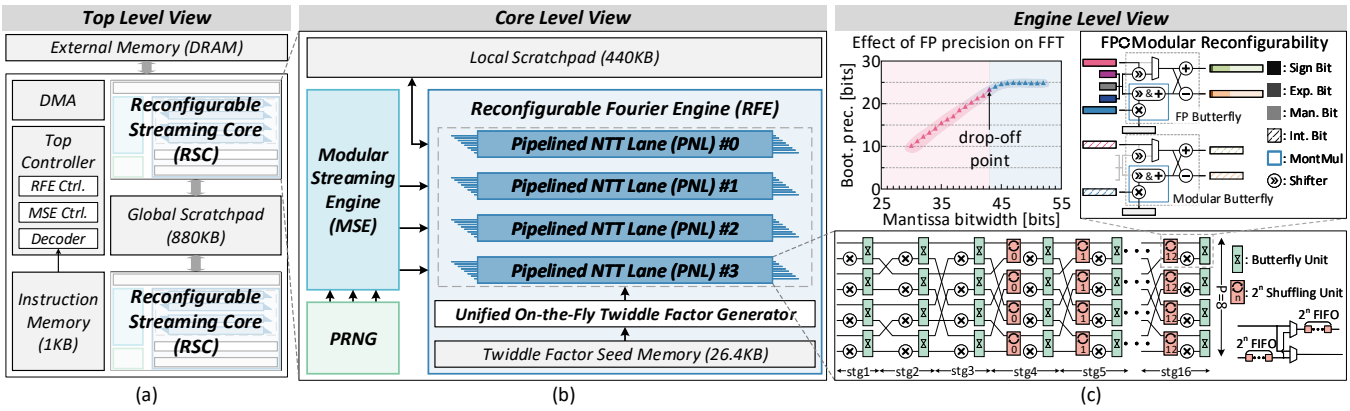


Fig. 3. Overall architecture of ABC-FHE (a) Top level view (b) Core level view (c) Engine level view.

as shown in Figure 2 (a), encoding and encryption require simultaneous support for IFFT and NTT to convert messages into ciphertexts, whereas decoding and decryption require FFT and INTT for reversing ciphertexts into messages. Second, as illustrated in Figure 2 (b), the number of operations for encoding and encryption is nearly ten times greater than for decoding and decryption. This imbalance underscores the inefficiency of implementing separate modules for encryption and decryption.

III. ABC-FHE Architecture

To address the limitations of prior works that are restricted to small parameters [3], [10], [22], [34], we designed an accelerator capable of efficiently handling larger parameters. In resource-constrained client-side environments, simply increasing throughput without regard for area and resource limitations can lead to memory bottlenecks or an unrealistic demand for on-chip memory. Thus, ABC-FHE employs a streaming architecture to maintain consistent throughput while optimizing area, addressing these challenges effectively.

Figure 3 (a) shows the overall architecture of ABC-FHE. At the top level, two homogeneous reconfigurable streaming cores (RSC) enable the streaming-based design to stably handle either message encoding or ciphertext decoding. The reconfigurable nature of RSC allows for three operational modes: doubling the throughput for encrypt, doubling the throughput for decrypt, or simultaneously performing encrypt and decrypt. Additionally, a global scratchpad is included to fetch messages or ciphertext from external memory, ensuring continuous data flow to the two cores.

Figure 3 (b) presents the core-level architecture of RSC, optimized for efficient streaming operations across its three operational modes. At its core, RSC incorporates two primary engines to enable seamless functionality: the reconfigurable Fourier engine (RFE) and the modular streaming engine (MSE). The RFE, equipped with four pipelined NTT lanes (PNLs), supports both I/NTT and I/FFT computations to maximize throughput for Fourier-like transforms. Complementing this, the MSE handles additional SIMD-like operations, including RNS, Chinese Remainder Theorem (CRT) computations for inverse RNS, and element-wise additions and multiplications. To minimize EMA, the architecture emphasizes on-chip value generation. Specifically, an on-chip PRNG produces random values such as masks, errors, and keys for ciphertext generation, eleminating the need to fetch them from external memory. While the OTF TF Gen, shared among the four PNLs, dynamically generates twiddle factors for Fourier-like transforms.

Figure 3 (c) highlights the RFE, the central component of RSC, designed to efficiently support both I/FFT and I/NTT operations through optimized bitwidth configurations. For NTT, recent studies addressed the challenge of substantial bitwidth requirements in CKKS by dividing data into smaller 32–36 bit segments across multiple levels while doubling the number of levels, a solution proven feasible for CKKS [1]. This allows ABC-FHE to adopt a smaller datapath bitwidth, optimizing efficiency. For I/FFT, required precision is measured by bootstrapping precision (Boot. prec.), which indicates the bit precision needed after server-side bootstrapping to maintain model accuracy. Prior research [19] established that a Boot. prec. above 19.29 bits sufficiently supports AI models without significant accuracy degradation. To minimize the FFT datapath, which traditionally relies on the 64-bit floating-point (FP64) data type, we iteratively reduced the floating-point mantissa bitwidth and evaluated Boot. prec. at each step. As shown in the graph in Figure 3 (c), maintaining at least 43 mantissa bits results in a Boot. prec. of 23.39 bits, exceeding the required threshold. Accordingly, we implemented a custom 55-bit floating-point (FP55) format for I/FFT and a 44-bit modular operation for I/NTT, designed in a reconfigurable manner to maximize efficiency for both computation types.

IV. Microarchitecture

A. Reconfigurable Fourier Engine (RFE)

ABC-FHE employs the NTT algorithm to perform polynomial multiplication. To achieve acceleration through increased throughput while maintaining a consistent rate, a pipelined NTT design with an multi-path delay commutator (MDC)-based backbone [16] (P=8) was chosen. However, the increased number of butterfly units required for this design results in larger overall area. To address this, ABC-FHE introduces three optimizations: (1) twiddle factor scheduling

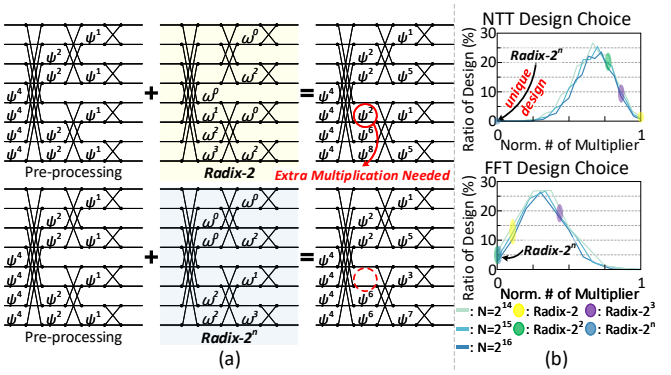


Fig. 4. (a) Twiddle factor scheduling in NTT signal flow graph (b) Distribution of multiplier counts across possible NTT/FFT design configurations.

to reduce the total number of multipliers, (2) an algorithm-hardware co-designed Montgomery multiplier to minimize multiplier area, and (3) leveraging reconfigurability among PNLs to support both NTT and FFT operations.

Twiddle Factor Scheduling. Research on pipelined architectures for the FFT has been extensive [11], [13], [15], with a particular focus on minimizing the area of computational units. In FFT, efforts to optimize area include treating the multiplication between twiddle factors and inputs through rotators, enabling the identification of optimal rotator conditions in pipelined architectures. However, in the NTT, all multipliers are unified as modular multipliers, unlike the FFT approach. This makes it essential to reduce the number of computational units. Additionally, to simplify polynomial operations in NTT, the nega-cyclic property is employed, which requires extra pre-processing and post-processing steps. The equations for pre-processing and post-processing are shown below.

$$A[k] = \sum_{n=0}^{N-1} a(n) \times W_N^{kn} \times \psi_{2N}^{n} \pmod{Q} \cdots (2)$$

$$a[n] = N^{-1} \sum_{k=0}^{N-1} A[k] \times W_N^{-kn} \times \psi_{2N}^{-k} \pmod{Q} \cdots (3)$$

$$\text{where} \quad \psi_{2N}^2 \equiv W_N \pmod{Q}$$

The NTT algorithm can be optimized by merging pre-processing and post-processing operations with twiddle factors, as demonstrated in [30] and [27]. This merging technique eliminates the need for additional multipliers by applying a consistent pattern of twiddle factor operations across stages in the signal flow graph. By aligning the arrangement of twiddle factors with the pre-/post-processing pattern, the merging technique can be seamlessly implemented without extra computational units. Consequently, the minimum number of multipliers required for a pipelined NTT architecture is theoretically $P/2 \times \log_2 N$. Figure 4 (a) shows the distribution of twiddle factors in the signal flow graph using an 8-point example. For instance, with a radix-2 design, pre-processing results in 13 twiddle factor multiplications, while a radix-2^n design reduces this to 12 multiplications. As the polynomial degree increases, the design choice significantly impacts the total number of multiplications required.

To further optimize multiplier usage, we analyzed a range of design configurations, including less conventional options, to evaluate their overheads. Figure 4 (b) compares the overheads of different design options for NTT and FFT. Our analysis revealed that only radix-2^n designs maintain the consistent twiddle factor pattern required for efficient pipelined NTT implementation, making them the most suitable choice. Consequently, we adopted radix-2^n designs for both NTT and FFT operations, highlighting that our radix-2^n configuration achieves a 29.7% and 22.3% reduction in number of multipliers compared to widely used radix-2 and radix-2^2 designs in NTT, respectively.

NTT-Friendly Montgomery Multiplier. Modular multiplication, which involves both multiplication and division by a specific number, is inherently expensive. To address this, both algorithmic optimizations [4], [25] and hardware implementations have been explored [8], [14], [21], [26]. Among the common approaches, Barrett multiplication approximates division with a multiplication to simplify the operation, while Montgomery multiplication first transforms the input into the Montgomery domain via pre-processing, performs modular reduction using simpler arithmetic, and then converts it back to the original domain via post-processing. Montgomery reduction efficiently reduces modular arithmetic computations in the Montgomery domain. It operates on two integers a and b ($a, b < Q$), with a modulus Q of p_bw-bit bitwidth and a coprime R ($R \geq Q$).

$$T = a \times b \cdots (4)$$

$$m \equiv ((T \pmod{R}) \times QInv) \pmod{R} \cdots (5)$$

$$t \equiv (T - m \times Q) \pmod{R} \cdots (6)$$

$$t = \begin{cases} t + Q, & \text{if } t < 0 \\ t, & \text{otherwise} \end{cases} \cdots (7)$$

Montgomery multiplication is typically the simplest and most efficient. However, Montgomery reduction inherently requires three multipliers, introducing significant overhead, particularly in FHE applications with large bitwidths. To address this issue, we optimized the Montgomery reduction algorithm by modifying the computation of $QInv$ to be processed using a shift-and-add approach.

Before presenting the optimized algorithm, we define the NTT-friendly prime number and Euler's theorem as follows:

$$Q = 2^{p_bw} + k \cdot 2^{n+1} + 1 \cdots (8)$$

$$QInv \equiv Q^{2^{n-1}-1} \pmod{2^n} \cdots (9)$$

Equation (8) represents the expression of an NTT-friendly prime, where k is an arbitrary value. Equation (9) uses Euler's theorem to define $QInv$. $\equiv$ in equation (10) and (11) represent modular equivalence with 2^n. Substituting the expression for Q from Equation (8) into Equation (9) yields the following:

$$QInv \equiv \sum_{i=0}^{2^{n-1}-1} \binom{2^{n-1}-1}{i} \cdot (2^{p_bw} + k \cdot 2^{n+1})^i \cdots (10)$$

TABLE I
AREA OF MODULAR MULTIPLIER

Algorithm	Area (μm^2)	Pipeline Stages (Cycles)
Vanilla Barrett	35054	4
Vanilla Montgomery	19255	3
NTT-Friendly Montgomery	11328	3

If k satisfies the condition $k \geq 2^{bw/2-1-n}$, with $k = \pm 2^a \pm 2^b \pm 2^c$, $QInv$ can be simplified as follows:

$$QInv \equiv -2^{p_bw} - (\pm 2^a \pm 2^b \pm 2^c) \cdot 2^{n+1} + 1 \quad \cdots (11)$$

By applying this to the Montgomery algorithm, all multiplications except for the initial multiplication can be converted into shift-and-add operations. This significantly reduces the hardware complexity compared to the original implementation, which required three multipliers.

Despite reducing the number of supported primes due to this optimization, the design still sufficiently supports 20-40 encryption levels. For instance, in the case of $N = 2^{16}$, the required 32-36 bit primes amount to a total of 443, which is more than adequate to handle all practical scenarios. Table I further highlights the area and pipeline stages of different modular multipliers (MM), synthesized at 600 MHz using 28 nm process. The NTT-friendly Montgomery MM achieves a 67.7% area reduction compared to the Barrett MM and a 41.2% reduction compared to the vanilla Montgomery MM.

Reconfigurablility among PNLs. To support FFT computations, complex number FP operations are required. These involve calculations such as:

$$(a + bi) \cdot (c + di) = (ac - bd) + i(ad + bc) \quad \cdots (12)$$

which require four FP multiplications. To efficiently handle these operations, the modular multipliers and FP multipliers are designed to operate in a reconfigurable manner. This enables the use of four modular multipliers to support complex number FP multiplication. The same approach is applied to the unified OTF TF Gen, ensuring seamless support for generating twiddle factors.

B. Efficient Memory Handling

ABC-FHE addresses the challenge of minimizing memory requirements for FHE operations on the client-side by generating essential data on-chip, thereby reducing reliance on external memory. For client-side FHE with a polynomial degree of 2^{16}, 44-bit precision, and 24 levels, calculations estimate the need for 16.5 MB of public key storage, 8.25 MB for masks and errors, and an additional 8.25 MB for twiddle factors. Such substantial on-chip memory is impractical for client-side applications, while frequent external memory fetches require high-bandwidth solutions, such as HBM, which are unsuitable for typical client-side environments.

To address these constraints, ABC-FHE incorporates a PRNG and an unified OTF TF Gen to generate data directly on-chip. The PRNG requires only a 128-bit seed on-chip to meet the 128-bit security level, eliminating the need to store extensive precomputed values. The unified OTF TF Gen dynamically generates twiddle factors for each stage of the pipelined NTT lane using a compact twiddle factor seed and step size, requiring approximately 27 KB, which reduces on-chip memory requirements by over 99.9%. This efficient approach not only significantly reduces memory overhead but also enables scalable and practical FHE acceleration for client-side environments.

TABLE II
THE AREA AND POWER BREAKDOWN OF ABC-FHE

Component	Area (mm^2)	Power (W)
4× PNL	10.717	1.397
Unified OTF TF Gen	0.697	0.089
Twiddle Factor Seed Memory	0.046	0.022
MSE	0.787	0.298
PRNG	0.069	0.028
Local Scratchpad	0.658	0.323
RSC	**12.973**	**2.156**
2× RSC	25.946	4.313
Global Scratchpad	2.632	1.290
Top CTRL, DMA, Etc.	0.060	0.051
Total	**28.638**	**5.654**

V. EVALUTATION

A. Hardware Modeling

We synthesize ABC-FHE's functional units, scratchpads, and interconnections using a 28nm process design kit, with a focus on minimizing area and power consumption while maintaining a 600 MHz clock frequency. All functional unit datapaths, including the RFE and MSE, support both 44-bit integer and 55-bit floating-point operations. The global scratchpad, designed as a double-buffered, single-port, multi-bank 256-bit width SRAM with a total capacity of 880 KB, stores messages received from the host and processed ciphertexts sent to the server. Assuming LPDDR5, which is commonly used in client-side environments with a bandwidth of 68.4 GB/s, this architecture effectively manages message fetching and ciphertext transmission. Additionally, each RSC incorporates a local scratchpad, implemented as a single-port, multi-bank 256-bit width SRAM. For PNL implementation, the shuffling unit requires FIFOs whose capacity doubles with each stage; these are also implemented as double-buffered SRAMs to minimize area overhead. Table II summarizes the synthesis results of ABC-FHE with two RSC instances. The overall design occupies 28.638 mm^2 and consumes up to 5.654 W of power. Based on the scaling methodology proposed in [31], scaling to a 7nm process would reduce the area to approximately 0.9 mm^2 and the power consumption to 2.1 W, making ABC-FHE highly feasible for client-side applications.

B. Evaluation Setup

To evaluate the performance of ABC-FHE, we employed a two-step approach: a cycle-level simulator was developed to measure latency, and Design Compiler was used to determine area and power characteristics. For this evaluation, we selected parameters used in CKKS, including a polynomial degree of 2^{16}. The bitwidth of the prime numbers was set to 36-bit, following the double scale technique [1], and the number of levels was doubled from the standard 12 to 24. Additionally, we assumed that messages from the client are encrypted to a

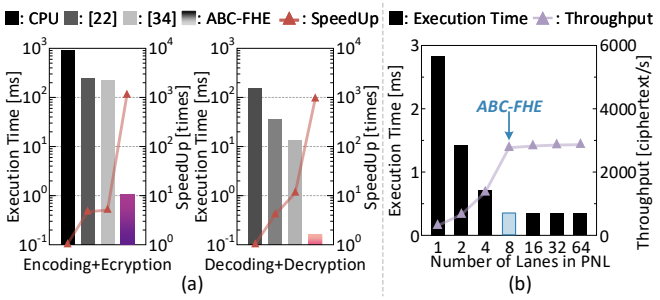


Fig. 5. Performance of ABC-FHE (a) Execution time and speed-up for encoding/encryption and decoding/decryption (b) Effect of number of pipelined NTT lanes on execution time and throughput.

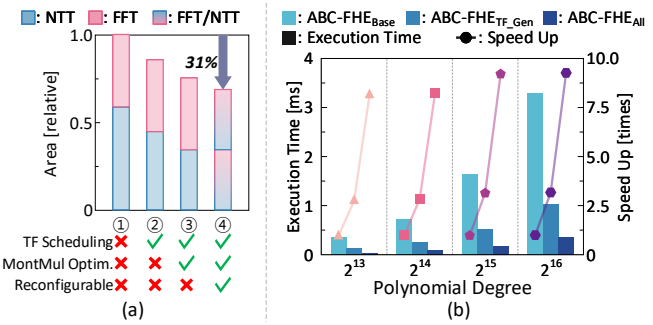


Fig. 6. (a) Area reduction as area optimization applied (b) Execution time improvement as memory optimization applied

24-level ciphertext, while ciphertexts from the server arrive at a 2-level state to facilitate efficient computation on the server and minimize computational overhead on the client.

C. Results and Discussions

Latency and Throughput. Figure 5 (a) highlights the significant performance improvements of ABC-FHE in terms of execution time and speed-up compared to prior works. Since previous designs do not support bootstrappable parameters, their reported latency was scaled by the proportion of operations for fair comparison. Additionally, to account for platform differences, FPGA and ASIC results were normalized to match ABC-FHE's 600 MHz frequency. ABC-FHE demonstrated a remarkable latency reduction: for encoding and encryption, it achieved a 1112× reduction compared to a PC-grade Intel Core i7-12700 using Lattigo [33] and a 214× reduction compared to SOTA ASIC and FPGA implementations. For decoding and decryption, ABC-FHE reduced latency by 963× compared to the CPU and 82× compared to state-of-the-art ASIC and FPGA implementations. These results underscore ABC-FHE's superior efficiency, particularly in handling encoding and encryption tasks with large levels through streaming operations.

Figure 5 (b) shows the impact of varying the number of PNLs on execution time and throughput. Under LPDDR5 specs typically used in client-side applications, the memory bottleneck was observed to cap performance at a maximum of 8 lanes, which ABC-FHE utilizes to maximize acceleration. By leveraging this optimal configuration, ABC-FHE maximized the number of ciphertexts processed per second, achieving peak performance in client-side FHE operations.

Area Optimization in RFE. Figure 6 (a) demonstrates the effectiveness of the proposed optimizations in reducing the hardware area of the RFE. All designs are evaluated using a P=8 MDC-based pipelined NTT architecture, comparing the area required to generate one FFT result and four NTT results. For fairness, the comparison assumes two RFEs for cases requiring separate hardware for NTT and FFT computations, with other configurations adjusted accordingly.

The baseline design adopts the commonly used radix-2 pipelined NTT architecture with distinct hardware implementations for NTT and FFT. Optimization techniques include twiddle factor scheduling to reduce the number of multipliers and Montgomery multiplier optimization to minimize the area of computational units. The final configuration represents a fully reconfigurable RFE capable of supporting both NTT and FFT operations. Combined, these optimizations achieved a 31% reduction in total area, underscoring the efficiency of the proposed approach.

On-chip Memory Optimization. To minimize EMA, ABC-FHE incorporates a unified OTF TF Gen and a PRNG to generate critical data such as twiddle factors, masks, errors, and keys directly on-chip. Figure 6 (b) evaluates the performance gain achieved by this approach across various polynomial degrees. Three configurations were analyzed: ABC-FHE$_{Base}$, which fetches all necessary data from external memory; ABC-FHE$_{TF_Gen}$, which generates only twiddle factors on-chip; and ABC-FHE$_{All}$, which uses both unified OTF TF Gen and PRNG to generate all data on-chip. Compared to external memory reliance, ABC-FHE$_{All}$ achieved a latency reduction of approximately 8.2–9.3×. Given that the combined area of the unified OTF TF Gen and PRNG constitutes only 6% of the total chip area, this design demonstrates significant performance improvements with minimal area overhead.

VI. Conclusion

In this work, we proposed ABC-FHE, a resource-efficient client-side accelerator designed to support bootstrappable parameters. Prior works fail to address bootstrappable parameters or rely on non-streaming architectures, leading to memory bottlenecks and latency overhead when throughput increases. To overcome these issues, ABC-FHE employs a streaming architecture for consistent throughput and generates parameters like random values and twiddle factors on-chip, minimizing external memory access. This design achieves significant speedups: 1112× and 214× faster encoding/encryption and 963× and 82× faster decoding/decryption compared to a CPU and the state-of-the-art accelerator, respectively.

Acknowledgment

This work was supported by the MSIT (Ministry of Science and ICT), Korea, through the IITP (Institute for Information & Communications Technology Planning & Evaluation) under the ITRC program (IITP-2025-RS-2020-II201847), the PIM Technology Development project (2022-0-01037), and the Graduate School of AI Semiconductor program (IITP-2025-RS-2023-00256472).

REFERENCES

[1] R. Agrawal, J. H. Ahn, F. Bergamaschi, R. Cammarota, J. H. Cheon, F. DM de Souza, H. Gong, M. Kang, D. Kim, J. Kim *et al.*, "High-precision rns-ckks on fixed but smaller word-size architectures: theory and application," in *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2023, pp. 23–34.

[2] T. Ahmed, "A low-power time-interleaved 128-point fft for ieee 802.15. 3c standard," in *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*. IEEE, 2013, pp. 1–5.

[3] Z. Azad, G. Yang, R. Agrawal, D. Petrisko, M. Taylor, and A. Joshi, "Race: Risc-v soc for en/decryption acceleration on the edge for homomorphic computation," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

[4] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 1986, pp. 311–323.

[5] J.-P. Bossuat, R. Cammarota, J. H. Cheon, I. Chillotti, B. R. Curtis, W. Dai, H. Gong, E. Hales, D. Kim, B. Kumara *et al.*, "Security guidelines for implementing homomorphic encryption," *Cryptology ePrint Archive*, 2024.

[6] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems: Recent advances and new frontiers," *Acm Sigkdd Explorations Newsletter*, vol. 19, no. 2, pp. 25–35, 2017.

[7] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer, 2017, pp. 409–437.

[8] W. Dai, D. D. Chen, R. C. Cheung, and C. K. Koc, "Area-time efficient architecture of fft-based montgomery multiplication," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 375–388, 2016.

[9] X. Deng, S. Fan, Z. Hu, Z. Tian, Z. Yang, J. Yu, D. Cao, D. Meng, R. Hou, M. Li *et al.*, "Trinity: A general purpose fhe accelerator," *arXiv preprint arXiv:2410.13405*, 2024.

[10] S. Di Matteo, M. L. Gerfo, and S. Saponara, "Vlsi design and fpga implementation of an ntt hardware accelerator for homomorphic seal-embedded library," *IEEE Access*, 2023.

[11] M. Garrido, S.-J. Huang, and S.-G. Chen, "Feedforward fft hardware architectures based on rotator allocation," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 2, pp. 581–592, 2017.

[12] M. Garrido, S.-J. Huang, S.-G. Chen, and O. Gustafsson, "The serial commutator fft," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 10, pp. 974–978, 2016.

[13] M. Garrido and P. Paz, "Optimum mdc fft hardware architectures in terms of delays and multiplexers," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 3, pp. 1003–1007, 2020.

[14] M. Huang, K. Gaj, S. Kwon, and T. El-Ghazawi, "An optimized hardware architecture for the montgomery multiplication algorithm," in *Public Key Cryptography–PKC 2008: 11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings 11*. Springer, 2008, pp. 214–228.

[15] J. K. Jang, H. K. Kim, M. H. Sunwoo, and O. Gustafsson, "Area-efficient scheduling scheme based fft processor for various ofdm systems," in *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*. IEEE, 2018, pp. 338–341.

[16] J. Johnston, "Parallel pipeline fast fourier transformer," in *IEE Proceedings F (Communications, Radar and Signal Processing)*, vol. 130, no. 6. IET, 1983, pp. 564–572.

[17] W. Jung, E. Lee, S. Kim, J. Kim, N. Kim, K. Lee, C. Min, J. H. Cheon, and J. H. Ahn, "Accelerating fully homomorphic encryption through architecture-centric analysis and optimization," *IEEE Access*, vol. 9, pp. 98 772–98 789, 2021.

[18] V. Kepuska and G. Bohouta, "Next-generation of virtual personal assistants (microsoft cortana, apple siri, amazon alexa and google home)," in *2018 IEEE 8th annual computing and communication workshop and conference (CCWC)*. IEEE, 2018, pp. 99–103.

[19] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn, "Sharp: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[20] J. Kim, G. Lee, S. Kim, G. Sohn, M. Rhu, J. Kim, and J. H. Ahn, "Ark: Fully homomorphic encryption accelerator with runtime data generation and inter-operation key reuse," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1237–1254.

[21] S. Kim, K. Lee, W. Cho, J. H. Cheon, and R. A. Rutenbar, "Fpga-based accelerators of fully pipelined modular multipliers for homomorphic encryption," in *2019 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*. IEEE, 2019, pp. 1–8.

[22] F. Krieger, F. Hirner, A. C. Mert, and S. S. Roy, "Aloha-he: A low-area hardware accelerator for client-side operations in homomorphic encryption," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–6.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[24] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *iEEE Access*, vol. 10, pp. 30 039–30 054, 2022.

[25] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[26] T. T. Nguyen, J. Kim, and H. Lee, "Ckks-based homomorphic encryption architecture using parallel ntt multiplier," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–4.

[27] T. Pöppelmann, T. Oder, and T. Güneysu, "High-performance ideal lattice-based cryptography on 8-bit atxmega microcontrollers," in *International conference on cryptology and information security in Latin America*. Springer, 2015, pp. 346–365.

[28] Prasetiyo, A. Putra, J.-Y. Kim *et al.*, "Morphling: A throughput-maximized tfhe-based accelerator using transform-domain reuse," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 249–262.

[29] A. Putra, Prasetiyo, Y. Chen, J. Kim, and J.-Y. Kim, "Strix: An end-to-end streaming architecture with two-level ciphertext batching for fully homomorphic encryption with programmable bootstrapping," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 1319–1331.

[30] S. S. Roy, F. Vercauteren, N. Mentens, D. D. Chen, and I. Verbauwhede, "Compact ring-lwe cryptoprocessor," in *Cryptographic Hardware and Embedded Systems–CHES 2014: 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings 16*. Springer, 2014, pp. 371–391.

[31] S. Sarangi and B. Baas, "Deepscaletool: A tool for the accurate estimation of technology scaling in the deep-submicron era," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[32] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[33] tuneinsight, "Lattigo v6," Online: https://github.com/tuneinsight/lattigo, Aug. 2024, ePFL-LDS, Tune Insight SA.

[34] J. Wang, C. Yang, J. Hou, F. Zhang, Y. Meng, Y. Su, and L. Liu, "A compact and efficient hardware accelerator for rns-ckks en/decoding and en/decryption," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2024.

MAS-ISP: A Proxy-Free Online Hyperparameter Optimization Framework for ISP Hardware System

Jiaming Liu[1], Xuan Huang[1], Zhijian Hao[2], Ruoxi Zhu[1], Qi Zheng[1], Shuocheng Wang[1], Shushi Chen[1], Chang Liu[1], Leilei Huang[3], Jun Tao[1,*] and Yibo Fan[1,*]

[1] State Key Laboratory of Integrated Chips and Systems, School of Microelectronics, Fudan University.

[2] State Key Discipline Laboratory of Wide Band-Gap Semiconductor Technology, School of Microelectronics, Xidian University.

[3] The Institute of Microelectronic Circuits and Systems, East China Normal University.

***Abstract*—The rapid advancement of visual autonomous systems, especially in autonomous driving, underscores the critical role of Image Signal Processors (ISPs) as they convert RAW sensor data into RGB images suited for visual interpretation. Traditional ISPs rely on tuning hyperparameters to adapt to varying imaging conditions; however, the vast parameter space and intricate tuning process pose significant challenges for real-time autonomous applications. Existing autonomous ISP hyperparameter optimization methods rely largely on offline or proxy-based online tuning, limiting their accuracy and responsiveness to real-time environmental changes. In response, we propose an online ISP hyperparameter optimization framework based on Deep Reinforcement Learning (DRL), marking the first proxy-free, real-time optimization approach. Our design exhibits a master-slave Multi-Agent System (MAS), enabling rapid and cooperative parameter optimization with improved inter-frame consistency. Furthermore, we design the MAS-ISP automated visual system, incorporating innovative hardware designs such as Strip Convolution Kernel and Stride-Aware Dual-Buffer Memory, which drastically reduce resource consumption in CNN hardware. MAS-ISP achieves 1080P@75FPS/240FPS on FPGA/ASIC platforms, supporting real-time and reliable visual systems.**

***Index Terms*—Hyperparameter optimization, ISP, proxy-free, hardware, multi-agent.**

I. Introduction

Artificial intelligence has revolutionized visual technologies [1]–[5], empowering automated systems to "see the world." Autonomous driving exemplifies this progress, where high-quality, real-time visual input is indispensable [6]–[9]. At the core of these systems is the Image Signal Processor (ISP), responsible for converting RAW sensor data into RGB images suitable for human and machine vision [10], [11].

Conventional ISPs rely on hyperparameter adjustments to adapt to diverse imaging conditions (e.g., indoor/outdoor, day/night) [12]. However, the vast hyperparameter space and reliance on manual tuning by expert teams significantly hinder efficiency [13]. Although automatic hyperparameter optimization (HPO) methods have been proposed, most are **offline** [13]–[16]. As illustrated in Fig. 1 (left), offline methods pre-tune parameters on datasets and fix them during runtime, leading to degraded image quality under real-world conditions. For high-stakes applications like autonomous driving, this lack of adaptability poses critical safety risks, emphasizing the need for real-time, online ISP HPO.

Recent research [17] introduced an **online** ISP HPO framework using deep learning (DL), shown in Fig.1 (middle). This method involves two steps: training a differentiable proxy to simulate the ISP and optimizing hyperparameters via training a parameter generation network. Once trained, the parameter generation network runs on hardware like Field-Programmable Gate Array (FPGA) [18]–[20] to enable dynamic parameter adjustment for the ISP. While novel, this approach faces key limitations: **1. Low Proxy Accuracy:** The proxy struggles with the ISP's nonlinear characteristics, limiting optimization reliability. **2. Lack of Inter-Frame Consistency:** Single-frame optimization ignores temporal correlations, causing unstable hyperparameters and visual discontinuities. **3. Hardware Inefficiency:** Processing RAW data in DL hardware introduces significant resource overhead due to its raster-scan format [21].

To address these challenges, we propose the first **proxy-free**, real-time **online** ISP HPO framework based on deep reinforcement learning (DRL), as illustrated in Fig.1 (right). Our framework introduces a Master-Slave Agent (Ms.Agent) architecture within a Multi-Agent System(MAS), where the master agent extracts global image features and slave agents optimize individual hyperparameters collaboratively. By learning from both ISP hyperparameters and processed RGB images, Ms.Agent ensures robust optimization, mitigating oscillatory quality variations and enhancing inter-frame consistency.

Our design leverages RGB images as the reward source, bypassing the computational burden of RAW data. Furthermore, we enhance hardware efficiency through the Strip Convolution Kernel and Stride-Aware Dual-Buffer (SADB) Memory architecture, transforming multi-row convolution into single-row operations, and drastically reducing intermediate buffering and computational overhead. Integrated with ISP hardware, our MAS-ISP system achieves real-time performance at 1080P@75FPS/240FPS on FPGA/ASIC platforms, enabling reliable and efficient autonomous visual systems.

Our Contributions:

1) **Proxy-Free Online Framework**: The first DRL-based proxy-free ISP HPO framework using hardware-in-the-loop training, eliminating proxy-related inaccuracies.

This work was supported in part by National Key R&D Program of China 2023YFB4404400, 2023YFB4404402 and 2023YFB4502802, in part by the National Natural Science Foundation of China 62031009, in part by Fudan-ZTE joint lab, in part by Alibaba Innovative Research (AIR) Program, in part by Alibaba Research Fellow (ARF) Program. *Corresponding authors: Jun Tao and Yibo Fan. (taojun@fudan.edu.cn and fanyibo@fudan.edu.cn)

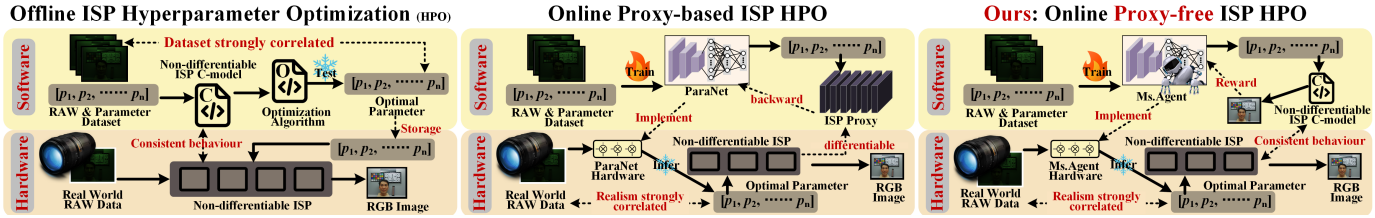


Fig. 1: Three types of ISP HPO methods: On the left is the offline method based on 0-order optimization, with parameters generated on the software side and strongly dataset-dependent. In the middle is the proxy-based online method using deep learning, where parameters are hardware-generated and closely related to real-world RAW data but require constructing differentiable proxies. On the right is our proxy-free online method based on deep reinforcement learning, where parameters are directly related to the real world without the need for proxies.

2) **Ms.Agent**: A master-slave multi-agent design enabling efficient, parallel hyperparameter optimization and improved inter-frame consistency.
3) **MAS-ISP**: A fully integrated automated visual system achieving real-time performance on FPGA/ASIC platforms while overcoming raster-scan inefficiencies.
4) **Efficient Hardware Design**: Novel Strip Convolution Kernel and SADB Memory architecture, significantly improving computational efficiency by reducing buffering and redundant operations.

II. Background

A. ISP Pipeline

ISP is central to camera imaging systems, processing raw sensor data (RAW) into high-quality RGB images through a sequence of pipeline stages [22]. RAW data follows the Bayer array format, with each pixel containing a single color value (R, Gr, Gb, or B). Sensors transmit pixel intensity to the ISP row by row via raster scanning [21].

For instance, the open-source xkISP-Lite [23] employs a seven-stage processing pipeline, including Digital Gain (DGAIN), Defect Pixel Correction (DPC), RAW-domain Denoising (DNS), White Balance (WB), Green Balance (GB), Demosaicing (DMC), and Edge Enhancement (EE). All stages involve 8 tunable parameters, creating a vast parameter space of 3.2746×10^{18} [17]. Similarly, commercial ISPs [24]–[26], utilize fixed pipelines comprising specific parameterized algorithms, designed to adapt to diverse imaging scenarios.

B. Proxy-Free ISP Hyperparameter Optimization

Unlike proxy-based HPO, which relies on differentiable models to approximate ISP behavior, proxy-free methods directly optimize hyperparameters without requiring differentiability, making them ideal for ISPs with complex, nonlinear operations. Common proxy-free approaches include 0-order optimizers and non-gradient-based learning techniques like evolutionary algorithms (EA) and deep reinforcement learning (DRL). For example, Mosleh et al. [14] applied the CMA-ES algorithm for hardware-in-the-loop ISP optimization, enabling end-to-end parameter tuning. Robidoux et al. [27] extended this approach to optimize high-dynamic-range ISPs, while Shi et al. [28] used EAs for architecture search and hyperparameter tuning within white-box ISP modules. DRL has also been explored; for instance, Ukcheol et al. [15] designed a white-box ISP pipeline using DRL, and Sun et al. [16] developed a DRL framework for black-box ISP optimization, leveraging RGB image quality as reward signals.

Despite their promise, EA and DRL methods generally involve prolonged optimization cycles, limiting their applicability for real-time adaptation in dynamic environments. Current proxy-free methods remain ill-suited for real-time online optimization, while proxy-based frameworks [17] face inherent accuracy and generalization challenges. Addressing the lack of **real-time adaptability in proxy-free ISP HPO** methods is thus a pressing research problem.

III. Methods

Our approach utilizes the model-free capabilities of DRL to develop a multi-agent online HPO framework with a master-slave agent architecture. Built upon Deep Q-learning (DQL), this framework facilitates automatic, real-time HPO for ISP pipelines and consists of two main phases: offline training and online optimization.

A. Multi-Agent Online HPO Framework

Offline Training: Our ISP HPO framework employs DRL to streamline hyperparameter adjustment in the offline phase. Unlike DL-based methods that require proxy training and separate parameter tuning (e.g., Using ParaNet), our single-stage training approach eliminates intermediate steps and directly optimizes ISP parameters without modeling proxy for the non-differentiable ISP. As shown in Fig.2, the offline phase includes a RAW dataset, an ISP hardware C-model, a master-slave multi-agent system (Ms.Agent), a replay buffer, and reward functions. At each timestamp t, the C-model processes RAW data I^{raw} with ISP parameters $\mathcal{P}_t$ to generate an RGB image I_t^{rgb} as per Eq.1:

$$I_t^{rgb} = f_{ISP}(I^{raw}, \mathcal{P}_t). \tag{1}$$

The agent observes the RGB image I_t^{rgb} and current parameters $\mathcal{P}_t$, selects an action A_t to update the parameters to $\mathcal{P}_{t+1}$, and iterates for subsequent timestamp $t+1$. Data tuples $(I_t^{rgb}, \mathcal{P}_t, A_t, R_t, I_{t+1}^{rgb}, \mathcal{P}_{t+1})$ are stored in the replay buffer for training.

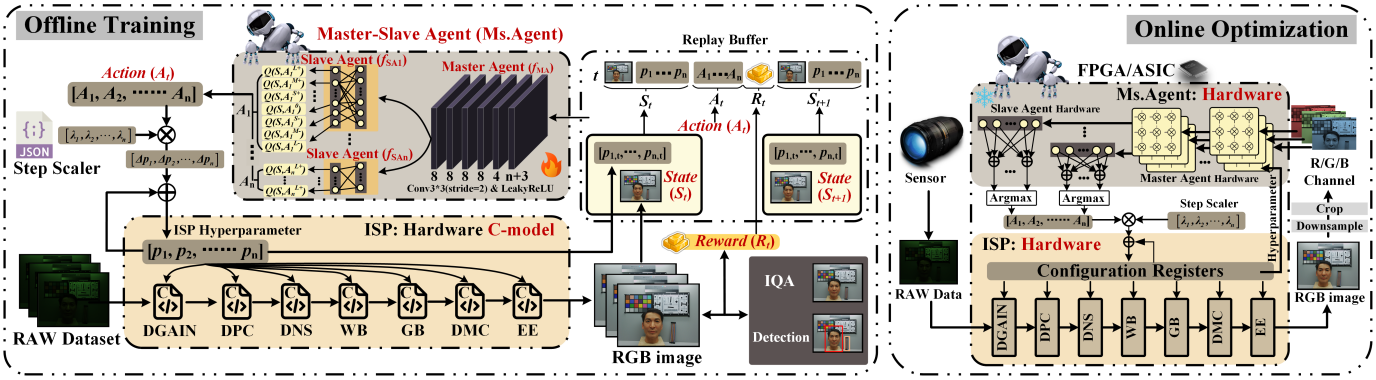


Fig. 2: Our multi-agent online HPO framework includes offline training and online optimization. Offline training uses a non-differentiable ISP hardware C-model in the loop, where current hyperparameters and RGB images are fed into the Ms.Agent (or random action), which adjusts the parameters and generates the next frame's RGB image to calculate the reward. These experiences are stored in the Replay Buffer for training the agent. Ms.Agent accelerates parameter tuning using multiple agents and scales the adjustment steps. Online optimization integrates the trained Ms.Agent with the ISP on the same hardware.

State Representation: The state $\mathcal{S}_t$ consists of the RGB image I_t^{rgb} and the ISP parameters $\mathcal{P}_t$:

$$\mathcal{S}_t = \{I_t^{rgb}, \mathcal{P}_t\}. \tag{2}$$

To enhance learning, the ISP parameters are expanded to match the resolution of the image and concatenated with the RGB image as input. This design ensures the agent accounts for parameter-image interactions and mitigates the risk of suboptimal convergence caused by image-only observations.

Reward Design: Incremental rewards $\mathcal{R}_t$, based on consecutive frame differences (Eq. 3), provide smoother feedback and faster adaptation to dynamic changes in RAW input:

$$\mathcal{R}_t = f_{IQA/Detect}(I_t^{rgb}) - f_{IQA/Detect}(I_{t-1}^{rgb}). \tag{3}$$

where $f_{IQA/Detect}$ represents evaluation metrics. For Human Vision Tasks (HVS), rewards derive from image quality metrics (e.g., PSNR, SSIM). For Machine Vision Tasks, metrics like mean Average Precision (mAP) based on bounding box detection are used. This design avoids the discrepancies between differentiable loss functions and final evaluation metrics seen in DL-based methods.

Q-Learning Algorithm: Agents are trained using Q-learning, where the Q-value update rule is as follows:

$$\begin{aligned}\mathcal{Q}(I_t^{rgb}, \mathcal{P}_t, \mathcal{A}_t) \leftarrow \mathcal{Q}(I_t^{rgb}, \mathcal{P}_t, \mathcal{A}_t) + \alpha(\mathcal{R}_t \\ +\gamma \max_{\mathcal{A}_{t+1}} \mathcal{Q}(I_{t+1}^{rgb}, \mathcal{P}_{t+1}, \mathcal{A}_{t+1}) - \mathcal{Q}(I_t^{rgb}, \mathcal{P}_t, \mathcal{A}_t)),\end{aligned} \tag{4}$$

where α is the learning rate, and γ is the discount factor.

Online Optimization: For real-time operation, we implement an integrated system comprising a sensor, FPGA/ASIC hardware, and a display. The system captures RAW data, processes it through ISP hardware, and generates RGB images for agent-based parameter analysis. Pre-trained agent parameters are frozen for efficient inference, and both agent and ISP hardware are co-located to minimize latency.

B. Master-Slave Agents (Ms.Agent)

Multi-Agent System: To accelerate HPO, we adopt a master-slave multi-agent architecture. Slave agents independently optimize individual ISP parameters, while the master agent coordinates their efforts. This parallel design enhances efficiency and scalability.

Master Agent: The master agent f_{MA} extracts high-level latent features from the RGB image and ISP parameters using a CNN with six convolutional layers and LeakyReLU activation (Eq. 5):

$$\mathcal{F}_t^{hidden} = f_{MA}(I_t^{rgb}, \mathcal{P}_t). \tag{5}$$

Centralized feature extraction improves the decision quality of the slave agents.

Slave Agents: Each slave agent uses a lightweight fully connected layer to estimate Q-values for its assigned parameter (Eq. 6), $\mathcal{A}_{t,n}$ is the set of actions for the n-th slave agent f_{SAn}:

$$\mathcal{Q}(I_t^{rgb}, \mathcal{P}_t, \mathcal{A}_{t,n}) = f_{SAn}(f_{MA}(I_t^{rgb}, \mathcal{P}_t)). \tag{6}$$

This lightweight structure minimizes computation overhead, making it ideal for resource-constrained edge devices.

Multi-Step Actions: Slave agents support multiple step sizes ($\mathcal{A}_{t,n}^{L+}$,$\mathcal{A}_{t,n}^{M+}$,$\mathcal{A}_{t,n}^{S+}$,$\mathcal{A}_{t,n}^{0}$,$\mathcal{A}_{t,n}^{S-}$,$\mathcal{A}_{t,n}^{M-}$,$\mathcal{A}_{t,n}^{L-}$) for parameter tuning $\mathcal{A}_{t,n}$, avoiding oscillations associated with fixed steps. A step scaler λ_n further refines adjustments for precise tuning.

Summary: Compared to DL-based methods, our framework eliminates the need for proxies, integrates ISP parameters into the state for inter-frame consistency, and employs incremental rewards to smooth learning. The master-slave architecture with multi-step actions accelerates optimization, enabling real-time, proxy-free ISP HPO through DRL.

IV. Hardware: MAS-ISP

A. Overall framework

We propose MAS-ISP, an integrated design that combines a multi-agent architecture for hyperparameter optimization (HPO) with imaging hardware. This design minimizes data

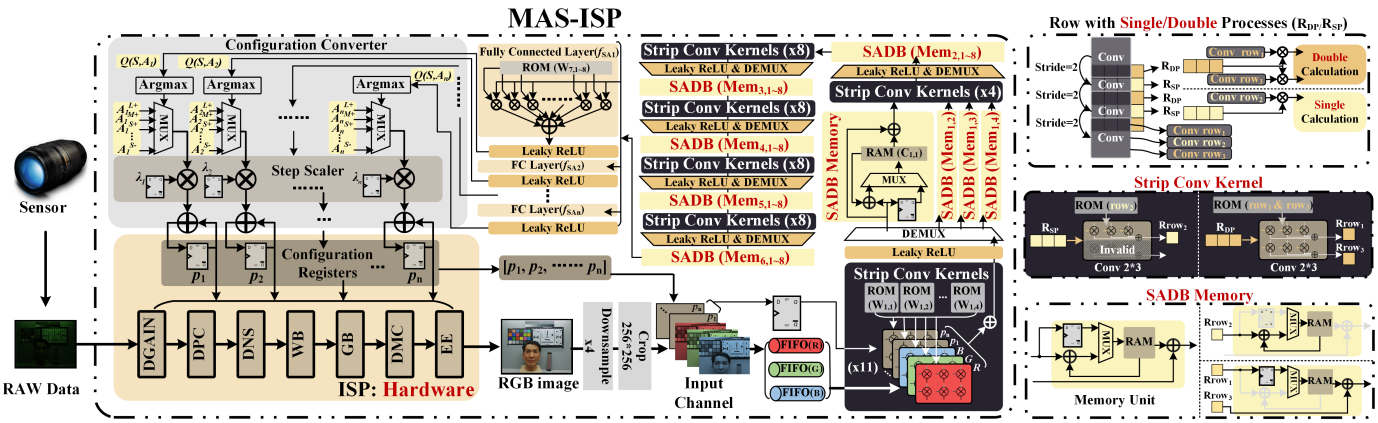


Fig. 3: The MAS-ISP includes the Ms.Agent streaming accelerator, ISP hardware, and a configuration converter that bridges the two. The ISP consists of configuration registers and 8 pipeline modules, while the configuration converter includes a step scaler for adjusting action magnitude. The Ms.Agent features 6 stages of convolution layers and 8 independent fully connected layers. The right image shows the row processing characteristics of a stride-2 convolution, along with our novel components, the Strip Conv Kernel and SADB Memory.

transfer and processing delays between agents and ISP hardware, enabling real-time response in critical applications such as autonomous driving. Additionally, it simplifies system complexity by eliminating redundant memory and I/O components. As illustrated in Fig. 3, MAS-ISP consists of three main components: ISP hardware, Ms.Agent's miniaturized streaming CNN hardware, and a configuration converter for mapping Q-values to parameter adjustments.

ISP Hardware and Configuration Converter: The ISP hardware receives RAW sensor data through raster scanning, preprocesses it, and generates RGB images. These images, along with the ISP parameters stored in configuration registers, are input to the agent. The configuration converter selects actions based on the maximum Q-value $\mathcal{Q}(I_t^{rgb}, \mathcal{P}_t, \mathcal{A}_{t,n})$ and applies a step scaler to adjust the action's magnitude, accounting for variations in hyperparameter sensitivities and ranges.

Temporal Window Optimization: In real-time video processing, agents often face temporal constraints, where parameter updates for a given frame might lag behind incoming raw data [17]. To address this, we employ 4× downsampling and cropping, focusing on the central region of the image, where critical information is usually located. This approach allocates processing time more effectively, prioritizing higher-value image regions and ensuring new ISP parameters are ready for subsequent frames. This optimization significantly improves responsiveness without compromising image quality.

Streaming Accelerator for Ms.Agent: The Ms.Agent accelerator leverages the pixel delay characteristics of downsampled image data to achieve efficient convolutional processing. The neural network includes 6 stride-2 convolutional layers for the master agent and 8 lightweight parallel fully connected layers for the slave agents, all using LeakyReLU activation functions. To further reduce resource usage, we introduce two innovative components: Strip Convolution Kernel and Stride-Aware Dual-Buffer (SADB) Memory. Detailed descriptions of these components are provided in Section IV-B.

B. Efficient Components for Stride-2 Convolution

Row Classification-Single and Double Processes: For 3×3 convolution kernels with a stride of 2, adjacent input rows are processed differently, as shown in the top right of Fig. 3. Rows with Single Process (R_{SP}): Interact only with the kernel's second row. Rows with Double Processes (R_{DP}): Interact with both the first and third kernel rows. This classification underpins the design of the subsequent components.

Strip Convolution Kernel: The Strip Convolution Kernel is specifically tailored for 3×3 stride-2 convolutions, as depicted in the middle right of Fig. 3. It processes one input row at a time, efficiently utilizing the unique coverage patterns of R_{SP} and R_{DP}. For R_{SP}: Only the kernel's second row is active, requiring three multipliers (1×3) for computation. For R_{DP}: Two sets of three multipliers ($2 \times 1 \times 3$) operate in parallel, computing results for both the first and third kernel rows. Compared to traditional 3×3 convolution, this design reduces multiplier usage by 33%, replacing nine-input adders with two three-input adders. Processing single input rows eliminates redundant computations, significantly improving resource utilization and efficiency.

Stride-Aware Dual-Buffer (SADB) Memory: The SADB Memory is optimized for stride-2 convolution as illustrated in the lower right of Fig. 3, using a single-row memory structure with pipelined storage, consisting of a RAM unit, two adders, a multiplexer, and a register. For R_{SP}, activated inputs for the second kernel row ($Rrow_2$) are added to the results of the first kernel row ($Rrow_1$) stored in RAM. The cumulative sum updates the RAM contents. For R_{DP}, activated inputs for the third kernel row ($Rrow_3$) are added to the cumulative sum of $Rrow_1$ and $Rrow_2$ in RAM to produce the final convolution result. Meanwhile, $Rrow_1$ is buffered in a register for one clock cycle before overwriting the original data. This design minimizes memory usage by leveraging a single-

TABLE I: Optimization results for continuous static and dynamic frames within 1 second, with red representing the best performance and blue representing the second-best performance.

Method	Mode	Type	Proxy	Kodak (Static)				Oneplus (Static)				BDD100K (Dynamic)		
				LPIPS↓	PSNR↑	SSIM↑	NIQE↓	Person↑	Car↑	Bus↑	mAP↑	Person↑	Car↑	mAP↑
Mosleh *et al.* [14] (*CVPR'20*)	Offline	EA	-	0.3507	14.4621	0.7204	8.3667	0.2503	0.3437	0.1763	0.2567	0.2317	0.2494	0.2405
Sun *et al.* [16] (*AAAI'24*)	Offline	DRL	-	0.3436	15.1633	0.7421	7.8325	0.2771	0.3941	0.2053	0.2921	0.2768	0.3044	0.2906
Liu *et al.* [17] (*DAC'24*)	Online	DL	UNet	0.3217	16.0153	0.7675	6.3353	0.3003	0.5921	0.2834	0.3919	0.3468	0.3412	0.3440
Liu *et al.* [17] (*DAC'24*)	Online	DL	HFP	0.3005	17.6885	0.8154	5.8983	0.3972	0.7092	0.3938	0.5001	0.3468	0.3534	0.3501
Ours	Online	DRL	-	0.2991	17.7912	0.8203	5.8595	0.4005	0.7112	0.4018	0.5045	0.3633	0.3681	0.3657



Robidoux et al, (CVPR'20) Sun et al, (AAAI'24) Liu et al, (DAC'24 -UNet) Liu et al, (DAC'24 -HFP) Ours Groundtruth

Fig. 4: Visual results for various ISP HPO methods on the static frames of the Kodak dataset, using xkISP-Lite.



Robidoux et al, (CVPR'20) Sun et al, (AAAI'24) Liu et al, (DAC'24 -UNet) Liu et al, (DAC'24 -HFP) Ours

Fig. 5: Visual results of detection for various ISP HPO methods on the dynamic frames of BDD100K dataset, using xkISP-Lite.

row storage strategy, reducing memory access frequency and computational complexity. By managing intermediate results efficiently and enabling pipelined storage across rows, SADB memory enhances throughput and processing speed while maintaining a compact footprint.

V. Experiments

Real-world real-time scenarios typically exhibit gradual changes in low-speed conditions, with consecutive video frames remaining nearly static over short periods [29]. In contrast, high-speed scenarios involve significant inter-frame variations. To provide a comprehensive analysis, we categorize video frames into **static frames** and **dynamic frames**.

A. Experiment Setup

Dataset: For training, we used the Pixelshift200 dataset [30] for HVS tasks and the OnePlus training set [22] for machine vision tasks. For static-frame testing, the Kodak dataset evaluated ISP HPO performance for HVS tasks, while the OnePlus validation set assessed nighttime object detection. For dynamic-frame testing, high-speed autonomous driving scenarios were tested using the BDD100K validation set [31].

TABLE II: The training cost of two online HPO methods

Method	Type	Proxy	Parameter (M)	Operations (FLOPs)	GPU Mem
Liu *et al.* [17] (*DAC'24*)	DL	UNet	ParaNet: 0.0093 Proxy: 4.3707	ParaNet: 0.0157 G Proxy: 21.1975 G	674 M
Liu *et al.* [17] (*DAC'24*)	DL	HFP	ParaNet: 0.0093 Proxy: 2.8399	ParaNet: 0.0157 G Proxy: 98.8356 G	2526 M
Ours	DRL	-	Ms.Agent: 0.0096	Ms.Agent: 0.0169 G	236 M

Training and Metrics: Our method employed DQL training on an NVIDIA A6000 GPU. Rewards were defined as normalized PSNR for HVS tasks and mAP for detection tasks. The ϵ-greedy strategy annealed ϵ from 1 to 0.08 over 600 episodes, with each episode comprising 200 samples. The learning rate was set to 1×10^{-4}. Evaluation metrics [32] included PSNR, SSIM, LPIPS, and NIQE for perceptual quality and mAP for accuracy. Yolov3 [33] is the detection evaluator.

B. Real-Time Continuous Frame Optimization

Static Frames: To simulate low-speed applications, we tested ISP HPO methods over 30 consecutive static frames (1-second sequence). Table I presents the results on Kodak and OnePlus datasets, excluding the first 0.1 seconds to minimize initialization effects. Offline methods [14], [16] failed to achieve satisfactory performance within the 1-second constraint due to iterative computations. Proxy-based online methods [17] showed better performance, with HFP outperforming UNet by 1.6732 dB on Kodak, emphasizing the impact of proxy quality. In contrast, our proxy-free DRL framework, leveraging MAS for real-time optimization, eliminated proxy-induced inaccuracies. Fig. 4 demonstrates enhanced detail retention and reduced overexposure compared to [17].

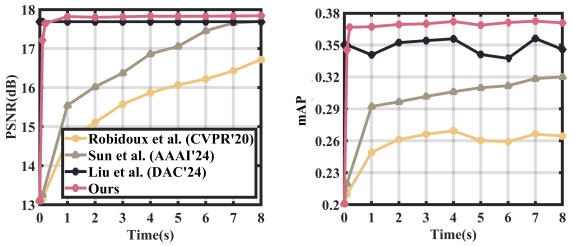
Dynamic Frames: For high-speed scenarios, we evaluated ISP HPO on 30 dynamic frames from the BDD100K validation set. Table I shows that CMA-ES-based methods [14] and DRL-based offline methods [16] struggled with rapid changes and failed to adapt in real-time. Our MAS framework surpassed proxy-based methods [17], achieving state-of-the-art detection accuracy. Fig. 5 highlights our model's ability to optimize detection accuracy, even when the visual output is less pleasing

TABLE III: Comparison of hardware resources and performance on FPGA and ASIC platforms.

Method	Type	FPGA Implementation (Xilinx XC7Z035-2FFG676I)			ASIC Implementation (GF 28nm)		
		LUTs/FFs/BRAMs/DSPs	Clock	Frame Rate	Area (mm^2)	Clock	Frame Rate
Liu *et al.* [17] (*DAC'24*)	DL	ParaNet: 37692 / 78936 / 36.5 / 365 xkISP-Lite: 17212 / 14832 / 25.5 / 127 Sum: 54904 / 93768 / 62.0 / 492	156.25 MHz	75 FPS	ParaNet: 0.7446 xkISP-Lite: 0.5514 Sum: 1.2960	500 MHz	240 FPS
Ours	DRL	Ms.Agent: 35884 / 79134 / 15.5 / 283 xkISP-Lite: 17212 / 14832 / 25.5 / 127 Sum: 53096 / 93966 / 41.0 / 410	156.25 MHz	75 FPS	Ms.Agent: 0.5782 xkISP-Lite: 0.5514 Sum: 1.1296	500 MHz	240 FPS



Fig. 6: A visual comparison of inter-frame consistency and object detection results for the two online HPO methods on the dynamic frames of BDD100K dataset, using xkISP-Lite.



(a) The static-frame Test (b) The dynamic-frame Test

Fig. 7: The time-performance curves for static and dynamic frames show the sustained optimization performance of various HPO methods over a longer period (8 seconds).

to human eyes. This highlights the divergence between human and machine vision.

C. Inter-Frame Consistency

Fig. 6 compares inter-frame consistency on nighttime driving frames from the BDD100K dataset. Our method achieved smooth visual transitions without over-adjustment, ensuring stable visual quality. In contrast, [17] exhibited local overexposure, degrading detection performance. These results underscore the importance of inter-frame consistency for high-speed applications.

D. Time-Performance Curve

Fig. 7 shows the time-performance curves for 8-second sequences of static and dynamic frames, evaluated with PSNR (Kodak) and mAP (BDD100K). Offline methods [14], [16] exhibited slow improvement, unsuitable for real-time systems.

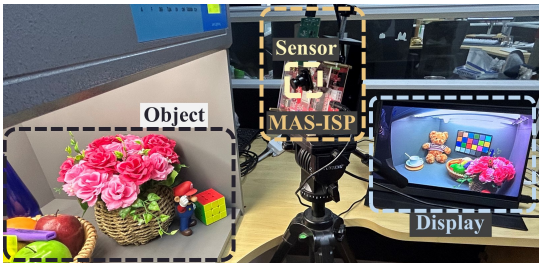


Fig. 8: The entire real-time automated vision system, which implements MAS-ISP on an FPGA, has been tested in practice.

Our approach matched [17] within 3 frames and maintained superior consistency over time, especially in dynamic scenarios, where offline methods failed to stabilize.

E. Training Complexity

Table II compares training complexity. Proxy-based methods incurred high computational costs due to additional proxy training. For instance, HFP significantly increased GPU memory requirements. Our proxy-free approach eliminated this overhead, reducing training time and resource consumption.

F. Hardware Implementation

We implemented the MAS-ISP framework on a Xilinx XC7Z035-2FFG676I FPGA and synthesized it on GF 28nm technology. As shown in Fig. 8, the system integrated a Sony IMX334 sensor with 12-bit quantization for both Ms.Agent and ISP operations. Table III summarizes hardware performance, achieving 1080P@75FPS/240FPS. Compared to [17], our framework reduced BRAM usage in neural network hardware by 57.6% through innovations like the strip convolution kernel and SADB memory. Quantization loss was negligible ($<$0.08 dB PSNR on Kodak and $<$0.03 mAP on OnePlus), ensuring optimal performance.

VI. Conclusion

This paper presents the first proxy-free, real-time ISP HPO framework tailored for autonomous systems, combining a master-slave MAS with innovative hardware designs like the strip convolution kernel and SADB memory. Our DRL-based online approach eliminates proxy-related inaccuracies, achieving state-of-the-art results in both static and dynamic scenarios while ensuring inter-frame consistency and low computational overhead. Experiments demonstrate the framework's versatility for human and machine vision tasks, achieving 1080P@70FPS/240FPS with significantly reduced resource consumption.

REFERENCES

[1] Y. He, S. Qu, G. Lin, C. Liu, L. Zhang, and Y. Wang, "Processing-in-sram acceleration for ultra-low power visual 3d perception," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 295–300.

[2] M. Morsali, B. Reidy, D. Najafi, S. Tabrizchi, M. Imani, M. Nikdast, A. Roohi, R. Zand, and S. Angizi, "Lightator: An optical near-sensor accelerator with compressive acquisition enabling versatile image processing," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[3] L. Huang, C. Yang, Y. Gong, Y. Sui, X. Zang, A. Goeckner, Q. Zhu, and B. Yuan, "Algorithm and hardware co-design for energy-efficient neural slam," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–4.

[4] Y. Zhong, J. Liu, X. Huang, J. Liu, Y. Fan, and M. Wu, "Cdcnet: A fast and lightweight dehazing network with color distortion correction," in *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2024, pp. 3020–3024.

[5] S. Wang, J. Liu, R. Zhu, J. Lian, C. Huang, H. Wei, and Y. Fan, "A lightweight cnn for detail enhancement and color correction of low-light capsule endoscopy," in *2024 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, 2024, pp. 1–5.

[6] T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang, and Y. Shan, "Yolo-world: Real-time open-vocabulary object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024, pp. 16 901–16 911.

[7] P. Zhao, G. Yuan, Y. Cai, W. Niu, Q. Liu, W. Wen, B. Ren, Y. Wang, and X. Lin, "Neural pruning search for real-time object detection of autonomous vehicles," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 835–840.

[8] Q. Zhang, L. Chi, D. Wang, M. Msahli, G. Memmi, T. Zhang, C. Zhang, and H. Qiu, "Laser shield: a physical defense with polarizer against laser attacks on autonomous driving systems," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[9] S. Wang, J. Liu, Y. Zhong, R. Zhu, J. Lian, H. Zhang, and Y. Fan, "A fast saturation based dehazing framework with accelerated convolution and attention block," in *ICASSP 2025 - 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2025, pp. 1–5.

[10] Z. Liang, J. Cai, Z. Cao, and L. Zhang, "Cameranet: A two-stage framework for effective camera isp learning," *IEEE Transactions on Image Processing*, vol. 30, pp. 2248–2262, 2021.

[11] Y. Zhang, Y. Zhang, Z. Zhang, M. Zhang, R. Tian, and M. Ding, "Isp-teacher: Image signal process with disentanglement regularization for unsupervised domain adaptive dark object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 7, 2024, pp. 7387–7395.

[12] I. Morawski, Y.-A. Chen, Y.-S. Lin, S. Dangi, K. He, and W. H. Hsu, "Genisp: Neural isp for low-light machine cognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 630–639.

[13] E. Tseng, F. Yu, Y. Yang, F. Mannan, K. S. Arnaud, D. Nowrouzezahrai, J.-F. Lalonde, and F. Heide, "Hyperparameter optimization in black-box image processing using differentiable proxies." *ACM Trans. Graph.*, vol. 38, no. 4, pp. 27–1, 2019.

[14] A. Mosleh, A. Sharma, E. Onzon, F. Mannan, N. Robidoux, and F. Heide, "Hardware-in-the-loop end-to-end optimization of camera image processing pipelines," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7529–7538.

[15] U. Shin, K. Lee, and I.-S. Kweon, "Drl-isp: Multi-objective camera isp with deep reinforcement learning," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7044–7051, 2022.

[16] X. Sun, Z. Zhao, L. Wei, C. Lang, M. Cai, L. Han, J. Wang, B. Li, and Y. Guo, "Rl-seqisp: Reinforcement learning-based sequential optimization for image signal processing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 5, 2024, pp. 5025–5033.

[17] J. Liu, Z. Liu, X. Huang, R. Zhu, Q. Zheng, Z. Hao, T. Liu, J. Tao, and Y. Fan, "Auto-isp: An efficient real-time automatic hyperparameter optimization framework for isp hardware system," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[18] J. Liu, Z. Hao, L. Zhao, J. Lian, J. Tao, Y. Fan, and Q. Feng, "A low valid throughput loss ldpc codec architecture with variable code rate," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 71, no. 5, pp. 2644–2648, 2024.

[19] M. Li, P. Li, S. Yin, S. Chen, B. Li, C. Tong, J. Yang, T. Chen, and B. Yu, "Winogen: A highly configurable winograd convolution ip generator for efficient cnn acceleration on fpga," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[20] Z. Hao, C. He, J. Liu, Q. Zheng, J. Xu, P. Ma, X. Ma, and Y. Hao, "A novel transform accelerator with fast kernel selection and efficient transform circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2025.

[21] N. Memon, D. L. Neuhoff, and S. Shende, "An analysis of some common scanning techniques for lossless image coding," *IEEE transactions on image processing*, vol. 9, no. 11, pp. 1837–1848, 2000.

[22] K. Yu, Z. Li, Y. Peng, C. C. Loy, and J. Gu, "Reconfigisp: Reconfigurable camera image processing pipeline," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4248–4257.

[23] OpenASIC, "xkISP," https://github.com/openasic-org/xkISP, 2022.

[24] Arm Ltd., "Mali-C71," https://www.arm.com/products/silicon-ip-multimedia/image-signal-processor/mali-c71, accessed: 2019-08-30.

[25] Movidius, "Myriad 2 Documentation," https://usermanual.wiki/Docume-nt/MDKMA2x5xSIPPUserManual.995467580.

[26] OnSemi Ltd., "AP0202AT," https://www.onsemi.com/products/sensors/i-mage-signal-processors-isps/ap0202at.

[27] N. Robidoux, L. E. G. Capel, D.-e. Seo, A. Sharma, F. Ariza, and F. Heide, "End-to-end high dynamic range camera pipeline optimization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 6297–6307.

[28] Y. Shi, S. Li, X. Jia, and J. Liu, "Refactoring isp for high-level vision tasks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 2366–2372.

[29] Z. Wu, Z. Sheng, X. Zhang, S.-Y. Cao, R. Zhang, B. Yu, C. Zhang, B. Yang, and H.-L. Shen, "Starnet: Low-light video enhancement using spatio-temporal consistency aggregation," *Pattern Recognition*, p. 111180, 2024.

[30] G. Qian, Y. Wang, J. Gu, C. Dong, W. Heidrich, B. Ghanem, and J. S. Ren, "Rethinking learning-based demosaicing, denoising, and super-resolution pipeline," in *2022 IEEE International Conference on Computational Photography (ICCP)*. IEEE, 2022, pp. 1–12.

[31] F. Yu, H. Chen, X. Wang, W. Xian, Y. Chen, F. Liu, V. Madhavan, and T. Darrell, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[32] Q. Zheng, Y. Fan, L. Huang, T. Zhu, J. Liu, Z. Hao, S. Xing, C.-J. Chen, X. Min, A. C. Bovik *et al.*, "Video quality assessment: A comprehensive survey," *arXiv preprint arXiv:2412.04508*, 2024.

[33] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

From Flatland to Forest: Exploring Pareto-optimal Design through RTL Hierarchy Trees

Donger Luo[1] Qi Sun[2] Xingheng Li[1] Cheng Zhuo[2] Bei Yu[3] Hao Geng[1†]
[1]ShanghaiTech University [2]Zhejiang University [3]Dept. CSE, CUHK

***Abstract*—The growing complexity of modern hardware has created vast design spaces that are difficult to explore efficiently. Current design space exploration (DSE) methods treat designs as flat parameter vectors, failing to leverage the rich structural information inherent in hardware architectures. This paper presents a novel RTL hierarchy aware approach to microarchitecture DSE that exploits the natural structure of hardware designs. We propose an RTL hierarchy aware kernel that enables direct comparison of RTL hierarchies, preserving their structural characteristics. Our method incorporates module importance derived from hierarchical synthesis reports through a weighted kernel extension. Additionally, we introduce a clustering method that leverages the proposed kernel to identify distinct architectural patterns, enabling efficient parallel evaluation. Experimental results and ablation studies on a Gemmini-based RISC-V SoC demonstrate the superiority of our approach.**

I. Introduction

Scaling Law, predicting that AI models are doubling in size every six months, more rapidly spawn customer-centric or application-specific silicon solutions like System-on-Chip (SoC). Modern SoCs must balance multiple competing objectives, from maximizing performance and minimizing power consumption to optimizing area utilization and reducing design costs. Each architectural decision, including processor configurations, memory hierarchies, interconnect topologies, and peripheral interfaces, can profoundly influence the overarching system attributes, thereby opening up an extensive design space. The rise of parameterized hardware languages like Chisel [1] and SpinalHDL [2] further expanded the design space by providing flexible microarchitecture design options. Design space exploration (DSE) has thus become crucial, with machine learning-aided approaches, particularly Bayesian optimization (BO), gaining prominence [3]–[8]. These methods use surrogate models to guide the exploration of the designs' microarchitecture parameters, reducing the number of expensive design evaluations needed. However, current approaches suffer from a fundamental limitation: they reduce the rich, multidimensional nature of hardware architecture into flat parameter vectors.

In this paper, we argue that hardware designs should not be represented as flat parameters in DSE frameworks. Consider a modern AI accelerator, it's an intricate system where compute arrays, memory hierarchies and interconnect networks work in concert to achieve optimal performance. Flattening such intricate designs into parameter vectors strips away critical structural information, much like reducing a three-dimensional object to its shadow. The hierarchical relationship between memory levels, the spatial patterns of data flow, and the topological characteristics of the interconnect, all become indistinguishable from auxiliary parameters in a one-dimensional list. Two designs with nearly identical memory hierarchies but different buffer sizes would appear as distant points in the parameter space, preventing search algorithms from leveraging their architectural similarity. This "flattening" of structural information not only makes it harder for machine learning methods to understand design implications but also creates an artificially complex search landscape where natural architectural patterns are obscured.

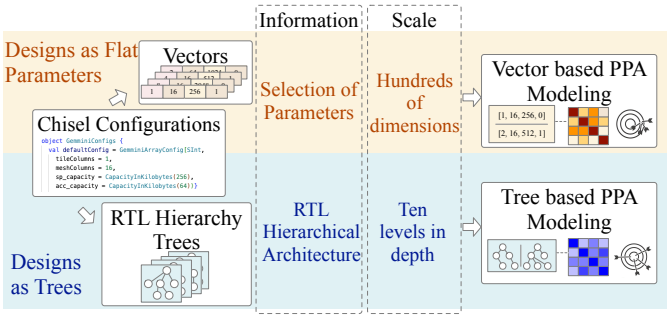


Fig. 1 While Chisel configurations have traditionally been represented as vectors of flat parameters, this approach provides limited information and creates an overly complex modeling space. In contrast, RTL hierarchy trees offer richer information content while providing a more constrained and manageable modeling space.

The flat parameter representation creates two additional critical challenges. First, the sheer number of design parameters, hundreds in Chisel generators like Gemmini [9], leads to the curse of dimensionality [10]. Furthermore, this "flatland" is full of "dead zones" since most parameter combinations fail to pass basic sanity checks. Second, the heterogeneous nature of these parameters poses a fundamental modeling challenge. It is difficult for a machine learning model to effectively handle the diverse parameter types simultaneously, including integer, boolean, and categorical.

To break free from this limited perspective, we propose an RTL hierarchy aware framework that preserves and exploits the natural structure of hardware designs by representing the designs with its hierarchy trees of RTL modules, as shown in Fig. 1. To make comparisons between these trees, we

†Corresponding author, also with Shanghai Engineering Research Center of Energy Efficient and Custom AI IC.

adopt the Weisfeiler-Lehman (WL) subtree kernel method. By constructing a Gaussian process regression model based on a weighted version of the WL subtree kernel, we enable predicting means and variance of PPAs with trees as inputs. We develop a clustering strategy that identifies and groups structurally similar designs, enabling efficient parallel evaluations of designs with distinct architectural patterns.

The main contributions of this work are:

- The first framework to break away from the flat parameter-centric perspective in microarchitecture design space exploration by proposing a novel RTL hierarchy aware kernel, which naturally captures structural similarities between different design variants.
- Theoretical analysis establishes that our approach achieves better sample complexity than the traditional parameter-based RBF kernel methods.
- A Gaussian Progress Regression model based on the weighted extension of the proposed kernel that incorporates module importance, enabling more accurate prediction by accounting for the differential impact of various hardware components.
- A clustering method that leverages the proposed RTL hierarchy aware kernel to identify distinct architectural patterns, enabling efficient parallel evaluation and improved exploration strategies.

II. Preliminaries

A. Chisel-based Agile Design and RTL Hierarchy Tree

Modern hardware design methodologies increasingly adopt Chisel [1] to generate RTL code for its parameterized hardware generation capabilities. Unlike traditional HDLs, Chisel allows designers to create flexible hardware generators that produce related designs through parameter configuration.

In conventional microarchitecture design space exploration (DSE), designs are represented as parameter vectors in Chisel:

$$\mathbf{x} = [p_1, p_2, ..., p_n] \in \mathcal{P}_1 \times \mathcal{P}_2 \times ... \times \mathcal{P}_n, \quad (1)$$

where p_i represents the i-th parameter and $\mathcal{P}_i$ is its corresponding domain. During Chisel code compilation, Chisel creates a hierarchical structure reflecting module relationships. This modularized approach facilitates direct comparison between modules through their instance names, while the hierarchical structure provides valuable architectural insights to guide the exploration process.

B. Bayesian Optimization and Gaussian Process

Bayesian optimization (BO) [11] optimizes expensive black-box functions by constructing a surrogate model and using an acquisition function to identify promising design points. Given design configurations $\mathbf{x}$ and performance metrics $\mathbf{y}$, Gaussian process regression (GPR) serves as the surrogate model, providing predictions and uncertainty estimates.

Multi-objective Bayesian optimization (MOBO) extends this framework to handle multiple objective functions $\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), ..., f_M(\mathbf{x})]$, each modeled by an independent GP.

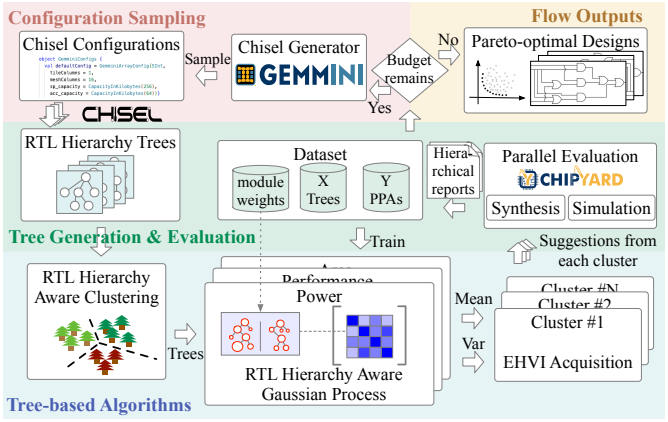


Fig. 2 Overall flow. The proposed approach leverages RTL hierarchy trees for design representation and conducts exploration in a tree-structured design space.

C. Problem Formulation

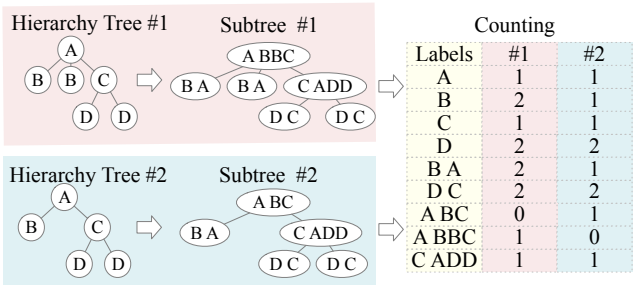
Problem 1 (Design Space Exploration for Microarchitecture)**.** *The goal of microarchitecture design space exploration is to automatically identify Pareto-optimal configurations that optimize multiple Quality of Results (QoR) metrics, including performance, power consumption, and area. This optimization process takes place within a defined microarchitecture configuration space.*

III. Overall Flow of Tree-based RTL Hierarchy Aware Exploration Framework

In this section, we present our proposed framework for microarchitecture design space exploration based on RTL hierarchy. This iterative approach takes Chisel configurations as input and generates Pareto-optimal configurations as output. Fig. 2 illustrates the workflow of our method.

Each iteration begins with sampling from the microarchitecture's Chisel parameter design space. Distinguished from conventional approaches, these samples are transformed in parallel into RTL hierarchy trees through the Chisel compiler, turning the design space from a "flatland" into a "forest". This procedure takes a few minutes but is rather quick compared with the evaluation of these designs which takes hours.

The Weisfeiler-Lehman subtree kernel (detailed in Section IV) is employed to process the RTL hierarchical information embedded in these trees. By integrating the kernel with our clustering method (detailed in Section V-B), we partition the "forest" into distinct clusters based on the hierarchical similarities among trees. These clustered trees are then fed into a specialized Gaussian process (GP) regression model (detailed in Section V-A) for PPA prediction. The core of our GP model is an enhanced subtree kernel, where nodes are weighted according to hierarchical information extracted from modern EDA tools' PPA reports. The model's predicted means and variances are grouped by their corresponding tree clusters, and the EHVI algorithm [12] is applied to select one suggestion from each group (detailed in Section V-B).



Labels	#1	#2
A	1	1
B	2	1
C	1	1
D	2	2
B A	2	1
D C	2	2
A BC	0	1
A BBC	1	0
C ADD	1	1

Fig. 3 The Weisfeiler-Lehman subtree kernel computation process with one iteration consists of three steps: (1) Initial RTL hierarchy trees representing two different microarchitectural implementations; (2) Subtree pattern extraction through WL relabeling, where each node is relabeled based on its neighbors; and (3) Feature counting of node and subtree labels to generate feature vectors. The similarity score is computed as the dot product of the two feature count vectors.

The multiple suggestions enable parallel execution of the most time-consuming evaluation phase. These suggestions are evaluated by generating complete RTL code using Chipyard [13]. The generated RTL code is then synthesized and simulated using commercial EDA tools to obtain PPA values and hierarchical reports, which are stored in the dataset for training the GP model in subsequent iterations.

The proposed flow takes advantage of the RTL-level architecture information and hierarchy reports generated from modern EDA tools, enabling more efficient microarchitecture design space exploration.

IV. RTL Hierarchy Aware Kernel

In this section, we first review the fundamental role of kernels in Gaussian Process regression and discuss the limitations of traditional kernels when applied to hardware design. We then present our RTL hierarchy aware kernel based on the Weisfeiler-Lehman subtree algorithm [14], which effectively captures the structural characteristics of hardware designs. Finally, we demonstrate the superiority of the WL subtree kernel over the conventional RBF kernel in microarchitecture DSE through theoretical analysis.

A. Kernels for Gaussian Process

In Gaussian process regression (GPR), kernels enable the quantification of similarities between different inputs, allowing for probabilistic predictions at unobserved points. For microarchitecture design space exploration using Bayesian Optimization (BO), kernels model the relationship between microarchitecture configurations.

The Radial Basis Function (RBF) kernel has been the de facto choice for traditional microarchitecture DSE:

$$k_{\text{RBF}}(x, x') = \exp\left(-\frac{|x - x'|^2}{2l^2}\right), \tag{2}$$

where l is the bandwidth parameter.

Algorithm 1 Weisfeiler-Lehman Subtree Kernel for RTL Hierarchies

Require: RTL hierarchy trees $\mathcal{G}$, $\mathcal{G}'$, number of iterations H
Ensure: Kernel value $k_{\text{WL}}(\mathcal{G}, \mathcal{G}')$

$k_{\text{WL}} \leftarrow 0$ ▷ *Initialize kernel value*
$\ell^{(0)}(v) \leftarrow \text{moduleType}(v)$ for all nodes v ▷ *Set labels*
for $h \leftarrow 0$ to H **do**
  $\Sigma_h \leftarrow \emptyset$ ▷ *Initialize set for current iteration labels*
  if $h > 0$ **then** ▷ *Skip label update in first iteration*
    for each node v in $\mathcal{G}$ and $\mathcal{G}'$ **do**
      $M_v \leftarrow$ collect and sort labels of neighbors of v
      $\ell^{(h)}(v) \leftarrow \text{HASH}(\ell^{(h-1)}(v), M_v)$ ▷ *Generate new label*
    end for
  end if
  for each node v in $\mathcal{G}$ and $\mathcal{G}'$ **do**
    $\Sigma_h \leftarrow \Sigma_h \cup \{\ell^{(h)}(v)\}$ ▷ *Collect all unique labels*
  end for
  Initialize count vectors $c_h(\mathcal{G})$ and $c_h(\mathcal{G}')$ with zeros
  for each label $\ell \in \Sigma_h$ **do**
    $c_h(\mathcal{G})_\ell \leftarrow$ count occurrences of label ℓ in $\mathcal{G}$
    $c_h(\mathcal{G}')_\ell \leftarrow$ count occurrences of label ℓ in $\mathcal{G}'$
  end for
  $k^{(h)} \leftarrow \sum_{\ell \in \Sigma_h} c_h(\mathcal{G})_\ell \cdot c_h(\mathcal{G}')_\ell$ ▷ *Compute similarity*
  $k_{\text{WL}} \leftarrow k_{\text{WL}} + k^{(h)}$ ▷ *Add to total similarity*
end for
return k_{WL}

B. Weisfeiler-Lehman Subtree Kernel for RTL Hierarchy Comparison

To address the limitations discussed in Section I, we propose using the Weisfeiler-Lehman (WL) subtree kernel, which naturally operates on the RTL hierarchy trees generated during the compilation of Chisel code.

Given two RTL hierarchy trees $\mathcal{G}$ and $\mathcal{G}'$, represented as labeled graphs where nodes correspond to hardware modules and edges represent hierarchical relationships, the WL subtree kernel computes their similarity through an iterative relabeling and counting process as shown in Algorithm 1. Fig. 3 gives a visualized example of how the algorithm works.

The WL subtree kernel is particularly well-suited for microarchitecture design space exploration, as it both preserves the inherent structural information of hardware designs and operates directly on hierarchy trees. Through its iterative process, the kernel naturally captures similarities at various hierarchical levels, from individual components to complex module arrangements.

C. Theoretical Analysis of WL Subtree Kernel for Microarchitecture

To establish the advantages of our approach, we first analyze the sample complexity of both RBF and WL kernels through Theorems 1 and 2. Then, in Theorem 3, we demon-

strate that the WL subtree kernel exhibits lower sample complexity compared to the RBF kernel in practical applications.

Theorem 1 (RBF Kernel Sample Complexity)**.** *To learn with RBF kernel on d-dimensional Chisel parameter vectors with accuracy ε and confidence $1-\delta$, the sample complexity is:*

$$N_{RBF} = O(\frac{d}{\varepsilon^2}\log(\frac{1}{\delta})). \tag{3}$$

Proof. This follows from the classic result of [15] on sample complexity bounds for RBF kernels in terms of their Rademacher complexity. □

Theorem 2 (WL Kernel Sample Complexity)**.** *For a WL kernel with H iterations on RTL trees of maximum height h and branching factor b, the sample complexity is:*

$$N_{WL} = O(\frac{H\log(b^h)+\log(\frac{1}{\delta})}{\varepsilon^2}). \tag{4}$$

Proof. By the fundamental theorem of statistical learning, for a hypothesis class with Vapnik-Chervonenkis (VC) dimension [16] d, the sample complexity is $O(\frac{d+\log(\frac{1}{\delta})}{\varepsilon^2})$. The VC dimension of the WL feature space after H iterations is bounded by $O(H\log(b^h))$. □

Theorem 3 (Relative Efficiency)**.** *For typical RTL designs where $d \gg H\log(b^h)$, the WL kernel achieves better sample complexity than the RBF kernel.*

Proof. Modern processors often have hundreds of parameters ($d \approx 200$) while their RTL hierarchies typically have height $h \approx 10$ (GemminiSoC [9]) and branching factor $b \approx 4$. With $H = 3$ iterations:

$$H\log(b^h) \approx 3\log(4^{10}) \approx 42 < d. \tag{5}$$

Given that $\log(\frac{1}{\delta}) \gg 1$ typically holds, we can conclude that $N_{\text{WL}} < N_{\text{RBF}}$ for practical RTL designs. □

This theoretical analysis establishes the superior sample efficiency of our WL kernel approach, particularly for complex RTL designs with deep hierarchies.

V. RTL Hierarchy Aware Gaussian Process and Clustering

In this section, we present two applications of the kernel introduced in the previous section. We first describe how we enhance the basic WL kernel with module-specific weights derived from evaluation reports in Gaussian process regression. We then detail our clustering approach that leverages the kernel's similarity measure to enable efficient parallel design space exploration.

A. Gaussian Process Regression with Weighted WL Kernel for PPA Prediction

Building upon the WL subtree kernel introduced in the previous section, we propose a weighted extension that better captures the varying importance of different modules in power and area prediction. While the basic WL kernel treats all modules equally when computing structural similarities,

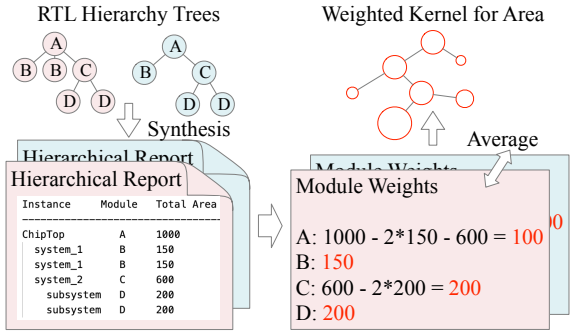


Fig. 4 Example of module weights' extraction in area prediction.

synthesis reports provide valuable information about each module's contribution to the overall PPA metrics.

Given a training dataset $\mathcal{D} = \{(\mathcal{G}_i, y_i)\}_{i=1}^n$, where $\mathcal{G}_i$ represents the RTL hierarchy tree and y_i denotes its corresponding PPA metric, we formulate the GPR model with the weighted WL subtree kernel as $f(\mathcal{G}) \sim \mathcal{GP}(0, k^w_{\text{WL}}(\mathcal{G}, \mathcal{G}'))$.

The weighted WL subtree kernel $k^w_{\text{WL}}(\mathcal{G}, \mathcal{G}')$ is enhanced by incorporating module-level importance weights:

$$k^w_{\text{WL}}(\mathcal{G}, \mathcal{G}') = \sum_{h=0}^{H} \lambda^h k^w_h(\mathcal{G}, \mathcal{G}'), \tag{6}$$

where $\lambda \in (0, 1]$ is a decay factor controlling the influence of higher iterations, and k^w_h is the weighted base kernel at iteration h:

$$k^w_h(\mathcal{G}, \mathcal{G}') = \sum_{\ell \in \Sigma_h} w(\ell) \cdot c^{(\ell)}_h(\mathcal{G}) \cdot c^{(\ell)}_h(\mathcal{G}'), \tag{7}$$

where module weights $w(\ell)$ are derived from synthesis reports, as shown in Fig. 4.

In Theorem 4, we demonstrate that the weighted WL kernel is semi-definite. Subsequently, in Theorem 5, we establish that semi-definite kernels are valid for use in Gaussian process regression models.

Theorem 4 (Positive Semi-definiteness of Weighted WL Subtree Kernel)**.** *The weighted WL subtree kernel $k^w_{WL}(\mathcal{G}, \mathcal{G}')$ is positive semi-definite for any non-negative weight $w(\ell)$.*

Proof. By expanding the weighted base kernel through Equation (7), we can prove that for any iteration h, the weighted base kernel $k^w_h(\mathcal{G}, \mathcal{G}')$ is positive semi-definite if $w(\ell) \geq 0$.

The weighted WL subtree kernel is defined as $k^w_{\text{WL}}(\mathcal{G}, \mathcal{G}') = \sum_{h=0}^{H} \lambda^h k^w_h(\mathcal{G}, \mathcal{G}')$, each k^w_h is positive semi-definite. Since $\lambda \in (0, 1]$, all coefficients λ^h are positive. Therefore, k^w_{WL} is positive semi-definite. □

Theorem 5. *The weighted WL subtree kernel defines a valid Reproducing Kernel Hilbert Space (RKHS) [17], ensuring the existence and uniqueness of the solution in the resulting Gaussian Process regression model.*

Proof. This follows directly from Moore-Aronszajn theorem [18], which establishes a one-to-one correspondence between positive semi-definite kernels and RKHSs. □

B. RTL Hierarchy Aware Design Space Clustering

The weighted WL subtree kernel provides a natural similarity measure that enables kernel k-means clustering [19] to identify distinct architectural patterns within the design space. This clustering capability serves two purposes: improving exploration efficiency and enabling parallel evaluation.

Given a target number of clusters k, the kernel k-means objective is formulated as:

$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{\mathcal{G}_i \in C_j} \|\phi_w(\mathcal{G}_i) - \mu_j\|^2, \tag{8}$$

where $\phi_w(\mathcal{G})$ is the implicit feature mapping induced by the weighted WL kernel.

The clustering is applied in two key steps:

1) Initial Design Space Sampling:

For initial design points selection, we select one representative design from each of the k clusters, where k is a hyperparameter. The selection strategy aims to identify designs that are most representative of their clusters' centers:

$$\mathcal{G}\text{init}^j = \operatorname{argmin} \mathcal{G} \in C_j |\phi_w(\mathcal{G}) - \mu_j|^2. \quad j = 1, \ldots, k \tag{9}$$

2) Parallel Expected Hypervolume Improvement:

During optimization, we enhance the Expected Hypervolume Improvement (EHVI) acquisition function [12] through cluster-based multiple suggestions generation:

$$\mathcal{G}^j_{\text{next}} = \operatorname*{argmax}_{\mathcal{G} \in C_j} \text{EHVI}(\mathcal{G}|\mathcal{D}), \tag{10}$$

then the parallel suggestions are evaluated concurrently.

This cluster-based parallelization strategy maintains exploration diversity while enabling efficient concurrent evaluation of distinct designs, particularly beneficial for time-consuming PPA evaluations.

VI. Experiment

A. Experimental Setup

We conducted our experiments on a Gemmini-based RISC-V SoC [9]. The design space of this SoC is illustrated in Fig. 5. We evaluate the performance using a suite of language models spanning different scales, including Transformer-small [20], Bert-base [21], as well as larger variants like Transformer-large, Bert-large, and GPT1-small [22].

The hardware implementation flow starts with Chipyard (version 1.9.1) for converting Chisel-based hardware descriptions into synthesizable RTL. For performance evaluation, we use Spike simulator to obtain cycle-accurate measurements of model inference latency. The physical implementation leverages Cadence Genus 201 with ASAP7 7nm process technology for logic synthesis, while power analysis is performed using Cadence Joules 201.

Our design space exploration considers multiple optimization targets encompassing cycles, power and area. The architectural search space is defined by 26 configurable parameters, theoretically enabling 1.3×10^{11} possible Chisel implementations. Through feasibility analysis sampling, we constructed a representative offline dataset comprising 5198 valid configurations for comprehensive evaluation.

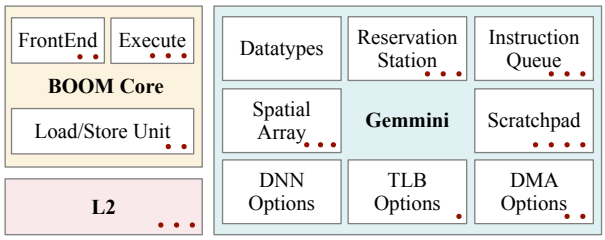


Fig. 5 Parameter selection of Gemmini-based RISC-V SoC. The blocks refer to sets of parameters controlling specific parts in the microarchitecture, while the dots refer to the number of parameters selected in our offline design space.

B. Comparison Against SOTA Methods

We compare our method with the following baselines for 100 times with each method assigned a total quota of 40 times evaluations (from chisel configuration to post-synthesis netlists, including performance simulation):

- DAC'16 [5]: A method combining statistical sampling and Adaboost-based active learning.
- ASPDAC'20 [6]: XGBoost-based method.
- DAC'23 [8]: A method that uses GNN to encode the design space into a latent space for Bayesian optimization.
- AAAI'24 [23]: Method based on reinforcement learning.

For fair comparisons between all baselines, hypervolume, which is one of the most widely used quality indicators in multi-objective optimization problems [24], is used as the indicator. A larger hypervolume indicates better exploration results. The following metrics are used to compare all methods: **HV** is the hypervolume of performance, power, and area. $\mathbf{HV}_{0,1}$ is the hypervolume of performance and power. $\mathbf{HV}_{0,2}$ is the hypervolume of performance and area. **MPP** is the maximum value of the product of the minimum clock period and the minimum power. **MPA** is the maximum value of the product of the minimum clock period and the minimum area. The results are normalized within the range of the shared initializing 5 data's scores to the highest scores. Evaluation quota is set as 40 for all methods.

We allocate the same evaluation quota rather than runtime to all methods because evaluation takes most of the runtime (more than 95%) in all these DSE methods. Giving the same evaluation quota eases the difficulty of understanding the experiment result.

TABLE I shows the comparison of our method with existing methods on Gemmini SoC while Fig. 6 gives the visualization. Given the same evaluation quota, our method achieves higher hypervolume, in other words, explores more design space than these methods. Our method is 29.3% higher than [5], 25.6% higher than [6], 12.9% higher than [8], 8.2% higher than [23] in hypervolume. The clustering algorithm facilitates parallel evaluation, allowing our method (shown by the orange curve) to achieve a k-fold reduction in total evaluation time, where k is the number of clusters, set as 3 in the experiment.

TABLE I Comparisons of Normalized DSE Performance against SOTA works.

Metrics	DAC'16 [5]	ASPDAC'20 [6]	DAC'23 [8]	AAAI'24 [23]	Ours w/o clustering	Ours w/ clustering
HV	0.775	0.796	0.884	0.928	0.967	**1**
HV$_{0,1}$	0.545	0.558	0.551	0.813	0.850	**1**
HV$_{0,2}$	0.553	0.557	0.689	0.833	0.870	**1**
MPP	0.814	0.763	0.907	0.904	0.942	**1**
MPA	0.899	0.791	0.976	0.917	0.974	**1**

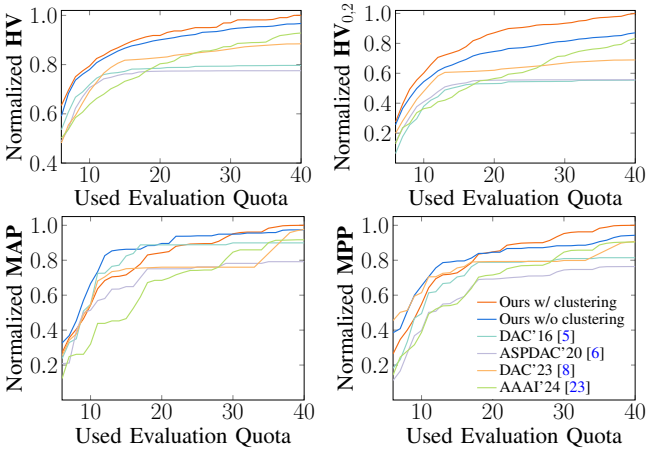


Fig. 6 The visualizations of normalized scores change with evaluations.

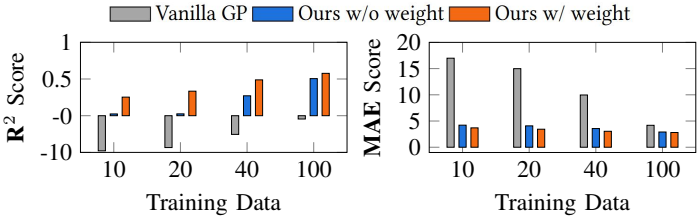


Fig. 7 Accuracies between the Vanilla GP model and our proposed method with small training set sizes. In the first chart, for better visualization, the bottom half of the y-axis is compressed by a factor of 10.

TABLE II Design Space's Clustering Results on PPA Space.

Metrics	K-means	RBF Kernel K-means	Ours
Silhouette	0.377	0.281	**0.597**
Davies-Bouldin	0.786	3.336	**0.473**

C. Ablation Studies

To rigorously evaluate our proposed approach, we conduct comprehensive ablation studies focusing on two key aspects: prediction accuracy under limited data conditions and clustering performance.

1) RTL Hierarchy Aware Gaussian Process:

We first investigate how our RTL hierarchy aware Gaussian Process performs under limited training data conditions, which is crucial for practical microarchitecture DSE. To show the proposed GP's performance in different stages of the exploration, we randomly select $(10, 20, 40, 100)$ training samples and evaluate the model's prediction accuracy using 500 samples from the remaining data as the test set. The test is repeated 200 times.

As the metrics of prediction accuracy, we use $\mathbf{R}^2$ (coefficient of determination) which measures the proportion of variance explained by the model, ranging from 0 to 1 where higher values indicate better fit, and **MAE** (Mean Absolute Error) which directly reflects the average prediction deviation.

Fig. 7 shows the average results of our proposed RTL hierarchy aware GP and Vanilla GP across different training set sizes. The results demonstrate that RTL hierarchy aware GP consistently outperforms the Vanilla GP model, particularly when training data is scarce.

2) RTL Hierarchy Aware Design Space Clustering:

We further evaluate the clustering capability of our approach compared to other clustering methods including standard k-means with RBF kernel. We assess the clustering quality using two widely adopted metrics: the **Silhouette** Score [25] which measures how similar an object is to its own cluster compared to other clusters (ranging from -1 to 1, higher is better), and the **Davies-Bouldin** Index [26] which evaluates intra-cluster similarity and inter-cluster differences (lower values indicate better clustering).

Considering that the inputs of our proposed method are graphs, while others are parameters, we compare the clustering results by testing the clustered labels with their corresponding PPA values, which is more practical in design space exploration. The number of clusters k is set as 3 and the test is repeated 50 times with different random states.

The results in TABLE II show that our RTL hierarchy aware Design Space Clustering consistently achieves better clustering quality. It proves that the proposed clustering method helps explore the design space, while naturally enabling parallel evaluation during the design space exploration.

VII. Conclusion

This paper presents a novel approach to microarchitecture design space exploration that fundamentally breaks away from traditional parameter-centric methods by leveraging the rich structural information contained in RTL hierarchy trees. We adopt WL subtree kernel to measure similarity of trees and theoretically prove that it is more efficient than RBF kernel in microarchitecture DSE. Gaussian process based on the weighted kernel extension incorporates the importance of a synthesis-derived module, improving exploration efficiency. An RTL hierarchy aware clustering method is developed to identify distinct architectural patterns and enable efficient parallel evaluation strategies. Experimental results and ablation studies on a Gemmini-based RISC-V SoC demonstrate the effectiveness of our approach, showing significant improvements over state-of-the-art methods.

REFERENCES

[1] J. Bachrach *et al.*, "Chisel: Constructing hardware in a scala embedded language," in *Proc. DAC*, 2012.

[2] C. Papon and Y. Xiao, "SpinalHDL." [Online]. Available: https://github.com/SpinalHDL/SpinalHDL

[3] M. Barros *et al.*, "Ga-svm feasibility model and optimization kernel applied to analog ic design automation," in *Proc. GLSVLSI*, 2007.

[4] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *Proc. HPCA*, 2007.

[5] D. Li *et al.*, "Efficient design space exploration via statistical sampling and adaboost learning," in *Proc. DAC*, 2016.

[6] Z. Xie *et al.*, "Fist: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *Proc. ASPDAC*, 2020.

[7] C. Bai *et al.*, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *Proc. ICCAD*, 2021.

[8] X. Yi, J. Lu, X. Xiong, D. Xu, L. Shang, and F. Yang, "Graph representation learning for microarchitecture design space exploration," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

[9] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *Proc. DAC*, 2021.

[10] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *Computational Intelligence and Bioinspired Systems*, J. Cabestany, A. Prieto, and F. Sandoval, Eds., 2005.

[11] J. Mockus, "The application of bayesian methods for seeking the extremum," *Towards global optimization*, 1998.

[12] M. T. Emmerich *et al.*, "Hypervolume-based expected improvement: Monotonicity properties and exact computation," in *Proc. CEC*, 2011.

[13] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, 2020.

[14] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels." *Journal of Machine Learning Research*, 2011.

[15] P. L. Bartlett and S. Mendelson, "Rademacher and gaussian complexities: Risk bounds and structural results," *Journal of Machine Learning Research*, 2002.

[16] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Learnability and the vapnik-chervonenkis dimension," *J. ACM*, vol. 36, no. 4, 1989.

[17] B. Schölkopf, "Learning with kernels: support vector machines, regularization, optimization, and beyond," 2002.

[18] N. Aronszajn, "Theory of reproducing kernels," *Transactions of the American mathematical society*, 1950.

[19] I. S. Dhillon, Y. Guan, and B. Kulis, "Kernel k-means: spectral clustering and normalized cuts," in *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[20] A. Vaswani *et al.*, "Attention is all you need," *Proc. NeurIPS*, 2017.

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[22] A. Radford *et al.*, "Improving language understanding by generative pre-training," https://openai.com/research/language-unsupervised/, 2018.

[23] C. Bai, J. Zhai, Y. Ma, B. Yu, and M. D. F. Wong, "Towards automated risc-v microarchitecture design with reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[24] M. Li and X. Yao, "Quality evaluation of solution sets in multiobjective optimisation: A survey," *ACM Comput. Surv.*, 2019.

[25] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, 1987.

[26] D. L. Davies and D. W. Bouldin, "A cluster separation measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979.

BitPattern: Enabling Efficient Bit-Serial Acceleration of Deep Neural Networks through Bit-Pattern Pruning

Gang Wang, Siqi Cai, Zhenyu Li, Wenjie Li, Dongxu Lyu, Yanan Sun, Jianfei Jiang, Guanghui He*
State Key Laboratory of Micro/Nano Engineering Science, School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai, China
{wangganginJSTU, caisiqi-2020, ambitious-lzy, wenjieli, lvdongxu, sunyanan, jiangjianfei, guanghui.he}@sjtu.edu.cn

***Abstract*—Bit-serial computation shows promise for accelerating deep neural networks (DNNs) by exploiting inherent bit sparsity. However, the original unstructured bit sparsity poses two major challenges for existing bit-serial accelerators (BSA): (1) workload imbalance from irregular bit distribution, and (2) inefficient memory access due to unpredictable non-zero bit locations. To address these issues, this paper proposes BitPattern, an algorithm/hardware co-design to efficiently accelerate bit-serial computation through bit-pattern pruning. At the algorithm level, we employ bit-pattern pruning to identify optimal combinations of predefined patterns and apply compression encoding to minimize weight storage. We further devise a pattern-similarity-based merging method to balance the bit-serial workload. At the hardware level, we co-design a bit-serial accelerator with a dedicated bit-pattern decoder and PE to leverage the potential of structured bit-pattern sparsity. The evaluation on several deep learning benchmarks shows that BitPattern can achieve $1.72\times$ memory reduction with negligible accuracy loss, and up to $2.11\times$ speedup and $1.86\times$ energy saving compared to state-of-the-art bit-serial accelerators.**

***Index Terms*—Bit-serial computation, DNN acceleration, Sparsity, Algorithm/hardware co-design, Compression**

I. Introduction

Deep neural networks (DNNs) have demonstrated remarkable success in various realms such as computer vision, natural language processing, and robotics. However, the superior performance of DNNs comes with rapid scaling of model size and computation complexity [1]. Such substantial burdens on memory and computation greatly hinder efficient DNN deployment in real-world scenarios, particularly on resource-constrained edge devices. Therefore, there is a strong need to prompt DNN efficiency from both software and hardware perspective.

Sparsification/pruning has been widely adopted in DNNs acceleration [2] to reduce the memory footprint and ineffective computations. Especially, structured pruning which targets spatially constrained non-zero value distribution of model weights can achieve a good tradeoff between model compression, workload balance, and hardware overhead. However, such benefits require a time-consuming retraining process and access to the full training dataset or specialized training techniques [3]. Unfortunately, state-of-the-art (SOTA) DNN retraining is impractical due to its prohibitive computational costs and limited access to original training datasets, particularly given commercial restrictions and privacy regulations.

This work was supported by the National Science and Technology Major Project of the Ministry of Science and Technology of China under Grant 2024ZD0802103 and the National Science Foundation of China under Grant 92464302. (*Corresponding author: Guanghui He).

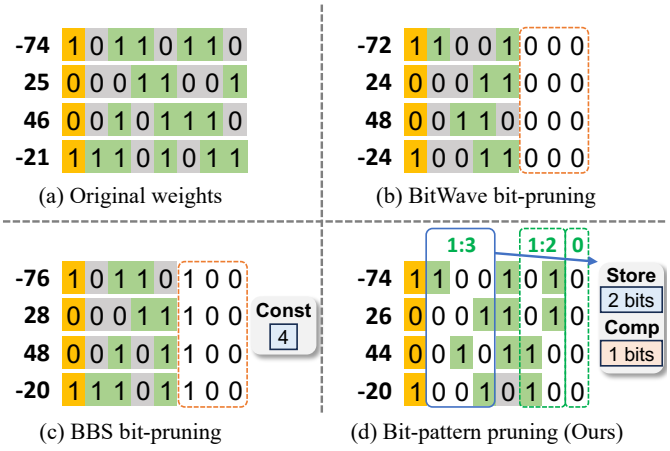


Fig. 1. Comparison of different bit-pruning methods for BSA. (a) Original weight values are represented in 2's complement. (b) Column-wise structured bit-pruning in BitWave [4] using sign-magnitude format. (c) Column-wise structured bit-pruning in BBS [5] using 2's complement format. (d) Bit-pattern pruning selectively prunes bits using predefined bit-patterns in sign-magnitude format. For example, a sparse bit-pattern (1:3) compresses the first 3 bits in current weight group into 2 storage bits per weight, requiring only 1 bit-serial computation cycle instead of 2, thereby reducing both memory consumption and computational workload.

Bit-serial computation, which can leverage inherent bit sparsity in model weights by skipping zero bits, is a promising solution to relieve the exhaustive training cost and enhance computation efficiency [5]. Several bit-serial accelerators (BSA) [6] have been proposed to exploit bit sparsity serially. However, inherent unstructured bit sparsity presents two major challenges for BSA: (1) workload imbalance and (2) inefficient memory access. Firstly, the irregular distribution of non-zero bits (NZB) in original weights induce unexpected large number of NZB, leading to significant workload imbalance and hardware underutilization, since the BSA performance is

fundamentally decided by the highest number of NZB. While prior works such as BitPruner [7] and Bit-Balance [8] attempt to address this issue by enforcing maximum NZB constraints, they incur substantial training overhead. Furthermore, due to the unpredictable locations of NZB in full data width, these methods fail to optimize model parameter footprint and exhibit inefficient memory access, which remain critical bottlenecks in DNN acceleration.

Structured bit pruning has emerged as a promising approach to achieve both workload balance and memory access efficiency in BSA by structurally pruning the bits in model weights. Recent works such as BitWave [4] and BBS [5] implement column-wise bit pruning, which systematically eliminates entire bit columns by converting all NZB to zero, enabling the exclusion of these columns from both memory storage and computation, as illustrated in Fig.1(b)(c). These approaches demonstrate superior performance compared to conventional BSAs. However, BitWave's fixed bit-position constraints in the column dimension suffer severe accuracy drop at high sparsity rates. BBS attempts to mitigate pruning errors through additional compensating constants which increases hardware complexity. Furthermore, neither approach optimizes the efficiency of remaining bit-serial computations, as they fail to skip zero-bit operations within compressed weights.

To tackle these problems, this paper proposes BitPattern, an algorithm/hardware co-design to efficiently accelerate bit-serial computation through bit-pattern pruning. The key insight of BitPattern is leveraging predefined bit-patterns with diverse sparse bit distribution to structurally compress weights and is capable of reducing bit-serial workload, offering better flexibility than column-wise bit pruning. For example, a sparse bit-pattern (1:3) compresses the first 3 bits per weight into 2 bits while requiring only 1 bit-serial computation cycle as shown in Fig.1(d). The main contribution of this paper are as follows:

1) We propose a bit-pattern pruning algorithm that identifies optimal pattern combinations through pattern fitting and error compensation, along with an efficient compression encoding scheme to minimize weight storage. We further introduce pattern-similarity-based merging to achieve reduced and balanced bit-serial workload.
2) We design a dedicated bit-serial accelerator with bit-pattern decoder and PE tailored with minimal hardware overhead via shared decoding patterns and direct shift mapping, harnessing the memory and computation advantages introduced by BitPattern.
3) Experiments on various DNN benchmarks are conducted. BitPattern achieves up to $2.11\times$ speedup and $1.86\times$ energy saving compared to SOTA bit-serial accelerators.

II. Background

A. Bit-serial Computation

Recent work [9] demonstrates abundant bit-level sparsity in quantized models with integer-based representation, enabling computation skipping for zero bits. This zero-skipping suggests a new computing paradigm, bit-serial computation, where non-zero bits are processed sequentially. A bit-serial PE computes the multiplication of an n-bit weight (W) with m-bit activation (X) as:

$$X \times W = \sum (-1)^{S_W} X(W_i << i), \tag{1}$$

where S_W is the sign of W. A set of BSA have been proposed to exploit bit sparsity. Pragmatic [6] utilize shift-and-add multiplications to skip zero bits while performing bit-serial computations. Bitlet [10] processes the serialized bits in a bit-interleaved manner and forward different bit-significances into separate shifter lanes. However, the performance of BSA is fundamentally determined by the highest number of NZB. The original weights typically exhibit irregular, unstructured bit sparsity with unexpected large number of NZB, leading to significant workload imbalance. To address this, Bit-Balance [8] use channel-wise quantization-aware training (QAT) to limit per-channel NZB. Nevertheless, this approach not only requires costly training process, but also retains weights at their original bit-width, failing to promote memory efficiency.

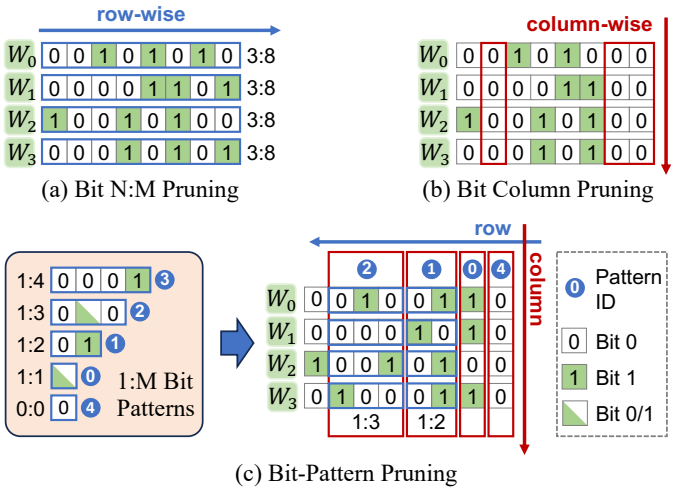
B. Bit Pruning

Model compression through pruning is a critical technique for reducing both memory footprint and computational complexity in DNNs. While conventional value pruning eliminates a portion of model parameters to enable compressed storage and skip redundant computations, its effectiveness in increasing zero-value density does not necessarily reduce the maximum number of NZB, which is crucial for BSA performance optimization.

In contrast, bit-level pruning offers a finer-grained approach that directly reduces bit-level computational workload of BSA. In particular, structured bit pruning systematically eliminates bit-serial operations while enabling efficient compression of remaining bits through structural patterns, thereby optimizing both computational throughput and memory utilization. Recent works have explored various approaches to structured bit pruning. BitWave [4] implements bit-column pruning across weight groups by iteratively eliminating NZB columns in sign-magnitude format, enabling efficient storage through non-zero column retention with minimal metadata overhead. Similarly, BBS [5] achieves compression by systematically pruning consecutive least significant bit-columns while maintaining accuracy through group-wise constant compensation. This paper introduces a novel structured bit-pruning methodology called bit-pattern pruning, which selectively eliminates bits based on predefined bit patterns for enhanced compression flexibility and efficiency.

III. Proposed Bit-Pattern Pruning

This section presents three key components in BitPattern algorithm: bit-pattern sparsity with compression encoding, output-oriented pruning algorithm, and pattern-similarity-based merging for workload balancing.



(a) Bit N:M Pruning (b) Bit Column Pruning (c) Bit-Pattern Pruning

Fig. 2. Three structured bit pruning strategies are presented: (a) Bit N:M pruning balance the row-wise bit numbers within individual weights, e.g. 3 NZB in 8-bit weight. (b) Bit column pruning systematically eliminates NZB in a column-wise manner across multiple weights. (c) Bit-pattern pruning employs multiple 1:M bit sparse patterns, partitioning weights into column-wise segments and selectively pruning them using the most fitted row-wise patterns.

A. Bit-Pattern Sparsity and Compression Encoding

To generate more bit sparsity and achieve computational efficiency in BSA, three structured bit pruning strategies can be employed, as illustrated in Fig.2. Fig.2(a) demonstrates a structured bit-pruning approach which maintains a balanced row-wise distribution of NZB, as implemented in BitBalance [8]. This approach is referred to as bit N:M[1] pruning due to its fine-grained N:M sparsity characteristics. While bit N:M pruning effectively reduces NZB, it fails to compress memory footprint. Compressing N:M sparsity needs to store the non-zero elements and an additional mask to indicate the non-zero positions. This strategy becomes impractical when applied to bit-level N:M sparsity, as it would require per-data bit masking. Fig.2(b) depicts bit column pruning, which achieves model compression by eliminating entire bit columns across weight groups and storing only remaining columns with compact metadata for pruned positions. However, this rigid column-wise sparsity significantly degrades model accuracy at high sparsity rates and requires full bit-serial processing of remaining columns.

Fig.2(c) presents our bit-pattern pruning method, which leverages structured bit-pattern sparsity through predefined patterns. We utilize multiple 1:M bit patterns where each pattern specifies 0 or 1 NZB within an M-bit column-wise weight segment. For instance, as shown in Fig.2(c), the weight segment of bit[6:4] employs a sparse bit-pattern (1:3) with pattern ID ❷. Our approach incorporates three pattern types: sparse bit-pattern for general compression, dense bit-pattern ❶ (1:1) to preserve critical bit distributions (e.g., bit[1] in Fig.2(c)), and zero bit-pattern ❹ (0:0) for high compression in least significant columns (e.g., bit[0] in Fig.2(c)). Notably, the bit column pruning represents a special case of our method,

[1]N non-zero bits in a consecutive block of M bits, where N $\leq$ M.

TABLE I
THE COMPRESSION ENCODING VALUE TABLE OF BIT PATTERNS.

ID	Patterns	Original Values(W)	Compressed Values(W')	Bit Compression	Compressed Encoding
0	1:1	[0, 1]	[0, 1]	-	$0/\log_2 W + 1$
1	1:2	[1, 2]	[0, 1]	$2 \rightarrow 1$	$\log_2 W$
2	1:3	[0, 1, 2, 4]	[0, 1, 2, 3]	$3 \rightarrow 2$	$0/\log_2 W + 1$
3	1:4	[1, 2, 4, 8]	[0, 1, 2, 3]	$4 \rightarrow 2$	$\log_2 W$

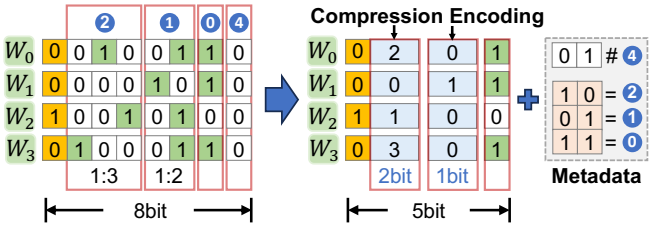


Fig. 3. An example for compression encoding of bit-pattern sparsity which prunes 3 bits per weight. A group of original weights are compressed into pattern specific encoding and metadata. The metadata contains (1) the number of least significant zero columns and (2) pattern IDs, where each pattern ID is encoded using 2 bits. The weights are stored in sign-magnitude format.

utilizing only dense and zero bit-patterns. This comprehensive design enables simultaneous optimization of row-wise workload balance and column-wise compression, effectively addressing the trade-off between computational efficiency and memory footprint.

Table I illustrates our compression encoding values of bit patterns. For instance, the sparse bit-pattern ❸ (1:4) can efficiently compress the 4-bit original value (1,2,4,8) into a 2-bit encoding (0,1,2,3), achieving 2-bit compression. The encoding values are computed using $\log_2 W$, which not only provides optimal encoding efficiency but also enables direct compatibility with BSA shifter operations (e.g., $8 = 1 << \log_2 8$). Notably, all four bit-patterns in Table I maintain this $\log_2 W$ encoding property and require single bit-serial computation, facilitating hardware-efficient bit manipulation in BSA. Fig.3 demonstrates our compression encoding scheme with a bit-pattern combination including different bit-patterns. Given a combination, weight groups are compressed using pattern-specific encodings and metadata, which contains pattern IDs and zero-column number. Particularly, zero bit-pattern ❹ are restricted to consecutive least significant columns, as eliminating higher significant columns exponentially impacts model accuracy. This approach enables efficient 2-bit pattern ID encoding for 4 bit-patterns, resulting in 8-bit metadata in this example. When more than three patterns are needed, we limit patterns to ❶ and ❷ to maintain consistent 8-bit metadata.

B. Output Oriented Bit-Pattern Pruning

To optimize the combinations of predefined bit-patterns which best fit the model weights, bringing larger bit-pattern sparsity and minimal pruning error, we propose an output-oriented bit-pattern pruning algorithm (Algorithm 1). Given a quantized weight matrix W and a per-weight pruning bit

Algorithm 1: Output Oriented Bit-Pattern Pruning.

Input : W: Quantized Weight, G: Group size, P: bit patterns, N: number of pruning bits
Output: $\hat{W}$: Compressed weights, M:medadata

1 W_r = W.reshape(-1,G) ▷ step 1
2 Pattern Candidates: $C \leftarrow \emptyset$
3 Generate all pattern combinations: PC = comb(P)
4 **for** combination $b \in PC$ **do**
5 **if** $Pbit_b = N$ **then**
6 $C = C \cup b$
7 Y = zeros_like(W_r) ▷ step 2
8 **for** Group $g \in [0 : G]$ **do**
9 bestError = ∞
10 **for** combination $c \in C$ **do**
11 All possible value V under c
12 Error = $|W[g] - V \otimes W[g].sign|^2$
13 minError = min(Error)
14 **if** sum(minError) $<$ bestError **then**
15 bestError = sum(minError)
16 $Y[g] = c$
17 Y = reshape(Y,$W.shape$) ▷ step 3
18 $\hat{W}$ = zeros_like(W)
19 **for** column j in W.shape[1] **do**
20 $\hat{W}_{:,j}$ = Pattern-constraint value search($W_{:,j}$, $Y_{:,j}$)
21 $W_{:,j:}$ = Error compensate($\hat{W}_{:,j}$)
22 M = { Y.numZeroColumns, Y.IDs }

number N, the algorithm first divides weights into groups and generates all feasible bit-pattern combinations with varying zero-column number, whose compressed bit-count equals N. In the second step, we determine optimal bit-pattern combinations for each weight group. A Kronecker Product $\otimes$ is used to generate all possible values associated with C. We compare generated values with the weight group to find the best matched values, and further sum up the errors the decide the group error for current combination c. This process is iterated through all combinations to search optimal pattern combinations Y that fit the pruned weights for each group.

However, step 2 optimize pruned weights independently and may not guarantee minimal output error in $O = WX$. Therefore, we further propose an method to identify optimal bit-pattern combinations that can minimize $\|WX - \hat{W}X\|_2$ in step 3. Building upon the column-wise optimization framework introduced in GPTQ [11], our method iteratively determines optimal compressed weights and corresponding pattern combinations on a per-column basis, and compensates the pruning error to subsequent weight columns. Note that the weight column dimension (step 3) is orthogonal to grouping dimension (step 1). Incorporating group-wise pattern optimization within step 3 would severely degrade potential parallelism. To balance computational efficiency and pruning effectiveness, we constrain the search space of each weight value to the

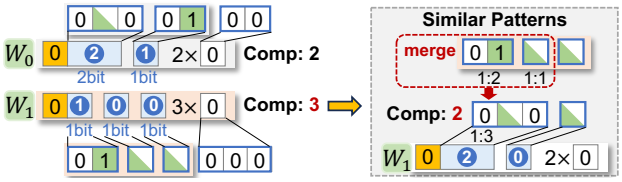


Fig. 4. Pattern-similarity-based merging example. Original W_1 compressed weight has three bit patterns which results in 3 bit-serial computation workload. Based on the pattern similarity, we can merge the first 2 patterns, 1:2 ❶ and 1:1 ⓪, into 1:3 ❷, which only involve 2 bit-serial workload.

determined optimal patterns Y and perform the column wise value search similar to step 2 (line 20 in Algorithm 1). The pruned error are further compensated following [11]. Our pruning algorithm is free of training and can be directly integrated with quantization frameworks.

C. Pattern-Similarity-Based Merging

Despite employing bit-pattern pruning, bit-serial computations with compressed weights can still face workload imbalance during hardware acceleration. Fig.4 illustrates this challenge with two weight groups: W_1 requires two bit-serial computation cycles due to its two bit-patterns, while W_2 demands three cycles, resulting in workload imbalance. To resolve this inefficiency, we introduce pattern-similarity-based merging, which combines adjacent bit-patterns into a similar pattern while preserving the original bit-width. As shown in Fig.4, pattern ❶ (1:2) and pattern ⓪ (1:1), initially compressing 3 bits and stored in 2 bits, are merged into a single pattern ❷ (1:3), achieving two-cycle bit-serial computations. As such, the bit-serial workload is balanced and reduced from 3 to 2. While this merging process introduces pruning errors, they can be effectively compensated through step 3 of Algorithm 1. Furthermore, the pattern merging mechanism can also be efficiently implemented by refining similar pattern combinations in step 1 of Algorithm 1.

IV. Hardware Architecture

In this section, we introduce the BitPattern accelerator architecture. Then, the bit-pattern decoder and corresponding bit-serial PE are proposed, which can efficiently exploit the bit sparsity after bit-pattern pruning.

A. Overall architecture

The overall BitPattern accelerator architecture is illustrated in Fig.5. The accelerator adopts an 32×32 bit-serial PE array with output-stationary dataflow, where each PE have 8 input lanes. The weights are broadcast vertically and the activations are broadcast horizontally. The input data are read from the DRAM interface to on-chip buffers and the output are stored in output buffers. BitPattern incorporates a metadata buffer to store the metadata after bit-pattern pruning. For each PE column, a BitPattern decoder is employed to decompress the compressed weights with metadata and generate corresponding bit-serial inputs.

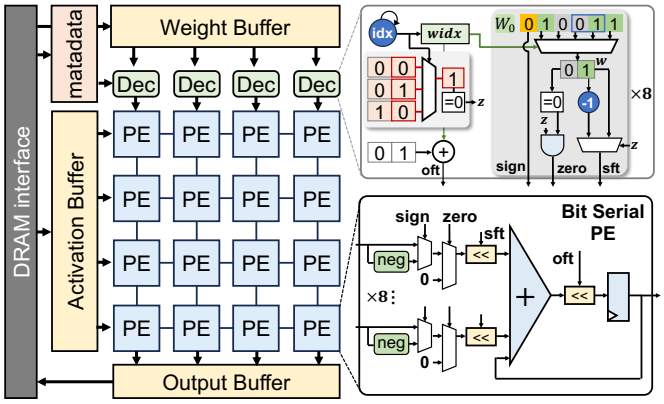


Fig. 5. BitPattern accelerator architecture.

B. BitPattern Decoder

The BitPattern decoder is shown in Fig.5, consisting a metadata scheduler and 8 weight decoder for each PE input lane. An inner index register is used to schedule current pattern ID and weight bit-index $widx$. In each cycle, 1 bit-pattern is selected to decoding the weight segment w at current $widx$ into the shift values used in bit-serial PE. The compression encodings exhibit identical property between pattern ⓪/② and ①/③ according to Table I. Therefore, efficient encoding scheme discrimination is enabled through the least significant bit of pattern ID. Noticing that the compressed encoding values follow a systematic value of $\log_2 W$, $\log_2 W + 1$, or 0, where W represents the original weight segment, and the shift operations are inherently based on $\log_2(W)$. This structural relationship enables the shift values used in bit-serial PE to be directly extracted from compressed weights, requiring only a lightweight "-1" subtractor to compensate for the "+1" offset in compression encoding. Consequently, the selected pattern ID's least significant bit is compared with 0 to generate corresponding zero and shift values. Finally, the weight bit-index $widx$ is added with zero-column number to generate the common offset. Therefore, the BitPattern decoder exhibits decoding efficiency and minimal hardware through shared decoding patterns and direct mapping between compressed values and shift operations.

C. Bit-Serial PE

Bit-serial PE comprises of multiplexers, adders, shifters and accumulators to accomplish bit-serial computation. In each cycle, 8 activations are inputted and first perform the bitwise multiplication with signs and zeros signal from BitPattern decoder. The results are further shifted by the encoded value in a specific bit-pattern. After that, all shifted values are accumulated and shifted by the common offset. Since the bit range in each pattern is constrained, the first-stage shifters offer low hardware overhead. Fig.6 illustrates our bit-serial dataflow with bit-pattern decoder. In cycle 0, the first pattern ID ⓪ is selected and the least significant of compressed weights are further decoded to the corresponding zero and shift values for bit-serial PE. While in cycle 1, the next pattern ② is used to decode the weight segment of bit[2:1].

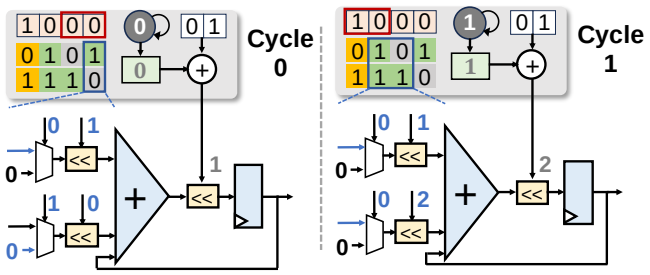


Fig. 6. An example of the bit-serial dataflow with 2 bit-patterns.

TABLE II
COMPARISON OF ACCURACY AND WEIGHT BIT-WIDTH BETWEEN BITPATTERN (BITP), BBS AND BITWAVE.

	ResNet-18		ResNet-34		ResNet-50		ViT-S		ViT-B	
	Acc	Bits	Acc	Bits	Acc	Bits	Acc	Bits	Acc	Bits
Float	69.76	-	73.29	-	76.14	-	81.39	-	84.53	-
BitWave	69.12	6.6	72.47	5.5	75.11	5.9	80.02	5.5	83.54	5.8
BBS	69.17	5.2	72.81	5.2	75.60	5.1	**80.23**	**5.1**	83.61	5.0
BitP	**69.40**	**4.5**	**72.97**	**4.3**	**75.66**	**4.4**	**80.06**	**5.1**	**83.65**	**4.9**
BitP-M	**69.27**	**4.5**	**72.93**	**4.3**	**75.61**	**4.3**	79.88	5.1	**83.65**	**4.9**

V. EVALUATION

A. Methodology

We evaluate BitPattern's effectiveness on various DNN models using ImageNet-1K [12], including ResNet [13] and Vision Transformer (ViT) [14]. Models are sourced from HuggingFace and PyTorch libraries and quantized to 8 bits. Our hardware evaluation compares the BitPattern accelerator against both the SOTA value-based accelerator ANT and bit-serial accelerators including Pragmatic, Bitlet, BitWave, and BBS (BitVert). The target number of pruning bits are set to 4. To achieve negligible accuracy loss, we retain 20% of the most sensitive channels and the remaining with 4 bit-columns pruned as detailed in [5] for BBS. For BitWave, 30% - 60% of the sensitive channels are retained, following [5]. For BitPattern, we apply a 2-bit pruning to the preceding layers to achieve a good tradeoff between model accuracy and inference efficiency under negligible accuracy loss.

All design results are implemented in SystemVerilog and synthesized using Design Compiler (DC) with TSMC 28nm technology at a clock frequency of 800 MHz. Power consumption is estimated using Synopsys PrimeTimePX. We implement a cycle-accurate simulator based on [15], with SRAM buffers modeled using CACTI [16] and DRAM memory access energy set to 1.2 pJ/bit, as reported in [17]. All acceleration arrays are configured with 1024 PEs and the same SRAM size.

B. Accuracy Comparison

Table II compares the accuracy and compressed weight bit-width of BitPattern against structured bit-pruning approaches in BBS and BitWave. The bit-width are calculated as the average bits per weight across all model layers, including associated metadata overhead. We denote **BitP-M** as BitPattern with pattern-similarity-based merging, which achieves more

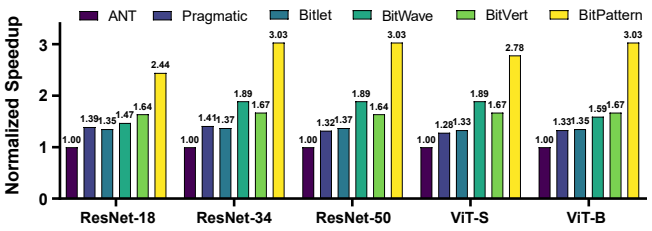


Fig. 7. The normalized speedup of different designs.

TABLE III
AREA AND POWER BREAKDOWN FOR BITPATTERN.

BitPattern	Configuration	Area(mm^2)	Power(mW)
PE	32×32	0.839	460.80
Decoder	32	0.007	5.26
Weight Buffer	128KB	0.426	104.96
Activation Buffer	256KB	0.832	145.41
Output Buffer	128KB	0.850	185.75
Metadata Buffer	8KB	0.027	6.56
Total	-	**2.980**	**908.73**

balanced bit-pattern distribution. BitWave requires a large portion of sensitive channels with higher bit-widths to maintain acceptable accuracy, primarily due to its highly dependency on natural zero-bit columns in original weights. While BBS achieves higher accuracy through constant compensation, it fails to exploit compression opportunities beyond least significant columns. Thanks to the flexibility of bit-pattern sparsity and proposed output oriented pruning, BitPattern achieves higher accuracy and lower weight bit-width across most cases. Specifically, it attains 1.72× and 1.73× model memory size reduction with an average accuracy loss of 0.67% and 0.75% under basic and pattern-similarity-based merging variants, respectively.

C. Hardware Comparison

We compare the speedup, energy consumption and efficiency with existing accelerators. The area and power breakdown of BitPattern are shown in Table III.

Fig. 7 presents normalized speedup comparisons between BitPattern and competing accelerators. While ANT reduces data bit-width through multi-type quantization, it fails to exploit bit sparsity. Both Pragmatic and Bitlet suffer from workload imbalance, limiting their acceleration efficiency. BitWave's higher retention of sensitive channels increases both computation and memory access cycles. Although BitVert reduces sensitive channel count, it requires an additional cycle for sign bit processing in 2's complement format. BitPattern achieves superior performance through pattern-similarity-based merging, requiring fewer and more balanced bit-serial workloads at equivalent compression width. Overall, our approach demonstrates significant speedups: average 2.86×, 2.13×, 2.11×, 1.66×, and 1.71× speedup over ANT, Pragmatic, Bitlet, BitWave and BitVert.

Fig.8 shows the energy consumption across different designs. BitPattern's effective bit-pattern pruning reduces DRAM and buffer access energy, achieving 1.25× lower access energy versus ANT. With pattern merging, our design demonstrates 1.82× and 1.73× core energy reductions compared to BitWave and BitVert. Overall, BitPattern achieves average energy savings of 1.31×, 1.77×, 1.86×, 1.49×, and 1.51× compared to ANT, Pragmatic, Bitlet, BitWave and BitVert, respectively.

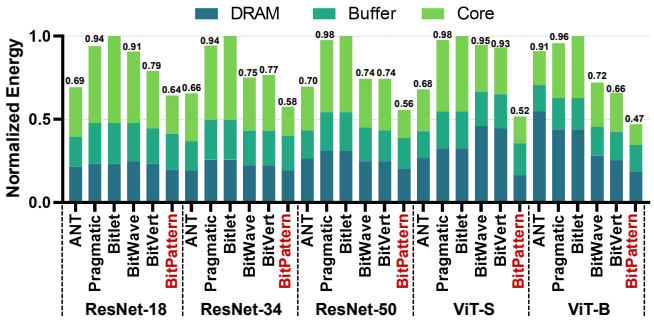


Fig. 8. The normalized energy breakdown of different designs.

TABLE IV
PE COMPARISON BETWEEN BITPATTERN AND PRIOR ACCELERATORS.

Design	PE Area (μm^2)	Power (mW)	Norm. Perf[1]	Norm. Perf/Area	Norm. Perf/Power
ANT	309.9	0.23	1.0×	1.00	1.00
Pragmatic	923.1	0.51	1.5×	0.50	0.68
Bitlet	1665.6	0.57	1.4×	0.26	0.56
BitWave	702.4	0.49	1.9×	0.84	0.89
BitVert	739.6	0.45	1.7×	0.71	0.87
BitPattern(Ours)	**819.5**	**0.45**	**4.0×**	**1.51**	**2.04**

[1] Normalized performance refers to the number of operations that can be executed on average per cycle in a layer.

Table IV compares the power and area efficiency between BitPattern and existing BSA at PE level. While all bit-serial PEs use 8 input lanes compared to ANT's single 6-bit lane, BitWave and BitVert achieve efficiency by sharing one shifter across lanes, though BitVert requires additional BBS constant logic. Although BitPattern PE utilizes lightweight per-input shifters resulting in slightly larger area compared to BitWave and BitVert, it can efficiently skip zero-bit computations based on the predefined bit-patterns while BBS and BitWave can not skip zero-bit computations in the compressed weights. As such, BitPattern PE achieves 1.51×, 1.80×, and 2.13× improvements in area efficiency compared to ANT, BitWave, and BitVert.

VI. CONCLUSION

We present BitPattern, an algorithm/hardware co-design for efficient DNN bit-serial acceleration. Our approach combines structured bit-pattern pruning for optimal bit distribution with a specialized hardware architecture featuring dedicated bit-pattern decoders and processing elements. Evaluations demonstrate that BitPattern achieves up to 2.11× speedup and 1.86× energy reduction compared to SOTA bit-serial accelerators through simultaneous optimization of memory access and computation efficiency.

REFERENCES

[1] A. Gholami *et al.*, "Ai and memory wall," *IEEE Micro*, 2024.

[2] Y. N. Wu *et al.*, "HighLight: Efficient and Flexible DNN Acceleration with Hierarchical Structured Sparsity," in *Proc. 56th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2023, pp. 1106–1120.

[3] C. Fang *et al.*, "An Algorithm-Hardware Co-Optimized Framework for Accelerating N:M Sparse Transformers," *IEEE Trans. VLSI Syst.*, vol. 30, no. 11, pp. 1573–1586, 2022.

[4] M. Shi *et al.*, "BitWave: Exploiting Column-Based Bit-Level Sparsity for Deep Learning Acceleration," in *Proc. IEEE Int. Symp. High-Perform. Comput. Archit. (HPCA)*. IEEE, 2024.

[5] Y. Chen, J. Meng *et al.*, "BBS: Bi-directional Bit-level Sparsity for Deep Learning Acceleration," *arXiv:2409.05227*, 2024.

[6] J. Albericio *et al.*, "Bit-pragmatic deep neural network computing," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2017, pp. 382–394.

[7] X. Zhao *et al.*, "BitPruner: Network pruning for bit-serial accelerators," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*. IEEE, 2020, pp. 1–6.

[8] W. Sun *et al.*, "Bit-balance: Model-Hardware Co-design for Accelerating NNs by Exploiting Bit-level Sparsity," *IEEE Trans. Comput.*, 2023.

[9] G. Wang *et al.*, "BSViT: A Bit-Serial Vision Transformer Accelerator Exploiting Dynamic Patch and Weight Bit-Group Quantization," *IEEE Trans. Circuits Syst. I*, vol. 71, no. 9, pp. 4064–4077, 2024.

[10] H. Lu *et al.*, "Distilling Bit-level Sparsity Parallelism for General Purpose Deep Learning Acceleration," in *Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2021, pp. 963–976.

[11] E. Frantar *et al.*, "GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers," *arXiv:2210.17323*, 2022.

[12] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *IEEE Conf. Comput. Vis. Pattern Recognit.* IEEE, 2009, pp. 248–255.

[13] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

[14] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," *arXiv:2010.11929*, 2020.

[15] C. Guo *et al.*, "ANT: Exploiting Adaptive Numerical Data Type for Low-bit Deep Neural Network Quantization," in *Proc. 55th Annu. IEEE/ACM Int. Symp. Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

[16] N. Muralimanohar *et al.*, "Cacti 6.0: A tool to model large caches," *HP laboratories*, vol. 27, p. 28, 2009.

[17] M. O'Connor *et al.*, "Fine-grained DRAM: Energy-efficient dram for extreme bandwidth systems," in *Proc. 50th Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2017, pp. 41–54.

Local-GS: An Order-Independent Gaussian Splatting Training Accelerator Exploiting Splat Locality

Yiyang Sun, Qinzhe Zhi, Yiqi Jing, Le Ye, Ru Huang and Tianyu Jia*
School of Integrated Circuits, Peking University, Beijing, China
*Corresponding Email: tianyuj@pku.edu.cn

***Abstract*—3D Gaussian Splatting has emerged as the SOTA approach for 3D representation and view synthesis. While Gaussian Splatting has demonstrated impressive capability and rendering quality on desktop GPUs, achieving on-demand training on resource-constrained edge devices is still challenging. In this work, we identified the training bottleneck from a few perspectives including algorithm splat locality and the limited memory and hardware under-utilization. To address these problems, we present Local-GS, a 3D Gaussian Splatting training accelerator with order-independent rendering to break the depth-wise data dependency between overlapping Gaussians. We further incorporate a parallel pixel intersection test unit to schedule thread workload based on Gaussian splat locality and improve hardware utilization. A set of unified training-rendering cores are designed to achieve efficient splat-level parallel rendering and gradient propagation. Our Local-GS is implemented in 7nm and is evaluated by several real-world 3D scenes. Compared to edge Jetson NX GPU, Local-GS achieve 26.9-53× training speedup and three orders of magnitude efficiency boost.**

***Index Terms*—Gaussian Splatting, 3D, Training, Accelerator**

I. Introduction

Novel view synthesis (NVS) is a widely studied computer vision task to generate photorealistic images from unobserved locations for a given 3D scene. Such generation tasks are highly desired in many consumer-level edge applications to provide a real-time seamless experience, such as AR/VR [1], live-streaming [2], and autonomous driving [3]. Traditional NVS algorithms [4] perform image synthesis based on 3D representations of meshes, voxels, or point clouds and leverage rasterization and ray tracing to generate object models, followed by output synthesized views. However, the conventional approach is computationally costly while it is challenging to achieve global consistency for high-quality rendering.

In recent years, learning-based NVS algorithms utilize implicit representation [5] and neural network models to learn and reconstruct 3D objects with remarkable quality. In particular, neural radiance field (NeRF) [6] is the most influential learning-based method that uses volume rendering with implicit neural scene representation to learn the geometry and lighting of a 3D scene. Although NeRF can offer a robust representation of 3D scene geometry, the challenges of high computational demand persist. 3D Gaussian splatting (3D GS) [7] is an emerging powerful technique for efficient and real-time NVS. Compared to NeRF, 3D GS is an emerging non-AI workload that portrays a 3D scene using numerous 3D Gaussian primitives or ellipsoids. Each ellipsoid in 3D GS is equipped with position, orientation, scale, opacity, and color information. To render these ellipsoids, they undergo a conversion from 3D to 2D space and are strategically organized for optimal rendering. Compared to NeRF, 3D GS can achieve very fast convergence speed for training, e.g. 15-20 min for a 1M Gaussian scene on RTX 3090, and photorealistic NVS rendering in real time, e.g. 100 FPS on RTX 3090 GPU and 60 FPS on laptop RTX 3060 GPU.

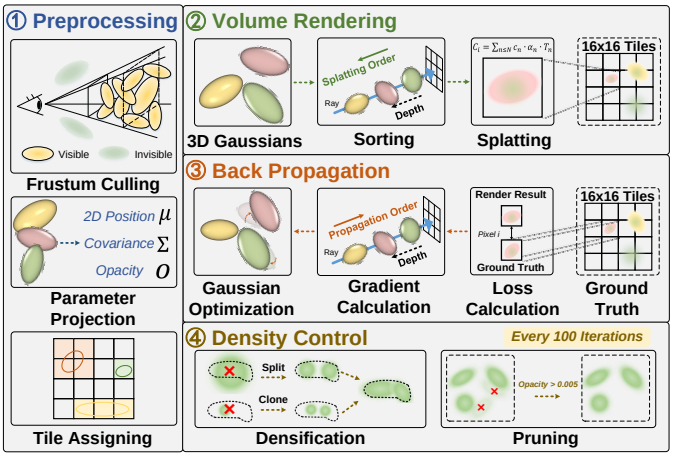


Fig. 1. Overview of 3D Gaussian Splatting operations.

Fig. 1 shows the main operation steps for 3D GS, including Gaussian preprocessing, volume rendering, loss backpropagation, and adaptive density control. Preprocessing transforms the 3D features of Gaussian ellipsoids into 2D, including frustum culling and tile assigning based on their scale and position. In volumetric rendering, the ellipsoids are "splatted" onto the camera plane in ascending order of depth to merge their 2D footprints into the rendered image and faithfully represent occlusion between transparent Gaussians. The back propagation training of GS begins with color loss calculation between ground truth images and rendered results, followed by a reversed rendering process, i.e. rendering Gaussians in the descending order of depth to peel off occlusion effects caused by former Gaussians. As for the ill-represented Gaussians with a large positional loss, 3D GS adopts a dynamic density control strategy, i.e. densification and pruning, to adjust the ellipsoid number and accurately represent the scene fine details.

Although 3D GS exhibits promising results for both training and rendering, it still faces considerable difficulty for the

deployment on mobile hardware. For instance, training a 1M Gaussian scene consumes 6 hours, and rendering a novel view will take 120-160 ms on edge platform Jetson Xavier NX. Considering that most NVS tasks are intended to be deployed on mobile devices, such lengthy training and rendering time is unacceptable for the applications.

Since 3D GS was first present in 2023, there are limited attempts to accelerate 3D GS via hardware implementations. For example, [8] presented a hardware acceleration unit to execute the rendering pipeline of 3D GS with algorithmic optimizations. [9] developed an accelerator for 3D GS-based SLAM rendering task. However, the existing works only focused on the rendering process, while the training is much more complicated for edge NVS deployment. In addition, the pre-trained NVS models often have poor versatility, i.e. render-only accelerators cannot make on-demand responses to the varying scenes. Hence the training capability at edge is critical to enable fast scene modeling to adapt to new environments for applications such as AR/VR and autonomous driving.

In this paper, we present Local-GS, a hardware-software co-designed accelerator for 3D GS training and rendering. Our work utilized the spatial locality of Gaussian splats to efficiently reschedule forward and backward rendering workload, as well as breaking the rendering order dependency to eliminate resource constraints on edge hardware. Implemented with 7nm technology, Local-GS achieves 26.9-53× training speedup compared with Jetson NX and 3.0-4.8× improvement compared to a RTX 3090 GPU across diverse scenarios. Significant energy efficiency boost is also obtained.

The contributions of this work are summarized as follows:

- We analyze the emerging 3D GS workload, and identify the memory and computation challenges due to the transparency dependencies in back propagation.
- We leverage a depth-guided order-independent blending scheme for parallel multi-Gaussian rendering and gradient propagation with significant performance gain.
- We develop a unified training-rendering digital architecture leveraging splat locality to achieve workload scheduling and high hardware utilization.
- We implement Local-GS accelerator using 7nm and evaluate it using a set of real-world benchmark, showing significant performance and efficiency gains.

II. Background

A. Preliminaries of 3D Gaussian Splatting

3D Gaussian Splatting starts with a set of images to generate a sparse initial point cloud through structure from motion (SfM) [10]. As shown in Fig. 1, a set of ellipsoid-shaped Gaussian primitives is created based on these points, where each Gaussian ellipsoid is represented by a three-dimensional Gaussian distribution with position(μ), scale(s), rotation(r), opacity(o) and spherical harmonics (sh) for color representation. After initialization, 3D GS repetitively iterates through all camera sets to refine the 3D model with the following operation steps: preprocessing, volume rendering, back propagation and density control.

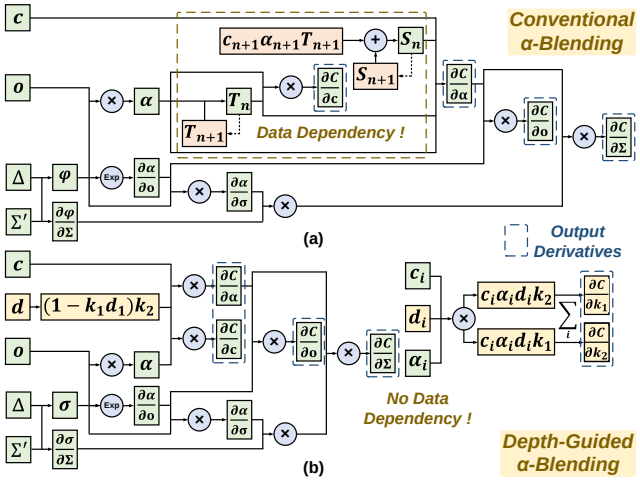


Fig. 2. Gradient computation in (a) conventional 3D-GS training, (b) 3D-GS training with depth-guided α blending.

Preprocessing of the given 3D Gaussian set starts by frustum culling the Gaussians of the scene outside the viewable area to focus on the Gaussians that need to be rendered. After obtaining visible Gaussians, the 3D features of Gaussians are converted into 2D position (μ), covariance matrix(Σ), opacity(o) on the projection plane. Meanwhile, a rectangular boundary is defined based on the projected Gaussian footprints to calculate their overlapping area with the screen. The Gaussians are then duplicated and assigned to the intersected image tiles (16×16 pixels) for tile-wise depth sorting to obtain correct color accumulation.

Volume rendering is carried out by tile-wise after preprocessing. The process begins by determining how much a Gaussian contributes to a certain pixel with its inherit transparency (α), and the accumulated transmittance (T) through the former-rendered Gaussians. The α value is calculated as following:

$$\alpha_i = o_i \cdot \exp\left(-\frac{1}{2}\Delta^T\Sigma^{-1}\Delta\right) = o_i \cdot \exp(\varphi) \tag{1}$$

where Δ is the position vector between the center of Gaussian and pixel and o_i representing Gaussian opacity. T is determined through $T_i = \Pi(1-\alpha_i)$ across previously rendered Gaussians. The per-Gaussian contributions are accumulated, i.e. called α blending process, in ascending order of depth across all Gaussians in the tile to generate the pixel colors:

$$C = \Sigma T_i c_i \alpha_i \tag{2}$$

Back propagation optimizes the Gaussian distribution through the difference or loss (L) between rendered images and the ground truth images. 3D GS adopts direct L_1 loss and structural D-SSIM loss to measure the difference. The loss is then back-propagated to the Gaussian parameters with partial derivatives, as shown in Fig. 2(a). The partial derivatives related to c, α, o, Σ' are computed respectively with corresponding Gaussian parameters. However, in the computation graph of $\frac{\partial C}{\partial \alpha}$, the computation results of deeper Gaussians are also involved. More specifically, the accumulated transparency

T_n and reversely accumulated transparency S_n has data dependency upon T_{n+1} and S_{n+1} as Eq. 3.

$$T_n = \frac{T_{n+1}}{1-\alpha_n}, S_n = c_{n+1}\alpha_{n+1}T_{n+1} + S_{n+1}. \quad (3)$$

Unlike the volume rendering process, the partial derivative related to α requires computation in the descending order of Gaussian depth due to the occlusion effects. Hence the Gaussians obtained during the rendering process should be stored to carry out reverse rendering process in back propagation.

Density control is executed every 100 iterations in the GS process to better model the unobserved scene details in the initial SfM point cloud. When density control is triggered, 3D GS detects the Gaussian ellipsoids with a large positional gradient, which indicates under or over reconstruction of scene detail. In such cases, either cloning or segmenting the Gaussian is implemented to densify the point cloud. As for the irrelevant transparent Gaussians, 3D GS periodically trims Gaussians with low opacity (o less than 0.005) to reduce the overall model size and improves rendering performance.

B. Order Independent Gaussian Splatting

As mentioned above, 3D GS requires depth-ordered Gaussian inputs to assure correct rendering results. However, sorting numerous Gaussians is costly, and limits the potential of intra-tile parallel rendering due to transmittance and gradient dependency. Recent algorithm advances start to explore partial-ordered [11] or order-independent [12] Gaussian splatting. For instance, depth guided weighted blending in [12] is an order-independent GS scheme, which completely break data dependency in rendering by using splat depth to guide the blending. By this way, the α blending is simplified as:

$$C = \Sigma c_i \cdot \alpha_i \cdot (1 - k_1 \cdot d_i) \cdot k_2 \quad (4)$$

in which d_i is the Gaussian depth and k_1, k_2 are learnable parameters. By replacing T with a depth-related factor, order-independent Gaussian splatting isolates the contribution of each Gaussian to pixel color and enables Gaussian-level parallelism. Moreover, the time cost to sort Gaussians and memory cost to store the ordered Gaussians list can be eliminated.

Fig. 2(b) shows the back propagation flow for depth-guided blending in order-independent GS. We mainly introduce the computation of $\frac{\partial C}{\partial c}$ and $\frac{\partial C}{\partial \alpha}$ as other partial derivatives can be calculated under the chain rule. Derived with Eq. 4, these two derivatives can be expressed as Eq. 5.

$$\frac{\partial C}{\partial c_n} = \alpha_n \cdot (1 - k_1 \cdot d_n) \cdot k_2, \frac{\partial C}{\partial \alpha_n} = c_n \cdot (1 - k_1 \cdot d_n) \cdot k_2 \quad (5)$$

Similar to the volume rendering process, the partial derivatives of one certain Gaussian can be computed solely with its own parameters. Consequently, back propagation of multiple Gaussians can be simultaneously executed without order dependency when propagating the loss from a certain pixel. In conclusion, order-independent GS with depth guided weighted blending simplifies the computation and enables parallelism for both the volume rendering and back propagation.

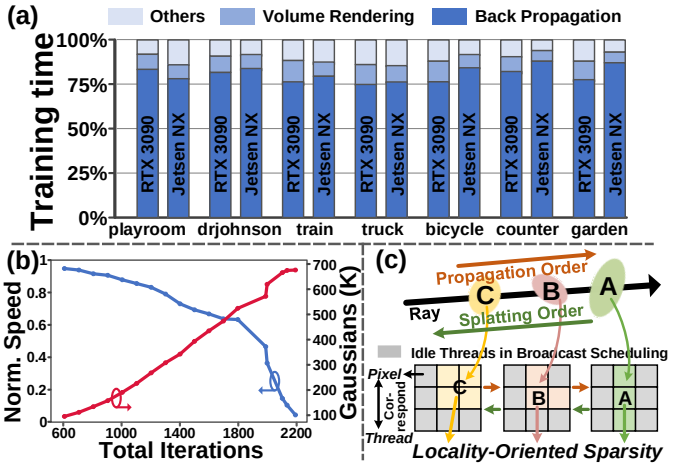


Fig. 3. (a) Runtime breakdown of RTX 3090(left) and Jetson NX(right). (b) throughput degradation caused by limited memory on Jetson NX. (c) thread idling and memory congestion in order-dependent Gaussian splatting.

III. Workload Profiling and Analysis

To better understand the workload characteristics, we ran the end-to-end process of 3D GS on complex real-world scenes from Tanks & Templates, Deep Blending [13] and Mip-NeRF 360 [14] datasets. We ran these test sets on two representative hardware platforms, i.e. an RTX 3090 GPU and edge Nvidia Jetson Xavier NX platform. A total of 30000 training iterations are sampled on RTX 3090, while 3000-7000 iterations are run on Jetson NX due to memory limitation.

As the runtime breakdown shown in Fig. 3(a), similar runtime distribution is observed on both platforms. We identify the back propagation process as the main bottleneck which contributes more than 70% of total runtime, while the volume rendering process consumes only around 10% latency. This is partially because of the numerous gradient update and propagation introduced in the backward propagation process. The remaining latency comes from other processes, including the preprocessing and density control process. When we analyze the training process on Jetson NX, we noticed a significant throughput degradation during training (Fig. 3(b)), since the growing Gaussian primitive numbers cause the memory out of space for the sorted Gaussian lists and rendering features.

Apart from memory limitation, we also observe another key reason for the lengthy back propagation, i.e. severe hardware under-utilization caused by locality-oriented Gaussian sparsity. As shown in Fig. 3(c), in conventional 3D GS flow, the target Gaussian splat is broadcast to the a thread block in GPU and each thread is assigned to render a certain pixel. However, in reality the Gaussian ellipsoids tend to be small to faithfully represent surface details, i.e. only a few pixels *local to* the splat center will be rendered. Such phenomenon leaves considerable pixel threads idle, which we refer as *Locality-Oriented Sparsity* or *Splat Locality*. In a typical thread block, splat locality will cause around 80-90% threads stay idle, being one major bottleneck in 3D GS.

To address the above limitations, we explore an efficient 3D GS training accelerator solution with the following strategies. First, to overcome memory limitations on mobile platforms,

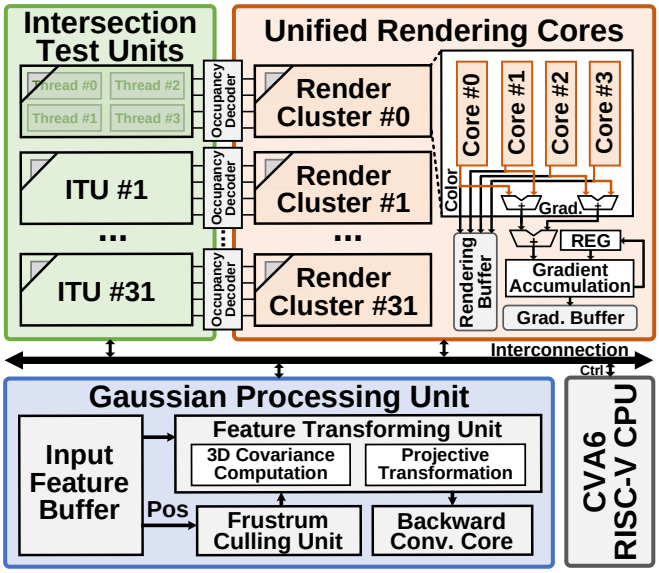


Fig. 4. Architecture overview of our Local-GS training accelerator.

we adopt an order-independent Gaussian rendering scheme to eliminate sorting latency and storage overhead. Second, to reduce the thread idling caused by splat locality, we employ pixel-wise intersection test and Gaussian-centered workload scheduling scheme. Hence the rendering threads only need to focus on pixels that intersected with the local sphere of GS.

IV. Training Accelerator for 3D GS: Local-GS

A. Architecture Overview

In this section, we introduce Local-GS, a unified training-rendering accelerator for order-independent 3D GS exploiting the splat locality feature. As shown in Fig. 4, Local-GS mainly consists of three components: a Gaussian processing unit that transforms 3D Gaussian features into 2D for splat rendering, 32 clusters of unified rendering cores for parallel volume rendering and back propagation and 32 intersection test units (ITUs) paired with rendering clusters through an occupancy decoder for parallel Gaussian intersection testing in the local regions. A host RISC-V CPU core is added for programming and general-purpose status control.

During 3D GS operations, the Gaussian processing unit first culls out redundant Gaussians for image rendering. Following that, the feature transforming unit converts the 3D Gaussian features into 2D features, as well as transforming the coordinate base vectors to assist pixel-wise intersection test. The Gaussian processing unit also holds a tiny convolution core to calculate simplified SSIM loss with reduced kernel size for backward propagation. With the help of base vectors, the ITUs perform incremental pixel-level intersection test local to the gaussian center to generate sparse occupancy bitmaps. The compressed bitmaps are then decoded into continuous pixel flow for rendering core workload assignment. Finally, the Gaussian rendering cores execute the forward volume rendering, or backward loss propagation, or Gaussian gradients.

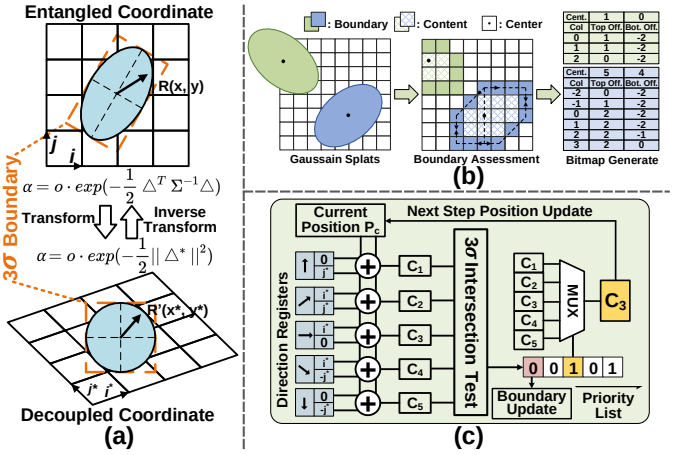


Fig. 5. (a) Our proposed intersection test method, (b) the workload scheduling of an ITU, and (c) detailed architecture of an ITU thread.

The Gaussian processing unit consists of a frustum culling unit (FCU) to cull out the out-of-view Gaussians, a feature transforming unit (FTU) to process Gaussian features and coordinate base vectors for intersection testing and a small convolution core for loss calculation. During the preprocessing stage, the FCU first identifies the viewing frustum boundaries as unit length, then normalize the input Gaussian coordinates accordingly. The FTU converts the 3D features of Gaussian ellipsoids into 2D features on the projection plane through estimated affine transformation with local Jacobian matrix to calculate Gaussian center position(μ), opacity(o), transformation matrix(T), and obtain an RGB color(c).

B. Parallel Intersection Test Unit

In this work, we adopted a local Gaussian-centered pixel-wise parallel intersection test to merge bounding-box detection and α computation at pixel-level, leveraging the splat locality feature and enhance the hardware utilization. In the original Gaussian splatting, intersection test creates a rough boundary for each GSussian splat. The contribution of pixels inside the boundary is measured by α, which is calculated pixel wise with precise positional information.

Fig. 5(a) exhibits our intersection test, in which an incremental α value computation of candidate pixels is performed once pixel in the 3σ local Gaussian boundary. By reorganizing covariance calculation in Eq. 1, the non-diagonal inverse covariance matrix Σ^{-1} can be represented with the 2D transformation $\Sigma^{-1} = (TT^T)^{-1}$ and the α entangled with X, Y can be decoupled as a 2D Gaussian kernel,

$$\alpha = o \cdot \exp\left(-\frac{1}{2}(T^{-1}\Delta)^T(T^{-1}\Delta)\right) = o \cdot G(\Delta^*) \quad (6)$$

where $\Delta^* = T^{-1}\Delta$, or two component 1D Gaussian kernels along major and minor axis of Gaussian ellipsoid:

$$\alpha_i = o_i \cdot G\left(\Delta_x^*\right) \cdot G\left(\Delta_y^*\right) \quad (7)$$

With this transform, we can incrementally iterate through pixels and identify their α values through Δ^*

$$\Delta^*(m,n) = \Delta^* + m \cdot i^* + n \cdot j^* \quad (8)$$

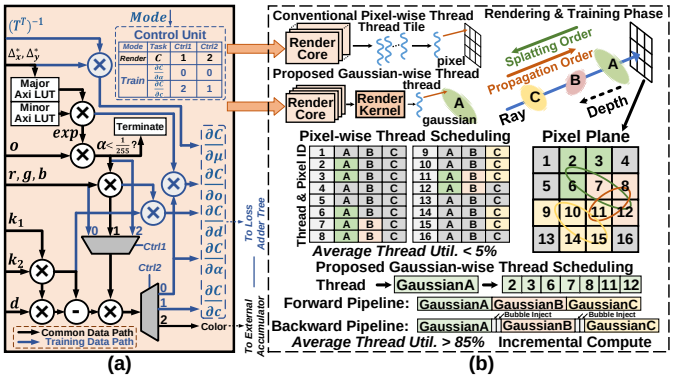


Fig. 6. (a) The training/rendering datapath of our Unified Rendering Core, (b) comparison between our Gaussian-centered workload scheduling and conventional scheduling in the original 3D GS.

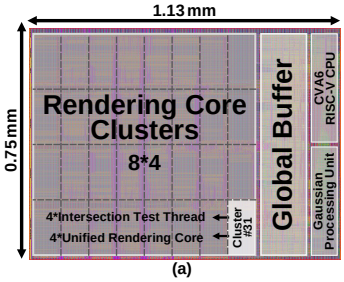
using relative pixel position m, n and transformed base vectors $i^* = T^{-1}i$ and $j^* = T^{-1}j$.

Fig. 5(b) illustrates the workflow of an ITU, where four working threads collaboratively search for intersected pixels within the local Gaussian splat. The process begins at the Gaussian center, and threads search along the Y axis until reaching the splat boundary, identifying valid top and bottom boundary pixels. The threads then split into two groups to explore adjacent columns along the X axis. The search terminates when the top and bottom pixels of the same column overlap, with no more adjacent intersected tiles. During the searching process, the ITU registers the offset of top and bottom from Gaussian center pixels of each column, i.e. 1 and -2 for column -1 of the blue splat, as well as the original pixel ID of the Gaussian center, i.e. (5,4), into a compact occupancy table to efficiently store the locality-based sparse intersection bitmap and schedule rendering core workload accordingly.

The detailed thread architecture of the ITU is shown in Fig. 5(c). When a thread reaches a new pixel, the working thread examines the valid searching directions and combines the corresponding transformed base vectors i^*, j^* stored in direction registers with current pixel position to generate next step position candidates. The candidate positions are then processed through the 3σ intersection test to check whether they are inside the Gaussian's local boundary. A candidate will be considered active only when both component vectors are located inside the boundary. The next step position is then updated according to the active candidate with highest priority, and update the ITU boundary register if the top-priority candidate C_1 is inactive.

C. Local-GS Rendering Core

Fig. 6(a) shows the datapath design of our unified rendering core. The core can render Gaussians or propagate partial gradients according to the assigned mode signals. In volume rendering mode, the training data path stays inactive, and the input Gaussian parameters flow though the common data path to generate rendered pixel color. The rendering core calculates the pixel α value as Eq. 7. Following that, α is weighted by Gaussian depth to formulate a blending component as Eq. 4.



Component	Configuration	Area[mm^2]	Power[mW]
Rendering Core Clusters	32×(4 Cores)	0.61	1282.4
Gaussian Processing Unit	4 Units	0.03	107.8
Global Buffer	256KB	0.16	148.2
RISC-V CPU + Others	CVA6	0.04	107.0
Total		**0.84**	**1685.0**

(b)

Fig. 7. (a) Layout view of Local-GS accelerator in 7nm technology, (b) design specification of Local-GS.

In back propagation mode, the training data path will be activated together with the common data path to calculate partial gradients in Fig. 2(b). The output gradients will be directed to a local adder tree in the kernel to accumulate with gradients obtained in other threads to form the accumulated splat-level gradient for parameter optimization. With local gradient reduction, pipeline congestion caused by simultaneous gradient updates can be eliminated.

Fig. 6(b) illustrates the workload scheduling strategy for rendering cores. In conventional pixel-wise thread scheduling in GPU and accelerator [8], the hardware utilization is low due to the splat locality feature, e.g. Gaussian A, B, and C only affect 7, 4, and 5 pixels in the pixel plane. Hence we adopted a Gaussian-centered scheduling scheme to improve hardware utilization. We aggregate four rendering cores to form a rendering kernel, which corresponds to one thread and processes a Gaussian. The thread calculates four pixels affected by the current Gaussian every cycle. When the thread completes the forward rendering for current Gaussian, changing assigned Gaussians will not influence the pipeline, as the inputs are pixel-aligned and will be accumulated upon storage. While in the backward running, we inject two pipeline bubbles between Gaussians to ensure the local gradient adder tree has finished the accumulation and generated valid output results. With all factors considered, our proposed workload scheduling scheme achieved more than 85% hardware utilization on average, which greatly outperformed the baseline method.

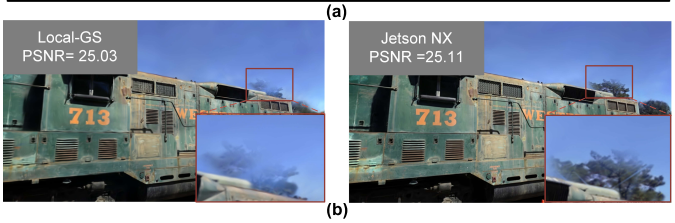
V. Evaluations

A. Implementation

We implemented our Local-GS accelerator using a 7nm technology [15], as the layout view shown in Fig. 7. The accelerator is fully developed in RTL, with the I/O buffers and CPU main memory generated by a commercial SRAM compiler. The design is synthesized by Cadence Genus and physically implemented by Cadence Innovus at 1GHz. The total active area of Local-GS is 1.13×0.75 mm for 32 rendering core clusters and host CPU, for which we adopt an open-source RISC-V CVA6 CPU [16]. The accelerator area is dominated by the 32 rendering core clusters, i.e. 72.7%, and corresponding buffers, i.e. 18.9%. The detailed design breakdown of Local-GS is listed in Fig. 7(b). The following energy, performance, and area results are reported based on post-P&R netlist at the typical corner, i.e. 0.7V at 25°C.

Scene	Playroom	Drjohnson	Train	Truck	Bicycle	Counter	Garden
3DGS	37.11	34.57	25.11	29.01	25.12	29.19	27.32
Sort-free	37.19	35.26	25.04	29.44	24.53	29.57	27.56
Local-GS	37.03	35.11	24.83	28.87	24.40	29.21	27.17

(a)



(b)

Fig. 8. (a) PSNR evaluation results, (b) example of one rendered image.

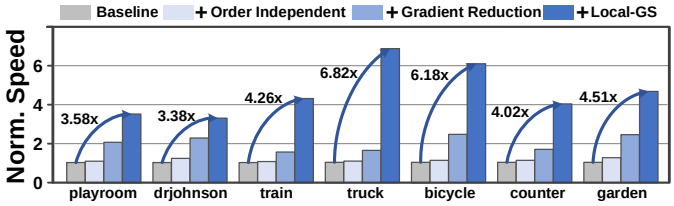


Fig. 9. Normalized speedup for 3D GS with our Local-GS accelerator.

We evaluate our order-independent architecture by comparing with the original 3D GS [7] and Sort-Free GS [12], which adopts the depth-guided blending method. Our Local-GS is compared with existing hardware of Jetson NX and RTX 3090 GPU using real-world workloads from multiple datasets with both indoor and outdoor scenes. The datasets include Tank and Templates, Deep Blending, and Mip-NeRF 360 dataset.

B. Benefits Evaluations

In this section, we evaluate the performance and efficiency benefits of our Local-GS accelerator. First, we compare our rendered image quality with the original 3D GS and Sort-Free GS. Fig. 8 presents the PSNR score across multiple rendering categories and an example rendering result from the train dataset. Local-GS achieves almost the same rendering quality compared with 3D GS and Sort-Free GS on the train dataset, i.e. 24.83 compared with 25.11 and 25.04. The degradation in mainly invoked by the order independent rendering process, which results in the semi-transparent object in Fig. 8(b).

To better evaluate the efficiency of our architecture, we compare the normalized speedup with state-of-the-art Gaussian rendering accelerator, GSCore [8]. Since GSCore only supports forward rendering, we provided it with a training hardware plugin of the same architecture and specifications as the baseline to ensure a fair comparison. As shown in Fig. 9, having optimizations of order-independent rendering and gradient loss reduction on GSCore improve the performance by an average of 1.1× and 1.8×, respectively. With our Local-GS architecture, an average 4.7× speedup is obtained across different scenarios compared with GSCore, which is mainly contributed by leveraging the splat locality via the pixel-wise intersection test and Gaussian-centered workload scheduling. For particular 3D scene category, Local-GS performs best on Truck and Bicycle scenes, yielding 6.82× and 6.18× speed-up compared to GSCore. This is mainly because of the frequent sudden depth changes that disabled the efficient early termination scheme in GSCore. In contrast, indoor scenes are relatively challenging, where objects may be completely occluded from sight, e.g., 3.58× and 3.38× speed-up for Playroom and Drjohnson.

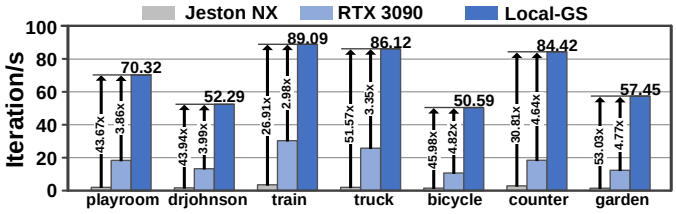


Fig. 10. 3D GS training throughput of baseline hardware and our design.

TABLE I
COMPARISON BETWEEN LOCAL-GS, GSCORE AND JETSON NX

	Ours	GSCore	Xavier NX
Frequency	1GHz	1GHz	1.1GHz
core Num.	32*4	4*16	384 CUDA Cores
Technology	7nm	28nm	12nm
Precision	FP16	FP16	FP32
Power(mW)	1685	870	2×10^4
Area(mm^2)	**0.848(Post P&R)**	**3.95(Synthesize)**	350
Energy Efficiency(mJ/Iter)	**19.6**	**9.88(Rendering Only)**	6.06×10^3
Support Training	✔	✘	✔
Training Time	5-10min	✘	4-6hour
Rendering Speed (FPS)	350.6	79.5	10.6

C. Comparison and Discussion

Fig. 10 shows our comparison with existing GPU hardware. Thanks to the benefits of software-hardware co-optimizations, Local-GS achieved a training throughput improvement by 26.9-53× across different scenarios compared to edge Jetson NX. Compared to the high-performance RTX 3090 GPU, Local-GS also receives an improvement of 3.0-4.8×.

Table I lists the detailed comparison between our Local-GS accelerator and prior accelerator and edge GPU. Compared with available training-capable edge hardware, e.g. Jetson NX, Local-GS achieves about 50× throughput gain and a 19.6 mJ/iteration efficiency, which is a nearly three orders of magnitude efficiency boost. For rendering, Local-GS achieves 4.38× better throughput than dedicated rendering GSCore since we exploit splat locality and obtain an average 85% hardware utilization. Overall, our Local-GS provides a promising 3D training/rendering hardware solution, which can be applied for diverse on-demand real-time NVS applications.

VI. CONCLUSION

In this paper, we presented a unified training-rendering accelerator, Local-GS, to facilitate on demand scene training and real-time rendering on edge devices. Compared to other accelerators, Local-GS achieves 3.5× hardware utilization increase by exploiting localization-based Gaussian sparsity and leveraging order-independent parallel training-rendering scheme. Our Local-GS accelerator is implemented and evaluated in 7nm technology, achieving 26.9-53× faster training speed and three orders of magnitude energy efficiency boost compared with Jetson Xavier NX mobile GPU.

REFERENCES

[1] J. Dai, Z. Zhang, S. Mao, and D. Liu, "A view synthesis-based 360° vr caching system over mec-enabled c-ran," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 10, pp. 3843–3855, 2020.

[2] S. Lewin, M. Vandegar, T. Hoyoux, O. Barnich, and G. Louppe, "Dynamic nerfs for soccer scenes," in *6th International Workshop on Multimedia Content Analysis in Sports*, p. 113–121, 2023.

[3] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M.-H. Yang, "Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21634–21643, June 2024.

[4] A. Kar, C. Häne, and J. Malik, "Learning a multi-view stereo machine," in *31st International Conference on Neural Information Processing Systems (NIPS)*, 2017.

[5] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 165–174, 2019.

[6] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: representing scenes as neural radiance fields for view synthesis," *Commun. ACM*, vol. 65, p. 99–106, dec 2021.

[7] B. Kerbl, G. Kopanas, T. Leimkuehler, and G. Drettakis, "3d gaussian splatting for real-time radiance field rendering," *ACM Trans. Graph.*, vol. 42, jul 2023.

[8] J. Lee, S. Lee, J. Lee, J. Park, and J. Sim, "Gscore: Efficient radiance field rendering via architectural support for 3d gaussian splatting," in *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 497–511, 2024.

[9] L. Wu, H. Zhu, S. He, J. Zheng, C. Chen, and X. Zeng, "Gauspu: 3d gaussian splatting processor for real-time slam systems," in *57th IEEE/ACM International Symposium on Microarchitecture (MICR)*, 2024.

[10] J. L. Schönberger and J.-M. Frahm, "Structure-from-motion revisited," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4104–4113, 2016.

[11] F. Hahlbohm, F. Friederichs, T. Weyrich, L. Franke, M. Kappel, S. Castillo, M. Stamminger, M. Eisemann, and M. Magnor, "Efficient perspective-correct 3d gaussian splatting using hybrid transparency," 2024.

[12] Q. Hou, R. Rauwendaal, Z. Li, H. Le, F. Farhadzadeh, F. Porikli, A. Bourd, and A. Said, "Sort-free gaussian splatting via weighted sum rendering," 2024.

[13] P. Hedman, J. Philip, T. Price, J.-M. Frahm, G. Drettakis, and G. Brostow, "Deep blending for free-viewpoint image-based rendering," *ACM Trans. Graph.*, vol. 37, dec 2018.

[14] J. T. Barron, B. Mildenhall, D. Verbin, P. P. Srinivasan, and P. Hedman, "Mip-nerf 360: Unbounded anti-aliased neural radiance fields," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5460–5469, 2022.

[15] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016.

[16] B. Sá, L. Valente, J. Martins, D. Rossi, L. Benini, and S. Pinto, "Cva6 risc-v virtualization: Architecture, microarchitecture, and design space exploration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

Clearance-Constrained PCB Global Placement with Heterogeneous Components

Yan-Jen Chen[1], Wei-Kai Huang[1], Chung-Ting Tsai[2], Chiao-Yu Ou[1], and Yao-Wen Chang[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106319, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106319, Taiwan
yjchen@eda.ee.ntu.edu.tw; wkhuang@eda.ee.ntu.edu.tw; b09901020@ntu.edu.tw; cyou@eda.ee.ntu.edu.tw; ywchang@ntu.edu.tw

***Abstract*—The complexity of design rules and intense time-to-market demands have made auto-placement tools essential for advanced printed circuit board (PCB) designs. This paper presents a novel PCB placement framework to handle pad-to-pad clearance constraints and heterogeneous components to address these challenges. Unlike existing academic placers, our framework focuses on the following key features: a wire-area model to account for various routing resource needs between power and signal nets, a pad-to-pad clearance model to minimize spacing violations, and a two-sided, pad-type-aware density model to reduce component and pad overlap. We further develop a quadratic programming-based legalizer to resolve constraint violations among components of varying shapes. Experimental results show the effectiveness and efficiency of our framework, surpassing two state-of-the-art academic placers in post-routing quality on both academic and industrial benchmarks.**

I. Introduction

The rapidly evolving demands in semiconductor devices have driven leading electronics companies to seek increasingly shorter printed circuit board (PCB) design cycles to maintain competitive time-to-market requirements. Furthermore, the rise of high-performance computing servers, artificial intelligence (AI) accelerators, and the Internet of Things (IoT) has made these design cycles more challenging. As a result, modern PCB designs incorporate more components and requires more complex design rules, inevitably lengthening the design cycles.

For the PCB design flow, physical design is among the most time-intensive stages. Experienced engineers need to find a desired layout by iterating placement and routing processes, requiring extensive expertise and even time-consuming human efforts. Existing commercial and open-source electronic design automation (EDA) tools, including Allegro [1], Altium [2], and KiCad [3], primarily focus on supporting manual placement, auto-routing, and design-rule checking. There is an industry-wide push to develop advanced PCB placement algorithms that could replace manual placement and expedite this critical design stage to shorten the design cycle [4], [5]. For example, Renesas' [6] recent $5.9 billion acquisition of Altium in February 2024 highlights the growing importance of PCB design automation in accelerating innovation.

The heterogeneous nature of PCB components and the complexity of associated design rules pose substantial challenges to placement algorithms. Figure 1 illustrates an example of heterogeneous components, highlighting unique features: components may have arbitrary shapes, be mounted on either side of the board, and incorporate both surface-mount-device (SMD) pads and through-hole (TH) pads, where only the latter penetrate through the board. Figure 2 shows the complexity of design rules derived from electrical requirements, such as minimizing crosstalk and ensuring insulation distances, which translate to clearance constraints (i.e., minimum spacing requirements between objects). Besides these features, PCBs are routed in a 135-degree style. In contrast, traditional placement algorithms often assume all components are rectangular, requiring only basic non-overlap constraints incompatible with modern PCB designs, and are routed in a 90-degree style. Given these unique challenges, PCB placement is widely regarded as a complex problem that necessitates advanced algorithms to achieve effective and efficient layout solutions.

A. Previous Work

Existing academic approaches to PCB placement can be broadly categorized into three methods: genetic algorithms, analytical placement, and reference placement. Genetic algorithm methods [7], [8] model component coordinates and orientations as chromosomes, optimizing a weighted objective function composed of heuristic metrics, such as thermal performance, timing, and layout tidiness. These algorithms adjust component locations and orientations through swapping or shifts. However, their reliance on random optimization techniques limits scalability, making them less suitable for large, complex designs.

This work was partially supported by AnaGlobe, ASUS, Delta Electronics, Google, Maxeda Technology, TSMC, NSTC of Taiwan under Grant NSTC 110-2221-E-002-177-MY3, NSTC 113-2218-E-002-041,NSTC 113-2223-E-002-007, and NSTC 113-2640-E-002-001.

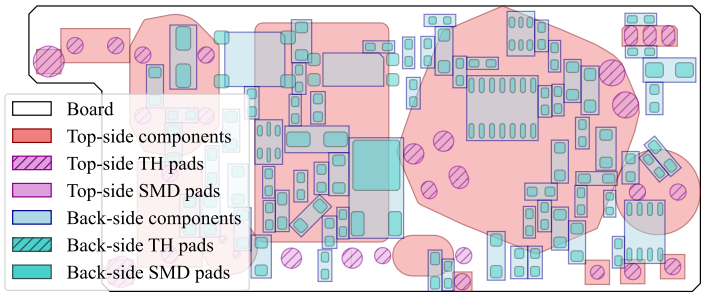
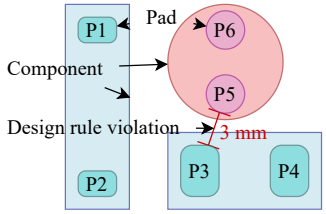


Fig. 1. An example of a modern PCB design with heterogeneous components. The board, components, and pads can have arbitrary shapes, with components rotatable at any angle and placeable on either the top or back side of the board. Through-hole (TH) pads penetrate the board, occupying placement areas on both sides, while surface-mount device (SMD) pads occupy placement areas on a single layer. This dense and complex architecture presents significant challenges for PCB placement automation.

	P1	P2	P3	P4	P5	P6
P1						
P2	0					
P3	1	1				
P4	1	1	0			
P5	5	5	5	5		
P6	5	5	5	5	0	

(a) Pad-to-pad clearance (unit: mm)



(b) Placement with one design rule violation

Fig. 2. An illustration of pad-to-pad clearance constraints in PCB design. (a) The pad-to-pad clearance represents the minimum spacing required between two pads. (b) Design rule violations can arise between arbitrary shapes, making them challenging to detect and resolve.

Analytical placement methods adopt strategies from integrated circuit (IC) placement, applying gradient descent to spread components and minimize objectives like total wirelength, component density penalties, and other additional costs. Cheng *et al.* [9] introduced a net separation regularizer to enhance net separation and improve routability for PCB designs. Huang *et al.* [10] developed a density calculation algorithm using line-drawing and polygon-clipping techniques to estimate density contributions from rotated rectangular components. Tsou *et al.* [11] proposed a force-directed PCB placement framework that optimizes different design intents (e.g., routability, power flow, and clearance between pads) in different optimization stages. Additionally, Chen *et al.* [12] proposed a 2.5D density model to simultaneously place different-layer components. In particular, Tsou *et al.*'s and Chen *et al.*'s approaches accommodate two-sided component placement, whereas most other methods are limited to one-sided placement. Despite these advancements, analytical approaches often model components with simplified rectangular bounding boxes, which can lead to wasted space and impaired spatial efficiency in heterogeneous component layouts. Moreover, while Tsou *et al.*'s method effectively reduces clearance constraint violations, it may introduce large displacements due to the use of a greedy search for legalizing placements.

The reference placement approach, by contrast, attempts to place components in new designs based on established patterns stored in a database. For instance, Su *et al.* [13] developed a subgraph-matching algorithm to enhance the efficiency and quality of pattern matching in PCB layouts. However, this approach requires an extensive pattern database, which is beyond the scope of this paper.

In summary, while these academic placement approaches offer valuable insights, they lack full compatibility with the demands of modern PCB de-

signs. Genetic algorithms struggle with scalability; analytical placements often simplify component shapes in ways that reduce spatial efficiency, and reference placement methods require a pre-existing pattern database, limiting their direct applicability to new designs without extensive data preparation.

B. Our Contributions

This paper adopts an analytical placement approach and presents a new PCB placement algorithm to address the limitations of existing academic placers, particularly with regard to pad-to-pad clearance constraints and heterogeneous components. The main contributions of our algorithm are as follows:

- To the best of our knowledge, this paper is the first PCB global placement work considering pad-to-pad clearance constraints and heterogeneous components.
- A wire-area model is introduced to calculate the total routing resource consumption for power and signal nets, establishing better placement optimization.
- A pad-to-pad clearance model is proposed to minimize spacing violations between pads, enhancing placement quality and routability.
- A two-sided, pad-type-aware density model is developed to reduce overlaps between heterogeneous components and pads. This approach supports co-optimization across both PCB sides and is particularly suited for dense PCB designs.
- A quadratic programming legalizer is proposed to resolve remaining overlaps between heterogeneous components and pads. This approach addresses runtime bottlenecks, significantly enhancing the scalability of the PCB placement algorithm.
- Experimental results on both academic and industrial PCB designs demonstrate the effectiveness and efficiency of our algorithm. The proposed placement algorithm achieves the best post-routing result compared to two state-of-the-art academic PCB placers. Specifically, there is an average 30% post-routing wirelength reduction in the academic designs, an average 74% clearance violations reduction, and an average 32% routability improvement in the industrial designs.

The remainder of this paper is organized as follows. Section II states our problem formulation and introduces our analytical placement framework. Section III gives our overall design flow. Sections IV and V detail our algorithms. Section VI reports the experimental results. Section VII gives the conclusion.

II. Preliminaries

This section introduces the PCB placement problem and our analytical placement framework. Table I lists the notations used in this paper.

A. Problem Formulation

The PCB placement problem can be formulated as a hypergraph $G = (V, E)$ placement problem, where the set of vertices V represents components and the set of hyperedges E represents nets. Let x_i and y_i denote the x and y coordinates of component v_i, respectively. Components are pre-assigned to either the top side or back side of the board, may have arbitrary shapes, and may be pre-placed at a fixed location. Each component can include up to two types of pads: surface-mount device (SMD) pads and through-hole (TH) pads. All pads are assumed to have convex shapes. In particular, TH pads penetrate the board, occupying space on both sides. Let w_e denote the wire-width of a net e. We intend to determine the desired position of each component such that the target cost (i.e., total wire area) is minimized and the following constraints are satisfied:

- **Nonoverlapping constraints.** Components on the same layer must not overlap. Additionally, components must not overlap with TH pads located on the opposite side of the board.
- **Pad-to-pad clearance constraints.** The minimum distance between any pair of two pads, p_m and p_n, must be at least $c_{m,n}$ units, where $c_{m,n}$ is a user-defined value.

B. Our Analytical Placement Framework

The PCB placement problem is solved in two main stages: (1) global placement, which aims to efficiently find a solution with minimal constraint violations, and (2) legalization, which resolves any remaining violations. Global placement is crucial, as it largely determines the quality of the final solution. In our approach, the global placement problem is formulated as a constrained optimization problem defined as follows:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \quad & W(\mathbf{x},\mathbf{y}), \\ \text{s.t.} \quad & C_{p2p}(\mathbf{x},\mathbf{y}) = 0, \\ & D(\mathbf{x},\mathbf{y}) = 0, \end{aligned} \tag{1}$$

TABLE I
Notations used in this paper.

V	Set of components.
$V_t(V_b)$	Set of top-side (back-side) components.
E	Set of nets.
$P_t^S(P_b^S)$	Set of top-side (back-side) SMD pads.
P^T	Set of TH pads.
O	Set of obstacles.
$(\mathbf{x},\mathbf{y})$	x and y coordinates of all components.
(x_i, y_i)	x and y coordinates of component v_i.
$W(\mathbf{x},\mathbf{y})$	Total wirelength.
$C_{p2p}(\mathbf{x},\mathbf{y})$	Total pad-to-pad clearance violations.
$D(\mathbf{x},\mathbf{y})$	Density penalty.
$p(p_m, p_n)$	Pad-to-pad clearance penalty between p_m and p_n.
$d(p_m, p_n)$	Minimum distance between pads p_m and p_n.
$c_{m,n}$	Clearance between pads p_m and p_n.
w_e	Wire width of net e.
M_s	Vertex matrix of a shape s.

where $W(\mathbf{x},\mathbf{y})$, $C_{p2p}(\mathbf{x},\mathbf{y})$, and $D(\mathbf{x},\mathbf{y})$ represent the total wirelength, total pad-to-pad clearance violations, and the density penalty (indicating the degree of component overlaps), respectively. To convert Equation 1 into an unconstrained optimization problem, we apply Lagrange relaxation as follows:

$$\min_{\mathbf{x},\mathbf{y}} \quad W(\mathbf{x},\mathbf{y}) + \lambda_1 C_{p2p}(\mathbf{x},\mathbf{y}) + \lambda_2 D(\mathbf{x},\mathbf{y}), \tag{2}$$

where λ_1 and λ_2 are the Lagrange multipliers. Equation 2 is then solved using gradient descent, with iterative increases in λ_1 and λ_2.

To better capture the 135-degree routing style typically seen in PCB design, the total wirelength $W(\mathbf{x},\mathbf{y})$ is modeled using the X half-perimeter wirelength (XHPWL) [14] instead of the traditional half-perimeter wirelength (HPWL) used in IC placement:

$$\begin{aligned} W(\mathbf{x},\mathbf{y}) &= \sum_{e\in E} l_e(\mathbf{x},\mathbf{y}), \\ l_e(\mathbf{x},\mathbf{y}) &= \left(\sqrt{2}-1\right)\left(\max_{v_i,v_j\in e}|x_i - x_j| + \max_{v_i,v_j\in e}|y_i - y_j|\right) - \\ &\quad \left(\frac{\sqrt{2}}{2}-1\right)\left(\max_{v_i,v_j\in e}|(x_i+y_i) - (x_j+y_j)| + \right. \\ &\quad \left. \max_{v_i,v_j\in e}|(x_i-y_i) - (x_j-y_j)|\right), \end{aligned} \tag{3}$$

where $e \in E$ represents a net and $l_e(\mathbf{x},\mathbf{y})$ denotes the XHPWL of net e. Since XHPWL is non-differentiable, we use a weighted-average model [15] for a smooth approximation.

The total pad-to-pad clearance violation $C_{p2p}(\mathbf{x},\mathbf{y})$ is defined as the sum of insufficient distances between all pairs of pads on each side of the board:

$$\begin{aligned} C_{p2p}(\mathbf{x},\mathbf{y}) = &\sum_{p_m,p_n\in P_t^S\cup P^T} p(p_m,p_n) + \\ &\sum_{p_m,p_n\in P_b^S\cup P^T} p(p_m,p_n), \\ p(p_m,p_n) = &\max\left(c_{m,n} - d(p_m,p_n), 0\right), \end{aligned} \tag{4}$$

where P_t^S (P_b^S), P^T, and $p(\cdot)$ are the sets of top-side (back-side) SMD pads, TH pads, and pad-to-pad clearance penalty between two pads, respectively, and $d(\cdot)$ represents the minimum distance between pads. To calculate $d(\cdot)$ for pads with arbitrary convex shapes, we employ the Gilbert-Johnson-Keerthi (GJK) distance algorithm [16] and the expanding polytope algorithm (EPA) [17].

The density penalty $D(\mathbf{x},\mathbf{y}) = D_t(\mathbf{x},\mathbf{y}) + D_b(\mathbf{x},\mathbf{y})$ comprises the top-side density penalty $D_t(\mathbf{x},\mathbf{y})$ and back-side density penalty $D_b(\mathbf{x},\mathbf{y})$, similar to the approach used by Chen *et al.* [12]. $D_t(\mathbf{x},\mathbf{y})$ is defined as the eDensity model [18] to minimize component overlaps:

$$D_t(\mathbf{x},\mathbf{y}) = \sum_{v_i\in V_t} q_i \phi(x_i, y_i), \tag{5}$$

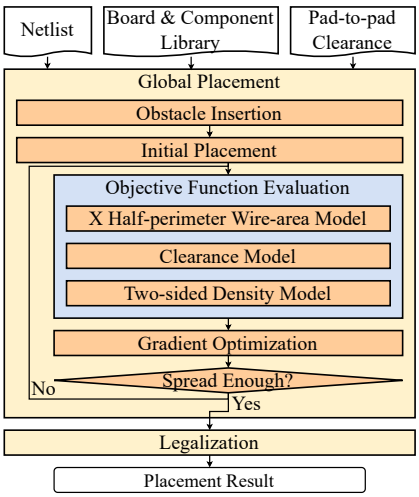


Fig. 3. The overall flow of our algorithm.

where V_t is the set of top-side components, q_i is the area of component v_i, and $\phi(x_i, y_i)$ is its electric potential, computed using spectral methods and the fast Fourier transform. The placement region is divided into $m_x \times m_y$ bins, each associated with a frequency index $(\omega_i, \omega_j) = (\frac{\pi i}{m_x}, \frac{\pi j}{m_y})$. The density map $\rho(x, y)$ can be calculated using these bins, representing the total area of components within each bin divided by the bin area. The electric potential $\phi(x, y)$ is then calculated as follows:

$$\begin{aligned} a_{i,j} &= \frac{1}{m_x m_y} \sum_{x,y} \rho(x, y) \cos(\omega_i x) \cos(\omega_i y), \\ \phi(x, y) &= \sum_{i,j} \frac{a_{i,j}}{\omega_i^2 + \omega_j^2} \cos(\omega_i x) \cos(\omega_i y), \end{aligned} \tag{6}$$

where $a_{i,j}$ is the density coefficient. $D_b(\mathbf{x}, \mathbf{y})$ is defined similarly. Since components have arbitrary shapes, a module drawing algorithm based on Bersenham's line drawing algorithm is used in the calculation [10]. However, the previous module drawing algorithm is not accurate. Further, TH pads must be considered on both sides due to their penetration. Therefore, we introduce our two-sided density model in Section IV.

III. Overall Flow

This paper presents a novel placement algorithm for PCB designs that is driven by clearance constraints and aware of heterogeneous components. As illustrated in Figure 3, our placement framework begins with input data comprising a netlist (i.e., schematic), a board library, a component library, and a pad-to-pad clearance table. The global placement algorithm spreads the components across the board to find a desired layout with slight overlaps and clearance violations. There are four stages in the global placement algorithm: obstacle insertion, initial placement, objective function evaluation, and gradient optimization. Finally, a legalization algorithm is employed to resolve any remaining constraint violations, ensuring a valid placement result. Detailed descriptions of these algorithms are provided in the following sections.

IV. Global Placement

In global placement, the process begins with obstacle insertion to convert a non-rectangular board into a rectangular placement area. Next, an initial placement is performed to find a configuration that minimizes the total wirelength. Following this stage, the objective function is iteratively evaluated, and gradient descent is applied to distribute the components. The iterative process continues until the components are sufficiently spread across the placement area. Each stage in our global placement algorithm is detailed in the following parts.

A. Obstacle Insertion

Since PCBs often have irregular shapes, converting a board into a standardized rectangular placement area is essential for efficient calculation. For a non-rectangular board B, represented by a set of counter-clockwise vertices, the rectangular placement area is determined as follows: (1) calculate the axis-aligned bounding box R of board B, (2) perform a geometry XOR operation between the polygons R and B to define the obstacles O, and (3) decompose O into a set of convex obstacles O'

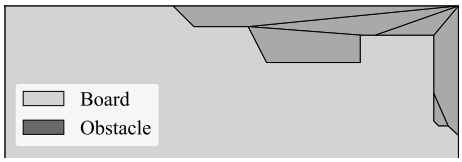


Fig. 4. An example of obstacle insertion for non-rectangular boards. Convex obstacles are added to create a rectangular placement area, preventing blocks from being placed outside the board.

using a convex decomposition algorithm. These obstacles contribute their areas to the density map $\rho(x, y)$ to prevent components from being placed in these restricted zones. Figure 4 illustrates an example of a board, along with the resulting placement area and obstacles generated by our algorithm.

B. Initial Placement

Following obstacle insertion, we seek an initial solution that minimizes the total XHPWL to establish a starting point for the main optimization process. All movable components are initially placed randomly near the center of the board. Gradient descent is then applied to iteratively minimize Equation 3, adjusting component positions to reduce the total wirelength.

C. Objective Function Evaluation

After initial placement, we aim to solve Equation 2 accurately with gradient descent algorithms. Therefore, a robust and accurate model for routing resources and density map is necessary. Moreover, a smooth and differentiable approximation model of the pad-to-pad clearance violation defined in Equation 4 is needed because the minimum distance calculation using GJK-EPA is not differentiable. These models are introduced in the following sections.

1) X Half-perimeter Wire-area Model: The XHPWL model does not accurately represent a net's consumed routing resource on a PCB, as the preferred routing wire width varies with nets. For instance, a power/ground net requiring a wider wire width consumes more routing resources than a signal net with the same wirelength. Consequently, components connected to nets with wider wire widths should ideally be positioned closer together. To address this problem, we propose an X half-perimeter wire-area (XHPWA) model $\tilde{W}(\mathbf{x}, \mathbf{y})$ in place of the XHPWL $W(\mathbf{x}, \mathbf{y})$, defined as follows:

$$\tilde{W}(\mathbf{x}, \mathbf{y}) = \sum_{e \in E} w_e l_e(\mathbf{x}, \mathbf{y}), \tag{7}$$

where w_e denotes the wire width of net e. This XHPWA model offers a more precise representation of routing resource consumption by considering varying wire width requirements across different nets, enabling better placement optimization.

2) Clearance Model: To achieve a smooth approximation for pad-to-pad clearance penalty $p(p_m, p_n)$, we use a bounding circle model to formulate a proximate penalty function $\tilde{p}(p_m, p_n)$. For each pad p_m, a bounding circle with radius r_m is pre-calculated. The proximate pad-to-pad clearance penalty function $\tilde{p}(p_m, p_n)$ can then be expressed as follows:

$$\begin{aligned} &\tilde{p}(p_m, p_n) = \\ &\quad \max\left(r_m + r_n + c_{m,n} - \sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}, 0\right). \end{aligned} \tag{8}$$

The gradient of $\tilde{p}(p_m, p_n)$ with respect to x_m is given by:

$$\frac{\partial \tilde{p}(p_m, p_n)}{\partial x_m} = \begin{cases} -\frac{x_m - x_n}{\sqrt{(x_m - x_n)^2 + (y_m - y_n)^2}}, & \tilde{p}(p_m, p_n) > 0 \\ 0, & \tilde{p}(p_m, p_n) \leq 0 \end{cases}. \tag{9}$$

The gradients with respect to x_n, y_m, and y_n are derived similarly. Figure 5 illustrates our pad-to-pad clearance model applied in global placement. Importantly, any overhead introduced by the bounding circle model is mitigated in the legalization algorithm discussed in Section V. Additionally, to avoid undefined behavior, pad-to-pad clearance violations are excluded in two special cases: (1) when two pads belong to the same component, where there is no room for optimization, and (2) when two pads belong to the same net, as they will ultimately be connected.

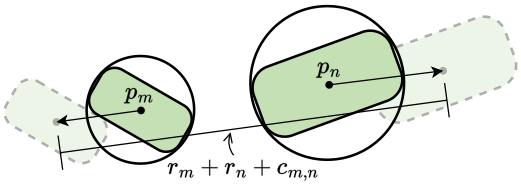


Fig. 5. Our pad-to-pad clearance model between two pads. Each pad, p_m and p_n, is represented by a bounding circle with radius r_m and r_n, respectively. The model ensures that p_m and p_n are moved apart if their center-to-center distance is less than the required spacing $r_m + r_n + c_{m,n}$.

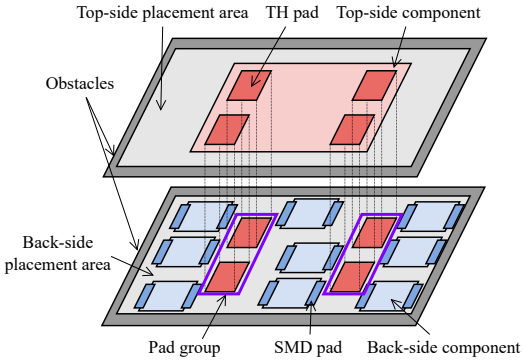


Fig. 6. Illustration of the two-sided density model. The model consists of two parts: the density penalty for the top side and the density penalty for the backside. Each density penalty corresponds to a placement area that includes obstacles, components on the same layer, and TH pads from the opposite side. TH pads are clustered into groups to prevent components from getting trapped between two TH pads.

3) Two-sided Density Model: We propose a two-sided density model to accurately distribute components on both sides of the board. Unlike previous models, ours accounts for interference from the opposite side due to TH pads and accurately calculates the density map using a coarsened grid. Figure 6 provides an overview of this two-sided density model. The back-side density penalty D_b spreads the blocks located in the back-side placement area, including the back-side components, SMD pads, and TH pads from the top side. The TH pads belonging to the same component will be further clustered into pad groups, effectively preventing back-side components from getting stuck in the middle of two TH pads. The top-side density penalty D_t is similarly defined.

Given that any shape, whether a component, a pad, or an obstacle, can be placed in various orientations and may need to be flipped (e.g., when placed on the back side), each shape is represented as a set of counter-clockwise vertices in a matrix form, M_s, as follows:

$$M_s = \begin{bmatrix} x \\ y \end{bmatrix} \mathbf{1} + M_f M_r M_o, \tag{10}$$

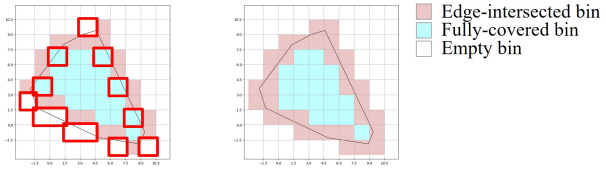
where (x, y) is the reference point (typically the component's center), and M_o, M_f, and M_r represent the contour offset matrix, flip matrix, and rotation matrix, respectively.

For non-convex shapes, we determine the need for conversion to a convex hull by calculating the area ratio $t = A_{real}/A_{cvx}$, where A_{real} is the actual area of the shape, and A_{cvx} is the area of its convex hull. If the ratio falls below a user-defined threshold, we partition the shape into convex sub-polygons; otherwise, we approximate the non-convex shape with its convex hull.

Once the exact contour of each shape is established, the density map $\rho(x, y)$ is updated using a supercover line-drawing algorithm from computational geometry [19]. The supercover line-drawing algorithm calculates the intersected bins by iterating through the adjacent vertices in M_s. This approach differs from previous methods by ensuring that every intersected bin is covered, preventing omissions. Figure 7 illustrates the difference between the prior approach [10] and our method. By covering all edge-intersected bins, our model enables efficient and robust density calculation with a coarsened grid.

D. Gradient Optimization

We employ a combination of Nesterov's method [18] and the Conjugate Gradient method [20] to optimize Equation 2. Nesterov's method, while



(a) Previous work. (b) Ours.

Fig. 7. An illustration of the differences between the density calculation methods in previous work and our approach. (a) Density calculation in the previous work [10]. Mismatched bins, circled in red, lead to over- or under-estimation of density, affecting the spreading direction when using a coarse grid. (b) Our supercover density calculation algorithm accurately computes intersected bins, ensuring greater precision.

achieving faster convergence, can become unstable as components spread gradually. In contrast, the Conjugate Gradient method converges more slowly but provides greater stability as optimization nears a steady point. Thus, Nesterov's method is applied in the initial phase of optimization, followed by the Conjugate Gradient method in the later phase.

The initial values for the Lagrange multipliers λ_1 and λ_2 in Equation 2 are set as $|\nabla \tilde{W}|/|\nabla C_{p2p}|$ and $|\nabla \tilde{W}|/|\nabla D|$, respectively, to balance the gradient norms between the total wire area, the total pad-to-pad clearance violation, and the density penalty. λ_1 and λ_2 are incrementally increased by a user-defined factor (e.g., 1.007) whenever the objective function fails to improve every k times, with k typically set to 3.

The optimization progress is tracked using the overflow ratio, which measures the extent of overlapping areas in the current solution and is defined as follows:

$$\frac{\sum_{b \in \rho_{\text{top}}} \max(N_b - A_b, 0) + \sum_{b \in \rho_{\text{back}}} \max(N_b - A_b, 0)}{A_V + 2A_{\text{TH}}}, \tag{11}$$

where A_V and A_{TH} denote the total area of all components and all TH pads, respectively; ρ_{top} and ρ_{back} are the density maps for the top and back sides of the board; N_b is the area of components in bin b; and A_b represents the bin area. If the overflow ratio exceeds a user-defined threshold, Nesterov's method is applied. Otherwise, the Conjugate Gradient method is used. The global placement process is completed once the overflow ratio reaches or falls below a target threshold.

V. Legalization

After the global placement, we address remaining overlaps and pad-to-pad clearance violations using quadratic programming. Traditional legalization algorithms are often incompatible with heterogeneous components and pads, so we develop a quadratic program formulation utilizing the separating axis theorem to address these issues.

Given two convex shapes (i.e., components and pads), represented in matrix form as M_s^1 and M_s^2, according to the separating axis theorem, if two convex shapes do not intersect, there exists a line (the separating axis) defined by $\mathbf{u} \neq 0$, where $\mathbf{u} \in \mathbb{R}^2$ and $\gamma \in \mathbb{R}$, such that:

$$\mathbf{u}M_s^1 + \gamma \leq 0 \quad \text{and} \quad \mathbf{u}M_s^2 + \gamma \geq 0. \tag{12}$$

The position of each shape is expressed relative to its reference location using a linear equation (Equation 10). For non-intersecting shapes, we identify the separating axis using the GJK algorithm. For intersecting shapes, the separating axis is derived from the minimum penetration vector obtained through the EPA algorithm. Non-convex shapes are decomposed into convex sub-polygons, enabling us to find separating axes between each sub-polygon and other shapes. This approach effectively transforms non-overlapping constraints into linear constraints. Similarly, by adding a separating axis between a boundary obstacle O' and a shape, we ensure that the shape remains within the board area.

The pad-to-pad clearance constraints are represented as the minimum point-to-line distance between each shape and its corresponding separating axes. According to the point-to-line distance formula, if two shapes must be separated by a minimum distance $c_{m,n}$, the clearance constraints can be rewritten as follows:

$$\frac{\mathbf{u}M_s^1 + \gamma}{|\mathbf{u}|} \leq -\frac{1}{2}c_{m,n} \quad \text{and} \quad \frac{\mathbf{u}M_s^2 + \gamma}{|\mathbf{u}|} \geq \frac{1}{2}c_{m,n}. \tag{13}$$

Let $(\hat{x}_i, \hat{y}_i)$ denote the coordinates of a component v_i after legalization. The primary goal of legalization is to satisfy these constraints while maintaining alignment with the global placement result. Therefore, we

TABLE II

THE STATISTICS OF THE BENCHMARK SUIT USED IN NS-PLACE. #FIXED DENOTES THE NUMBER OF FIXED COMPONENTS. #LAYERS DENOTES THE NUMBER OF ROUTING LAYERS.

Design	Width	Height	$\|V\|$	#Fixed	#Pads	$\|E\|$	#Layers
PCB1	20.5	13.9	8	1	40	15	2
PCB2	50.8	22.9	18	5	77	34	2
PCB3	55.0	28.0	34	4	138	38	2
PCB4	22.0	60.0	28	6	140	52	2
PCB7	50.8	22.9	46	2	207	69	2
PCB10	101.6	53.3	57	16	319	99	2
PCB12	58.0	59.5	58	4	233	35	4
PCB13	86.0	71.5	61	3	314	63	4

minimize the displacement from the global placement, represented as the total quadratic moving distance:

$$(x-\hat{x})^2+(y-\hat{y})^2. \tag{14}$$

To enhance robustness, each separating axis is allowed to slide along its normal direction. This flexibility is incorporated by introducing a support variable $\hat{\gamma}$ for each separating axis and adding its quadratic sliding distance $(\gamma-\hat{\gamma})^2$ to the objective function.

Combining these elements, let Γ represent the set of all separating axes. Our quadratic programming model is formulated as follows:

$$\min_{\hat{\mathbf{x}},\hat{\mathbf{y}},\hat{\gamma}} \quad \sum_{v_i\in V}\left[(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2\right]+\sum_{(i,j)\in\Gamma}(\gamma_{i,j}-\hat{\gamma}_{i,j})^2, \tag{15}$$

$$\text{s.t.} \quad \mathbf{u}_{i,j}M_s^i+\hat{\gamma}_{i,j}\le 0 \quad \text{and} \quad \mathbf{u}_{i,j}M_s^j+\hat{\gamma}_{i,j}\ge 0, \quad \forall i\in V,\forall j\in O', \tag{16}$$

$$\mathbf{u}_{i,j}M_s^i+\hat{\gamma}_{i,j}\le 0 \quad \text{and} \quad \mathbf{u}_{i,j}M_s^j+\hat{\gamma}_{i,j}\ge 0, \quad \forall i,j\in V_t\cup P^T, \tag{17}$$

$$\mathbf{u}_{i,j}M_s^i+\hat{\gamma}_{i,j}\le 0 \quad \text{and} \quad \mathbf{u}_{i,j}M_s^j+\hat{\gamma}_{i,j}\ge 0, \quad \forall i,j\in V_b\cup P^T, \tag{18}$$

$$\frac{\mathbf{u}_{i,j}M_s^i+\hat{\gamma}_{i,j}}{|\mathbf{u}_{i,j}|}\le -\frac{c_{m,n}}{2} \quad \text{and} \quad \frac{\mathbf{u}_{i,j}M_s^j+\hat{\gamma}_{i,j}}{|\mathbf{u}_{i,j}|}\ge \frac{c_{m,n}}{2}, \quad \forall i,j\in P_t^S\cup P^T, \tag{19}$$

$$\frac{\mathbf{u}_{i,j}M_s^i+\hat{\gamma}_{i,j}}{|\mathbf{u}_{i,j}|}\le -\frac{c_{m,n}}{2} \quad \text{and} \quad \frac{\mathbf{u}_{i,j}M_s^j+\hat{\gamma}_{i,j}}{|\mathbf{u}_{i,j}|}\ge \frac{c_{m,n}}{2}, \quad \forall i,j\in P_b^S\cup P^T. \tag{20}$$

In this model, Equation 15 defines the objective function, while Equations 16, 17, 18, 19, and 20 specify the placement region constraints, top-side non-overlapping constraints, back-side non-overlapping constraints, and both top-side and back-side pad-to-pad clearance constraints, respectively.

The direction of each separating axis plays a crucial role in ensuring the feasibility of the quadratic programming model. To enhance the robustness of our legalizer, a two-step approach is adopted. In the first step, Equations 15, 16, 17, and 18 are solved to produce an initial non-overlapping layout. In the second step, the separating axes are updated based on the results of the first step, and Equations 15 to 20 are resolved using the updated axes to refine the placement. Further, TH pads are clustered into groups to prevent components from being trapped between two TH pads on opposite sides, further improving the robustness of the model. If any step results in an infeasible model, the output from the previous step is returned as the final solution.

VI. EXPERIMENTAL RESULTS

The proposed algorithm was implemented in C++ and Python, utilizing the OpenMP library for multi-threading. The Gurobi Optimizer [21] was employed as the quadratic programming solver. All experiments were conducted on a Linux workstation equipped with AMD Ryzen 3990X 2.9GHz processors and 128 GB of memory, with the number of CPU threads restricted to eight.

Our PCB placement algorithm was compared with two state-of-the-art academic placers: NS-Place [9] and the placer developed by Tsou *et al.* [11] Table II provides statistics for the benchmark suite used by

TABLE III

THE STATISTICS OF THE INDUSTRIAL BENCHMARK SUIT. #FIXED DENOTES THE NUMBER OF FIXED COMPONENTS, #LAYERS DENOTES THE NUMBER OF ROUTING LAYERS, AND P^S DENOTES THE SET OF TOP- AND BACK-SIDE SMD PADS.

Design	Width	Height	$\|V_t\|$	$\|V_b\|$	#Fixed	$\|P^S\|$	$\|P^T\|$	$\|E\|$	#Layers
CB1	48.0	48.0	14	44	5	98	33	36	1
CB2	82.8	35.2	15	72	4	175	31	53	1
CB3	102.0	44.5	20	69	5	157	42	50	1
CB4	124.5	22.0	108	98	5	520	27	135	2
CB5	129.0	79.1	39	159	12	357	99	108	2
CB6	68.2	30.5	88	119	3	533	55	128	4

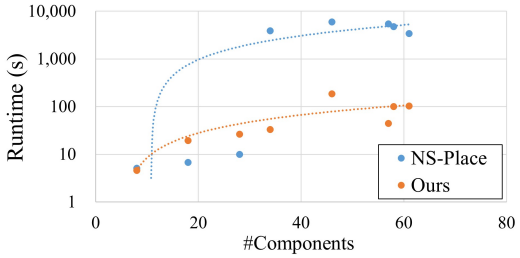


Fig. 8. The scalability comparison with NS-Place.

NS-Place and its predecessor, PCBRouter [22]. This benchmark suite was sourced from GitHub [23]. Due to confidentiality restrictions, PCB6, PCB11, and PCB14 were unavailable, as clarified by the original authors. Moreover, PCB5, PCB8, and PCB9 were removed because they were inconsistent with NS-Place. Note that we found some mismatches in the number of locked components compared to the values reported in NS-Place and made every effort to reconcile these differences.

Table III summarizes another benchmark suite that includes challenging modern industrial cases. These cases feature highly heterogeneous components across various design styles and demand strict pad-to-pad clearance constraints, making them suitable for comparison with the placer by Tsou *et al.*. **Note that achieving high routability in these industrial cases is hard, according to the provided company.** To evaluate the placement quality, the benchmarks in Table II were routed using FreeRouting [24], the same tool employed by NS-Place, and the design rule violations (DRVs) were evaluated using KiCad 6.0 [3]. The DRVs are listed as follows:

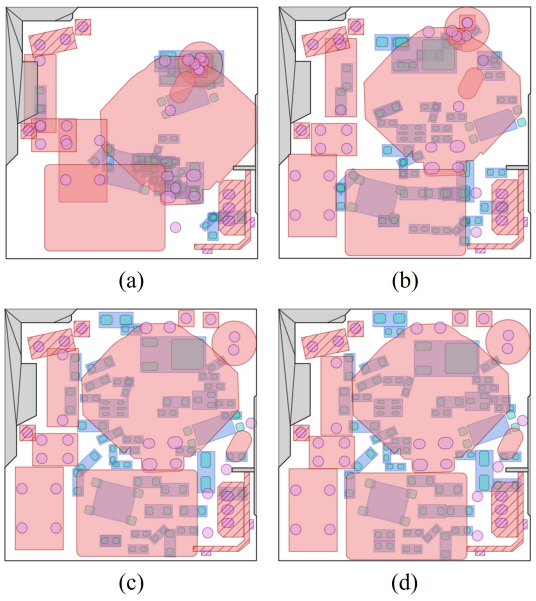


Fig. 9. The placement snapshots of CB1. (a) Initial placement: Components are centered and arranged to minimize the total XHPWL. (b) During global placement: Components are distributed by solving Equation 2. (c) After global placement: Components are positioned with minimal overlap and clearance constraint violations. (d) After legalization: Remaining overlaps and clearance violations are resolved.

TABLE IV
COMPARISON BETWEEN NS-PLACE AND OUR PLACER. HPWL, RT, PRWL, AND #DRVS DENOTE THE HPWL AFTER PLACEMENT, PLACEMENT RUNTIME, POST-ROUTING WIRELENGTH, AND NUMBER OF DESIGN RULE VIOLATIONS (INCLUDING UNROUTED/SHORTED NETS).

Design	NS-Place [9]					Ours				
	HPWL(mm)	RT(s)	PRWL(mm)	#Vias	#DRVs	HPWL(mm)	RT(s)	PRWL(mm)	#Vias	#DRVs
PCB1	72.10	5.10	121.00	4	0	**68.56**	4.61	**83.02**	1	0
PCB2	296.18	6.80	421.00	5	23	**275.88**	19.45	**296.57**	0	0
PCB3	**271.30**	3917.60	616.00	15	0	322.38	33.26	**463.84**	9	0
PCB4	**531.19**	9.90	806.00	54	0	667.50	26.55	**702.27**	9	0
PCB7	1601.40	6008.20	2949.00	93	0	**779.26**	184.73	**971.96**	51	0
PCB10	3315.46	5417.20	4914.00	97	0	**3228.83**	44.32	**3442.34**	73	0
PCB12	941.30	4752.30	1720.00	45	0	**885.74**	100.87	**1280.82**	15	0
PCB13	2151.77	3416.30	2897.00	83	0	**1876.17**	103.62	**2337.04**	61	0
Compare	1.00	1.00	1.00	1.00	-	**0.95**	**0.82**	**0.70**	**0.48**	-

TABLE V
COMPARISON BETWEEN TSOU ET AL.'S PLACER AND OUR PLACER. #CV DENOTES THE NUMBER OF CLEARANCE VIOLATIONS AND ROUT. DENOTES THE ROUTABILITY.

Design	Tsou *et al.* [11]					Ours				
	HPWL (mm)	XHPWA (mm^2)	#CV	RT (s)	Rout.	HPWL (mm)	XHPWA (mm^2)	#CV	RT (s)	Rout.
CB1	580.47	456.91	79	36.55	0.60	584.07	483.95	77	84.71	**0.82**
CB2	1176.25	1430.22	540	52.51	0.44	855.42	724.72	67	201.59	**0.81**
CB3	1330.15	1407.12	439	63.32	0.54	856.38	822.91	45	149.24	**0.82**
CB4	4661.46	1636.76	1557	84.91	0.40	1708.20	582.31	56	436.33	**0.90**
CB5	3185.41	3008.72	175	181.05	0.64	2085.04	1914.83	53	1005.18	**0.84**
CB6	4112.86	1365.94	1228	67.71	0.48	2031.99	682.82	29	409.92	**0.82**
Avg.	2507.77	1550.95	669.67	81.01	0.52	1353.52	868.59	54.50	381.16	**0.84**
Comp.	1.00	1.00	1.00	**1.00**	-	**0.65**	**0.61**	**0.26**	4.21	**-**

out-of-boundary blocks, overlapping components, pad-to-pad clearance violations, and open/short nets. The benchmarks in Table III were routed using the in-house commercial router applied in Tsou *et al.*'s work and evaluated the pad-to-pad clearance violations (CVs) using the same in-house design rule checker used by Tsou *et al.*.

A. Performance on Academic Benchmarks

Table IV compares the placement and routing results of our algorithm against NS-Place, with NS-Place results cited from the original paper [9]. Key metrics such as HPWL and runtime (RT) are used to evaluate placement effectiveness and efficiency, along with the post-routing wirelength (PRWL), the number of vias (#Vias), and the number of design rule violations (#DRVs) to assess the routability of the placement results. The results show that, on average (normalized), our placer achieved improvements of 5% in HPWL, 18% in runtime, 30% in PRWL, and 52% in via usage. Further, no DRVs were found in our results, whereas NS-Place reported 23 DRVs in PCB2.

The improvements in placement quality can be attributed to several factors: (1) NS-Place aims to separate each net's routing region, which often leads to over-spreading of components. (2) Our wire-area model accounts for the 135-degree routing nature, providing a more accurate estimation of routing resources. Figure 8 compares scalability, showing that our placer achieves a speedup of 33X to 100X in large cases. This performance gain is primarily due to the effectiveness of the quadratic programming legalizer we adopted, which outperforms the mixed-integer linear programming legalizer used in NS-Place.

B. Performance on Modern Industrial Benchmarks

Table V compares our algorithm with Tsou *et al.*'s placer, where the results for Tsou *et al.*'s placer were obtained by running their binary. Key metrics such as HPWL, XHPWA, the number of clearance violations (#CV), and runtime (RT) are used to evaluate placement effectiveness and efficiency. Routability is included to assess placement quality. The results show that, on average (normalized), our algorithm outperformed Tsou *et al.*'s placer by 35% in HPWL, 39% in XHPWA, 74% in clearance constraint violations, and 32% in routability. These improvements are attributed to the proposed models in Section IV, which handle clearance constraints and heterogeneous components more effectively. Unlike Tsou *et al.*'s approach, which addresses clearance constraints only during the legalization stage, our algorithm incorporates clearance constraints during global placement. This distinction reduces large displacements and improves routability, demonstrating the importance of addressing clearance constraints early in the placement process. The runtime overhead in our results is mainly due to constructing quadratic programs with separating axes. Our algorithm is shown to be a highly effective and efficient solution for capturing complex geometric relationships, as evidenced by the substantial reduction in clearance violations. Importantly, manually addressing these violations is both highly time-consuming and labor-intensive, demonstrating that the trade-off in runtime is necessary for achieving significant gains in modern PCB designs.

Figure 9 shows snapshots of the placement process with our algorithm. In the initial placement phase, blocks were centered on the board. During global placement, blocks were distributed by clearance and density constraints. Finally, the proposed quadratic programming legalizer resolved remaining overlaps and clearance violations, ensuring a high-quality placement result.

VII. CONCLUSION

This paper has presented a novel PCB placement framework considering pad-to-pad clearance constraints and heterogeneous components. Unlike existing academic placers, our framework has incorporated the following features: (1) a wire-area model to differentiate routing resource consumption between power and signal nets, (2) a pad-to-pad clearance model to minimize spacing violations, and (3) a two-sided, pad-type-aware density model to reduce component(pad) overlap. We have also developed a quadratic programming-based legalizer to remove constraint violations between components with arbitrary shapes. Experimental results have demonstrated the effectiveness and efficiency of our approach, with our algorithm outperforming two state-of-the-art academic placers in post-routing quality on both academic and industrial benchmarks. Specifically, our algorithm has shown an average improvement of 30% in post-routing wirelength over NS-Place, an average reduction of 74% in clearance violations over Tsou *et al.*'s work, and an average improvement of 32% in routability over Tsou *et al.*'s work. These results have justified the practical applicability of our PCB placement solution.

VIII. ACKNOWLEDGMENT

We thank Mr. Pin An Chen, Mr. Wei No Chen, Mr. Sean Huang, Ms. Ann Wei, Mr. Charles Liu, and Mr. Denver Liu of Delta Electronics for their great help in this research.

REFERENCES

[1] Allegro X design platform. [Online]. Available: https://www.cadence.com/en_US/home/tools/pcb-design-and-analysis/allegro-x-design-platform.html

[2] Altium designer. [Online]. Available: https://www.altium.com

[3] KiCad EDA: A cross-platform and open source electronics design automation suite. [Online]. Available: https://www.kicad.org

[4] W.-H. Liu, A. Agnesina, and H. M. Ren, "Challenges for automating PCB layout," in *Proc. of ISPD*, Taipei, Taiwan, March 2024, pp. 91–92.

[5] Y.-W. Chang, "Physical design challenges in modern heterogeneous integration," in *Proc. of ISPD*, Taipei, Taiwan, March 2024, pp. 125–134.

[6] Renesas. [Online]. Available: https://www.renesas.com/en

[7] F. S. Ismail, R. Yusof, and M. Khalid, "Optimization of electronics component placement design on PCB using self organizing genetic algorithm (SOGA)," *Springer JOIM*, vol. 23, no. 1, pp. 883–895, 2012.

[8] T. Badriyah, F. Setyorini, and N. Yuliawan, "The implementation of genetic algorithm and routing Lee for PCB design optimization," in *Proc. of ICIC*, Mataram, Indonesia, October 2016, pp. 148–153.

[9] C.-K. Cheng, C.-T. Ho, and C. Holtz, "Net separation-oriented printed circuit board placement via margin maximization," in *Proc. of ASPDAC*, Taipei, Taiwan, January 2022, pp. 288–293.

[10] F. Huang, D. Liu, X. Li, B. Yu, and W. Zhu, "Handling orientation and aspect ratio of modules in electrostatics-based large scale fixed-outline floorplanning," in *Proc. of ICCAD*, San Francisco, California, October/November 2023, pp. 1–9.

[11] C.-H. Tsou, S.-Y. Lin, W.-C. Hung, and Y.-W. Chang, "Late breaking results: Modern automatic PCB placement with complex constraints," in *Proc. of DAC*, San Francisco, California, June 2024, pp. 1–2.

[12] Y.-J. Chen, C.-H. Hsieh, P.-H. Su, C. Shao-Hsiang, and Y.-W. Chang, "Mixed-size 3D analytical placement with heterogeneous technology nodes," in *Proc. of DAC*, San Francisco, California, June 2024, pp. 1–6.

[13] M. Su, Y. Xiao, S. Zhang, H. Su, J. Xu, H. He, Z. Zhu, J. Chen, and Y.-W. Chang, "Subgraph matching based reference placement for PCB designs: late breaking results," in *Proc. of DAC*, San Francisco, California, July 2022, pp. 1–2.

[14] T.-C. Chen, Y.-L. Chuang, and Y.-W. Chang, "X-architecture placement based on effective wire models," in *Proc. of ISPD*, Austin, Texas, March 2007, pp. 87–94.

[15] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "TSV-aware analytical placement for 3-D IC designs based on a novel weighted-average wirelength model," *IEEE TCAD*, vol. 32, no. 4, pp. 497–509, 2013.

[16] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE JORAA*, vol. 4, no. 2, pp. 193–203, 1988.

[17] G. Van Den Bergen, "Proximity queries and penetration depth computation on 3D game objects," in *Proc. of GDC*, San Jose, California, March 2001, p. 209.

[18] J. Lu, P. Chen, C.-C. Chang, L. Sha, D. J.-H. Huang, C.-C. Teng, and C.-K. Cheng, "ePlace: Electrostatics-based placement using fast fourier transform and Nesterov's method," *ACM TODAES*, vol. 20, no. 2, pp. 1–34, 2015.

[19] Bresenham-based supercover line algorithm. [Online]. Available: http://eugen.dedu.free.fr/projects/bresenham

[20] T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen, and Y.-W. Chang, "NTUplace3: An analytical placer for large-scale mixed-size designs with preplaced blocks and density constraints," *IEEE TCAD*, vol. 27, no. 7, pp. 1228–1240, 2008.

[21] Gurobi optimizer reference manual. [Online]. Available: http://www.gurobi.com

[22] T.-C. Lin, D. Merrill, Y.-Y. Wu, C. Holtz, and C.-K. Cheng, "A unified printed circuit board routing algorithm with complicated constraints and differential pairs," in *Proc. of ASPDAC*, Tokyo, Japan, January 2021, pp. 170–175.

[23] PCB benchmarks. [Online]. Available: https://github.com/aspdac-submission-pcb-layout/PCBBenchmarks

[24] FreeRouting. [Online]. Available: https://freerouting.org

FlexStep: Enabling Flexible Error Detection in Multi/Many-core Real-time Systems

Tinglue Wang†, Yiming Li†, Wei Tang†, Jiapeng Guan‡, Zhenghui Guo†, Renshuang Jiang§, Ran Wei¶*, Jing Li‖*, Zhe Jiang†*
†Southeast University, China. ‡Dalian University of Technology, China. §National University of Defense Technology, China.
¶Lancaster University, UK. ‖New Jersey Institute of Technology, US. *The corresponding authors.

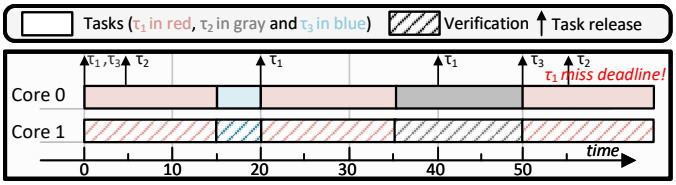
***Abstract*—Reliability and real-time responsiveness in safety-critical systems have traditionally been achieved using error detection mechanisms, such as LockStep, which require pre-configured checker cores, strict synchronisation, static error detection regions, or limited preemptions. However, these core-bound hardware mechanisms often lead to significant resource over-provisioning and diminished real-time performance in modern systems where tasks with varying reliability requirements are consolidated on shared processors for efficiency and cost reduction. To address these challenges, this work presents FlexStep, a systematic solution that integrates hardware and software across the SoC, ISA, and OS scheduling layers. FlexStep features a novel microarchitecture that supports dynamic core configuration and asynchronous, preemptive error detection. The FlexStep architecture naturally allows for flexible task scheduling and error detection, enabling new scheduling algorithms that enhance both resource efficiency and real-time schedulability.**

I. Introduction

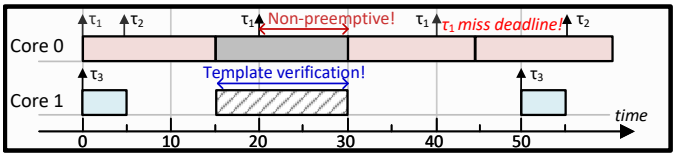
In safety-critical systems, such as automotives and avionics, processor cores must simultaneously detect and correct faults during execution [1]–[4] and guarantee task schedulability [5]–[7] as the foundational components to ensure reliable system. However, since reliability and schedulability address distinct safety dimensions, they are usually achieved through different development phases and across various architectural levels. Reliability is often ensured through hardware-level error detection mechanisms, such as LockStep used in ARM Cortex R series [8]–[11]. While software mechanisms exist [12]–[15], they offer limited coverage and degrade performance significantly. Conversely, schedulability is ensured through scheduling algorithms implemented in the Operating System (OS) [16]–[18].

Existing works. As hardware computational capabilities advance and software applications diversity expands, rising demand emerges to execute multiple tasks with varying reliability requirements on shared processor cores [19]–[23]. In this context, LockStep technology faces significant limitations due to its rigid hardware design. LockStep binds one or more identical cores, synchronously executing same program and comparing output results at each cycle. This design mandates that all tasks undergo the highest level of error detection, irrespective of their actual reliability needs, which leads to inefficient use of error detection capabilities, suboptimal resource utilisation, significant power and area overheads, and challenges to system schedulability. As Fig.1(a) demonstrates, LockStep's mandatory error checking for non-verification tasks like τ_1 and τ_3 causes unnecessary resource consumption and the third job of τ_1 to miss its deadline.

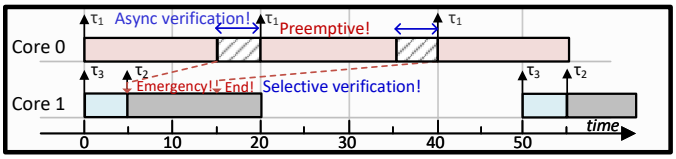
Methods supporting reconfiguration, such as LockStep with split-lock [24]–[26], have been developed to reduce resource overhead. For instance, the most recent Hybrid Modular Redundancy (HMR) approach [24] explicitly separates mission-critical and performance-critical code sections. It leverages hardware designs to enable runtime reconfiguration for split-lock, providing a degree of flexibility. However, the inherent limitations of the "core binding" still impose significant constraints. Specifically, when reconfigured into verification mode, checker cores must perform error detection synchronously with main core. Moreover, this synchronous error-checking execution



(a) LockStep configuration with fixed main core 0 & checker core 1.



(b) HMR [24] configuration with limited flexibility and synchronous checking.



(c) FlexStep allowing asynchronous, selective, and preemptive checking.

Fig. 1: Schedules on different dual-core architectures. τ_1, τ_2, τ_3 have implicit deadlines and worst-case execution time of 15, 15, 5. Non-verification tasks τ_1 and τ_3 do not require checking. An emergency occurs when τ_2's first job arrives and requires checking its first 10 units of work.

cannot be preempted by non-verification tasks, even if those tasks have higher priorities or earlier deadlines. Additionally, HMR can only statically perform checking for pre-determined tasks, lacking the capability to provide selective error checking based on dynamic system requirements. These constraints significantly reduce flexibility and impair schedulability. As shown in Fig. 1(b), although HMR with runtime split-lock reduces resource wastage by not performing error checking for non-verification tasks τ_1 and τ_3 (τ_3 can be executed on core 1), the "core binding" design still prevents τ_1 from preempting τ_2's error checking, resulting in τ_1 missing its second deadline.

Contributions. To guarantee reliability and schedulability for safety-critical systems while maintaining resource efficiency, developing more flexible hardware error detection architecture has become an industry necessity [26]–[28]. As illustrated in Fig. 1(c), such architecture should decouple cores from rigid LockStep and employ a flexible error detection mechanism. This mechanism, with OS support, should enable *error detection that is asynchronous, selective, and preemptable* by non-verification tasks. However, achieving this level of flexibility and efficiency presents significant technical challenges.

To address these challenges, we present FlexStep, a comprehensive full-stack framework that integrates hardware and software to deliver high flexibility and configurability building on three key technologies. First, we modified the Rocket [29] microarchitecture to enable

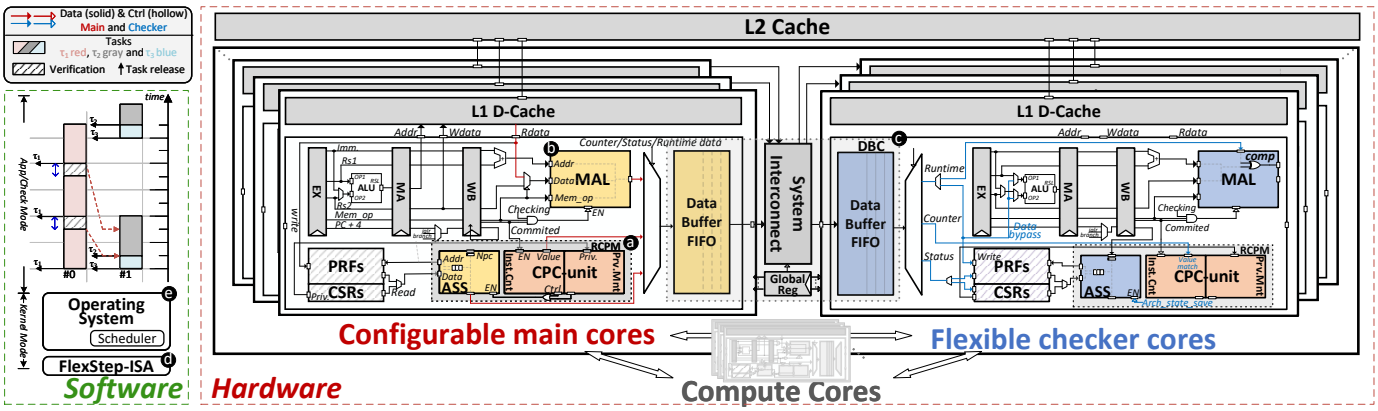


Fig. 2: FlexStep overview. At the hardware level, colored modules represent functional units added by FlexStep: orange hues denote identical functionality, while yellow and blue hues highlight variations in functionality for different cores. ⓐ Register Checkpoint Management (Sec. III-A). ⓑ Memory Access Log (Sec. III-B). ⓒ Data Buffering and Channelling (Sec. III-C). At the software level, FlexStep provides ⓓ customised ISA (Tab. I) and enables ⓔ a control flow to perform context switching between verification and non-verification tasks and flexible scheduling.

dynamic hardware reconfiguration and asynchronous error detection at thread level, establishing foundation for scheduling flexibility. Second, through abstraction of hardware configurations into a customized Instruction Set Architecture (ISA), we integrate runtime hardware control with OS context switching, allowing dynamic task preemption and resource reallocation. Third, the proposed full-stack framework systematically coordinates multicore architecture extensions with OS scheduling primitives, while our formalized theoretical model enables novel scheduling algorithms that exploit hardware flexibility to improve real-time schedulability. This approach overcomes the limitations of synchronous, core-binding methods through preemptable error detection and adaptive resource management for tasks with varying reliability requirements. We publicly release FlexStep's source code, at *https://anonymous.4open.science/r/FlexStep-DAC25-7B0C*.

We implemented FlexStep on AMD Alveo U280 FPGA and evaluated it using various metrics. Results demonstrate that FlexStep achieves microsecond-level detection latency with 1.07% slowdown, 2.21% area overhead, and 2.89% power consumption while delivering improved schedulability compared to LockStep and HMR.

II. FlexStep: Hardware/Software Co-design Framework

In FlexStep, any processor core can be configured as a main or checker core. The main core executes applications like a standard core, while checker core verifies the correctness of main core by re-running the same program. Fig. 2 illustrates the FlexStep framework.

Unlike the synchronised verification in LockStep, FlexStep employs error detection based on Register Checkpoints (RCPs) – architectural states at specific points – and memory accesses, similar to Paramedic [30]–[32]. Specifically, a user thread running on main core is divided into small checking segments, which can be reproduced on its associated checker core(s). Each segment is identified by ***Start RCPs*** (***SCPs***) and ***End RCPs*** (***ECPs***) stored in RCP Management units (Fig. 2.ⓐ, Sec. III-A). The checker core halts memory access and sequentially replays the segments by initializing its architectural state to the segment's SCPs and compares its final state to the ECPs. During execution, memory access data (e.g., addresses and data for `LD/ST`, `LR/SC`, `AMO` instructions) recorded in Memory Access Log (Fig. 2.ⓑ, Sec. III-B) are used for replay execution and verified at runtime. Main core execution is deemed correct if the ECPs of all checking segments and memory access data match the original traces.

The fundamental idea is that, as long as all data related to RCPs and memory accesses – required for execution replay and verification – are recorded and buffered, the checker thread can be executed *asynchronously* on any other core. Therefore, checker core can execute other tasks instead of verifying, as long as the necessary data of verification task have been extracted from main core. This design improves both utilisation and flexibility. FlexStep allows user threads running on any core to be duplicated and verified on different core(s) using a data buffer and ***configurable interconnected channel*** (Fig. 2.ⓒ, Sec. III-C). Unlike LockStep and HMR, which supports specific modes such as Dual-Core LockStep (DCLS) and Triple-Core LockStep (TCLS) by binding cores via shared input path , FlexStep's interconnected channel can be configured to operate in one-to-one (similar to DCLS), one-to-two (similar to TCLS), or more modes. This accommodates scenarios with varying safety-criticality demands.

To manage cores and schedule error detection, FlexStep abstracts the configurable hardware control interfaces as a ***customised ISA*** (Fig. 2.ⓓ, Tab. I) and integrates a control flow into OS. The ISA defines the instructions specific to main or checker core and global instructions, exposing all ***core attributes*** (main, checker, or compute) to OS for runtime configuration and scheduling facilitation (Fig. 2.ⓔ). Preemptive execution is supported, allowing any core to be interrupted and enabling more urgent tasks to be executed. Thus, the OS can dynamically configure core attributes and verification modes as needed, leveraging asynchronous verification and task preemption to optimise real-time scheduling while ensuring reliability.

III. Microarchitecture (μ-arch)

Implementing hardware-level FlexStep requires μ-arch modifications adding several key functional units, as illustrated in Fig. 2. We used the Rocket core as a case study to implement our framework and evaluate its effectiveness. Noted that incorporating identical units into each core is crucial for dynamic attribute switching, which requires maximising functional unit reuse to enhance utilisation and reduce area costs. This section discusses the details of the modified μ-arch.

A. Register Checkpoint Management Units (RCPM)

RCPM (Fig. 2.ⓐ) consists of ***Checkpoint Control*** (***CPC***) and ***Architectural State Snapshot*** (***ASS***). CPC controls the start and end of a checking segment in the main core, and identify segment boundaries and verify execution correctness in the checker core. ASS is a storage unit, which captures Register Checkpoint snapshots and releases them for transmitting or applying under the control of CPC.

TABLE I: FlexStep ISA, abstracting control interface for software.

Instruction	Description
`G.IDs.contain`	Return core attributes (Main/Checker)
`G.Configure`	Configure the main and checker cores'ID
`M.associate`	Allocate one or multiple checker core(s) to main
`M.check`	Enable/Disable the checking function
`C.check_state`	Switch the checking state (busy/idle)
`C.record`	Record the context to ASS
`C.apply`	Apply the SCP from data channel
`C.jal`	Jump to the next pc (npc) of SCP
`C.result`	Return the comparison result

Checkpoint Control (CPC) includes an instruction counter (Fig. 2. Inst.Cnt) and a privilege mode monitor (Fig. 2. Priv.Mntr). FlexStep only supports user mode checking and all cores could enter kernel during execution, resulting in premature extermination (Fig. 3.①) and temporary deviation (Fig. 3.②) of a checking segment.

CPC's Mechanism. For main core, a new checkpoint is generated when: a) privilege mode switches; b) instruction count limit is reached (default is 5000). At ECP, the counter stops and records the number of instructions of the segment. During the segment, main core sends SCP, `LD/ST` log, instruction count (IC), and ECP in order (Fig. 3) for checker core to receive. For checker core, upon applying SCP, it begins checking till the instruction count reaches the same value as main core before using its own architectural state to verify ECP.

Architectural State Snapshot (ASS) facilitates duplicate execution by temporarily storing Checkpoints and architectural states. It enables immediate access to architectural states and reduces the overhead of frequent memory access, achieving fast execution state switching.

B. Memory Access Log Unit (MAL)

For main core, information required for duplicate execution and verification needs to be recorded and transmitted during checking segments. MAL (Fig. 2.ⓑ) records data and addresses of a memory instruction, packaging them for transmission in the main core, or making comparisons to report potential errors in the checker core.

Handling instructions with multiple micro-ops. Regular `LD/ST` instructions are packaged into a single entry, while multi-micro-op instructions (`LR`, `SC` and `AMO`) use multiple entries to minimize data width. This introduces slight latency but reduces storage overhead, outweighing disadvantages due to their low frequency in workloads.

C. Data Buffering and Channelling Units (DBC)

DBC (Fig. 2.ⓒ) consists of ***Data Buffer FIFO*** to buffer data of checking segments for conflict resolution and asynchronous error checking and ***System Interconnect*** that establishes and alters links of the FIFOs between main cores and their associated checker cores.

Configurable Inter-core Channels. Inter-core communication channels between main and checker cores are achieved through the System Interconnect, a fully connected MUX-DEMUX network laid out between FIFO of each core. A global register, which can be modified by custom instructions, generates control signals for MUX/DEMUX to establish channels between the main core and one or more checker cores. This design eliminates communication conflicts and ensures minimal communication latency at a small scale. As the design scales up, the wiring complexity grows exponentially, and the interconnect could be replaced with a bus interconnect or NoC.

Buffering for Conflict Resolving and Asynchrony. In line with previous work [30]–[32], an SRAM-based FIFO is used for data buffering, enabling the checker core to retrieve information directly from the FIFO instead of main memory. This supports asynchronous verification; with larger capacity comes greater flexibility in scheduling. The main core's FIFO resolves conflicts when multiple main cores compete for a checker core, buffering data until the checker core is available. Additional main memory buffering, accessed via DMA, can further support conflict resolution and asynchrony.

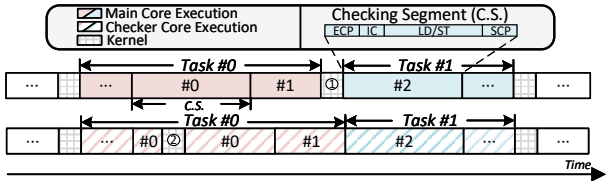


Fig. 3: Checking segments: their components, main and checker execution of them, and them of default length or interrupted by kernel.

IV. New ISA and OS Kernel Add-ons

In FlexStep, the OS plays a pivotal role in managing hardware resources and ensuring real-time schedulability. To enhance its functionality, we leverage a control interface within the ISA (Tab. I), enabling efficient core management with minimal modifications to the OS. This approach ensures compatibility with existing application programs while improving overall scheduling performance.

A. New ISA Support

As shown in Tab. I, the new ISA is classified into three categories for the main core (`M.x()`), checker core (`C.x()`) and both (`G.x()`). Global instructions include `IDs.contain` to identify core attributes and `Configure` to write the IDs of main and checker cores into global configuration registers. For main cores, `associate` allocates checker cores for redundant execution, and `check` controls the checking capability by managing the relevant units. For checker cores, we present `check_state` for changing the checking state of the core, and `record` for recording the architectural state into the ASS. In addition, a pair of "atomic" instructions `apply` and `jal` are presented to replay the checking segment on checker cores.

Algorithm 1: Each core's context switch.

```
1  Function Context_Switch(task *current, core core_id):
2      if (G.Main_IDs.contain(core_id)) then
3          M.check.disable();
4      else if (G.Checker_IDs.contain(core_id)) then
5          C.check_state(idle);
6      Kernel.Intr(DISABLE);
7      task *next = NULL;
8      Kernel.Context.save(current);
9      next = Kernel.Find_next();
10     if (next→new_release) then
11         /* Configure the main and checker core's ID */
12         G.Configure(Main_IDs, Checker_IDs);
13         Kernel.Context.init(next);
14     else then Kernel.Context.restore(next);
15     current = next;
16     Kernel.Intr(ENABLE);
17     if (G.Main_IDs.contain(core_id)) then
18         /* Associate checker cores and enable checking */
19         M.associate(Checker_ID(s));
20         M.check.enable();
21     else if (G.Checker_IDs.contain(core_id) and
           next→checker_thread) then
22         C.check_state(busy);
23     Kernel.Context.jalr(current→pc);
```

The `apply` is used to update the architectural registers, while the `jal` is used to jump to the main thread, which is modified from the standard jump instruction with a designated target from the main cores for accurate branch misprediction handling without necessitating any changes to the microarchitecture. Finally, `result` reports the comparison result at each ECP.

B. OS Kernel Add-ons and Customised Checker Thread

The modification of OS (Fig. 2.ⓔ) is shown in Al. 1, which requires minimal code additions to the scheduler's context switch function. During context switch, cores executes different instructions based on its attributes to switch off checking function (Al. 1: lines 2 - 5). Additionally, when a new task is released, `Configure` (re-)initializes the global registers (Al. 1: line 12). Finally, for main cores, `associate` equips it with the specific checker core(s), while `check_enable` activates the checking function (Al. 1: lines 19 - 20). For checker cores, it will switch to busy state upon receiving verification tasks(Al. 1: line 22) and enter a checker thread.

Besides the general modification of the context switch, we have also developed a dedicated thread for checker cores (Al. 2). Similar to the regular context switch, `record` is used to store current architectural state in ASS for state restoration (Al. 2: line 4). Next, the checker core will enter a loop and continuously request new SCP from the FIFO (Al. 2: line 6). Once a new SCP arrives, `apply` will deploy it to the architectural registers (Al. 2: line 7), followed by `jal` to control the directional jump of PC (Al. 2: line 8), thereby ensuring that the checker core executes the correct sequence of instructions. Lastly, the verification result is returned by `result` (Al. 2: lines 5), and the correction mechanism will be triggered if an error is detected.

Algorithm 2: Customised checker thread.

```
1 Function Checker_Thread() :
2     /* launching checker thread with P-Thread */
3     ...
4     C.record(ASS); // return position after checking
5     if(!C.rslt()) then ReportErr();
6     while (C.NewSCP() != ready);
7     C.apply(C.NewSCP.data);
8     C.jal(C.NewSCP.npc);
```

V. System Model and Scheduling Analysis

We now describe formal model and analysis for scheduling problem with verification ensuring reliability and real-time guarantees.

Model. We seek to schedule a task set consisting of n sporadic tasks on m cores. Each task τ_i is characterised by its worst-case execution time C_i, period T_i, and implicit deadline $D_i = T_i$. Tasks can be classified into three types: (1) A task in $\mathcal{T}^N$ is a non-verification task that only needs to execute its normal work once every period and meet its deadline. (2) A task in $\mathcal{T}^{V2}$ may require double-check online. (3) A task in $\mathcal{T}^{V3}$ may require triple-check online. In the event of an emergency, the system dynamically triggers additional error checking for one or more jobs of specific verification tasks based on the nature of the emergency. Double-check (or triple-check) verification involves duplicating the computation under verification once (or twice) and reproducing it on one (or two) cores distinct from the core performing the original computation.

If asynchronous verification is supported, the duplicated computations can be executed after the original computation but must still be completed before the job deadline to ensure timely error detection and maintain system reliability. As discussed earlier, FlexStep supports both asynchronous verification and selective error checking, allowing error checking to be dynamically triggered for a task and performed on specific portions as needed. As an initial step in modeling of this flexibility, we limit our focus on asynchronous verification only, while assuming that all jobs of verification tasks require full error checking and must meet their deadlines. Without additional information, such as the probability of dynamic verification or assumptions allowing low-criticality tasks to miss deadlines during system emergencies, this formulation represents the system's worst-case behaviour.

Scheduling algorithm and analysis. For this scheduling problem, we propose a simple scheduling algorithm and corresponding schedulability analysis based on the partitioned Earliest Deadline First (partitioned EDF). The algorithm partitions all non-verification tasks, verification tasks, and their duplicated computations for double-check or triple-check verification on m available cores.

Algorithm 3: Scheduling with asynchronous verification.

```
1  Input: Γ: Task set τ_i ∈ {T^N, T^V2, T^V3}, m available cores
2  Output: Deadline-compliant task partitioning
3  for each core k ∈ {1,...,m} do Δ[k] ← 0;
4  for each τ_i ∈ {T^V3, T^V2} do
5      if τ_i ∈ T^V2 then D'_i ← D_i/2;
6      else if τ_i ∈ T^V3 then D'_i ← (√2 − 1)D_i;
7      δ^o_i ← C_i/D'_i; δ^v_i ← C_i/(D_i − D'_i);
8      k ← argmin_{k∈{1,...,m}} Δ[k];
9      Assign τ^o_i to core k; Δ[k] ← Δ[k] + δ^o_i;
10     k' ← argmin_{k'∈{1,...,m}\{k}} δ[k'];
11     Assign τ^v_i to core k'; Δ[k'] ← Δ[k'] + δ^v_i;
12     if τ_i ∈ T^V3 then
13         k'' ← argmin_{k''∈{1,...,m}\{k,k'}} Δ[k''];
14         Assign τ_i^{v'} to core k''; Δ[k''] ← Δ[k''] + δ^v_i;
15 for each τ_i ∈ T^N do
16     δ^o_i ← C_i/D_i;
17     k ← argmin_{k∈{1,...,m}} Δ[k];
18     Assign τ_i to core k; Δ[k] ← Δ[k] + δ^o_i;
19 for each core k ∈ {1,...,m} do
20     if Δ[k] > 1 then return Fail;
21 return Success;
```

Since error-checking computations cannot begin before the original computation, our algorithm assigns a virtual deadline D'_i to each verification task to reserve enough time for verification. This virtual deadline is used for scheduling the original computation on its assigned core using EDF, while the original deadline is used for scheduling the duplicated computations. For a double-check task $\tau_i \in \mathcal{T}^{V2}$, $D'_i = D_i/2$, and for a triple-check task $\tau_i \in \mathcal{T}^{V3}$, $D'_i = (\sqrt{2}-1)D_i$. The virtual deadline is chosen to minimise the sum of the density of the original computation $\delta^o_i = C_i/D'_i$ and the density of the duplicated computation $\delta^v_i = C_i/(D_i - D'_i)$, optimising system schedulability. Note that, in practice, duplicated computations can start as soon as the original computation begins execution. However, our schedulability analysis consider the worst-case scenario — the error-checking starts only after the virtual deadline.

Our partitioning algorithm (Al. 3) partitions verification tasks with descending utilisation and ensures that their original and error-checking computations are allocated to different cores (Al. 3: lines 4-14). Their calculated densities δ^o_i, δ^v_i update the total density of allocated cores. Non-verification tasks are then partitioned with descending utilisations (Al. 3: lines 15-18). As EDF is used to execute tasks allocated to each core, core k is schedulable if its total density

($\Delta[k]$) of assigned tasks does not exceed one. The entire task set is guaranteed to be schedulable if all tasks (and their error-checking) are successfully assigned to cores and all cores remain schedulable (Al. 3: lines 19-20). Since our schedulability test is a sufficient test,when the test fails and hard real-time guarantees are not required, we can remove the virtual deadline and use the verification task's original deadline and utilisation for scheduling and partitioning.

VI. Evaluation

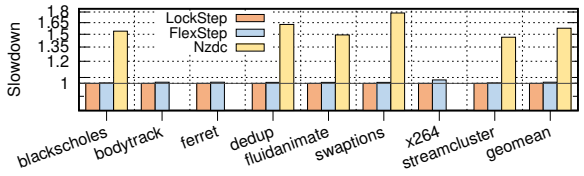
We built FlexStep featuring homogeneous SoC with multiple Rockets and implemented the microarchitecture upon an open-source platform Chipyard [33] (v1.8.0). With TSMC 28nm PDKs [34], we synthesised the RTL using Synopsys (v2019.12). We deployed the RTL design on the AMD Alveo U280 FPGA using FireSim [35] to simulate the settings in Tab II and boot Linux kernel (v5.7), executing full SPECint 06 [36] and Parsec V3 [37] with simmedium dataset.

TABLE II: Hardware configurations evaluated.

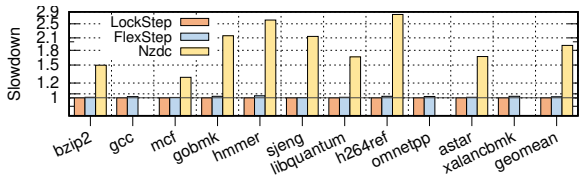
Homogeneous Core	
Core	In-order scalar Rocket, @1.6GHz
Pipeline	5-stage pipeline, 64 Int/FP Phy Registers, 1 ALU, 1 DIV, 1 FPU, 1 CSR
Branch Pred.	512-entry BHT, 28-entry BTB, 6-entry RAS
Memory Hierarchy	
L1 I-Cache	16 KB, 4-way, Blocking, 2 LatencyCycles
L1 D-Cache	16 KB, 4-way, Blocking, 2 LatencyCycles
L2 Cache	512 KB, 8-way, 8 MSHRs, 40 LatencyCycles

A. Performance Overhead

Experiment setup. We executed SPECint and Parsec with dual-core mode and compared the performance slowdown of Flexstep with the most widely used hardware scheme Lockstep, and the only available open-source software error-detection scheme Nzdc [38] which fails to compile on some workloads (e.g., bodytrack, ferret, gcc).



(a) Parsec (v3.0).



(b) Full SPECint CPU2006.

Fig. 4: Performance slowdown using LockStep, FlexStep and Nzdc.

Results. As shown in Fig. 4, FlexStep incurs a performance slowdown (geomean) of 1.07% when running Parsec and 1.24% when running SPEC, which is attributed to the extraction of RCPs and backpressure from Data Buffering resulting from cores undergoing different kernel mode switches and instruction executions. In contrast, the slowdown of Nzdc is almost 57.7% in Parsec and 91.5% in SPEC, roughly 1.56x and 1.89x slower than FlexStep. Nzdc experiences a significant performance degradation due to the redundant checking instructions. While LockStep incurs no performance drop, it requires an additional equivalent area. FlexStep achieves minimal slowdown with reasonable area overheads (Sec. VI-E), highlighting its advantages in both performance and resource efficiency.

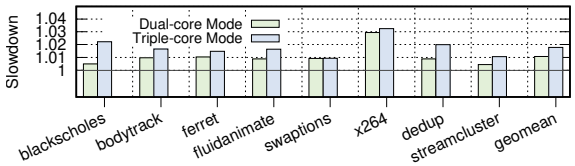


Fig. 5: Performance slowdown in dual-core and triple-core mode.

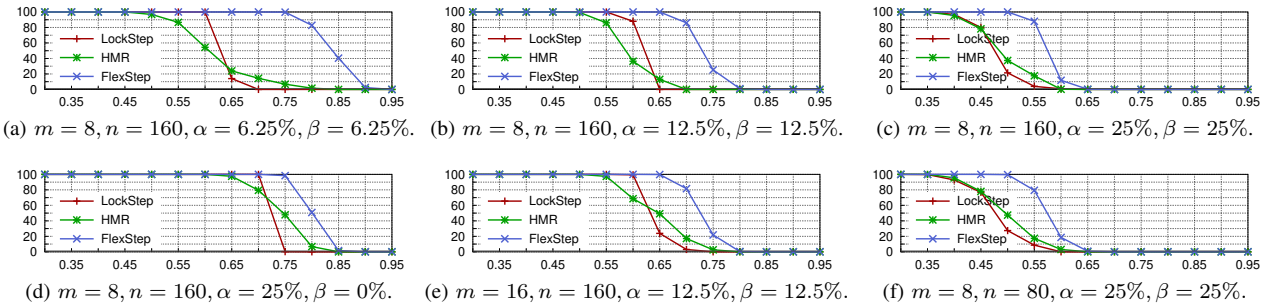
We further compare the slowdown of Parsec under different verification modes in Fig. 5. Results indicate that the slowdown (geomean) in triple-core mode increases slightly to 1.77%. This modest increase is attributed to having more checker cores, which exacerbates the execution inconsistency between cores, leading to more frequent backpressure on the main core. Overall, the performance overhead in both modes remain minimal, demonstrating that FlexStep's support for different verification modes is highly performance-efficient.

B. Percentage of Schedulable Task Sets

Experiment setup. To assess FlexStep's schedulability improvements, we conducted numerical simulations using randomly generated task sets across various configurations. Task sets were generated following the UUnifast algorithm [39]. We compared three error detection schemes: LockStep, HMR, and FlexStep, under partitioned EDF. For LockStep, tasks with different reliability levels were partitioned into separate queues and ordered by descending utilisation. Verification tasks were allocated first, minimising the number of checker cores by only allocating a new group of main and checker cores when the current group was full. Non-verification tasks were allocated last across all cores. For HMR, verification tasks were also allocated first. Then non-verification tasks were assigned, first filling cores without verification tasks and then cores with lowest utilisation. The partitioning of FlexStep is described in Sec. V.

Results. Fig. 6 shows that FlexStep consistently outperforms LockStep and HMR, especially at higher utilisation levels. As utilisation increases, FlexStep and HMR exhibit a more gradual performance decline, whereas LockStep experiences a sharp drop. This difference arises from LockStep's rigid verification approach. HMR encounters few deadline misses even under moderate utilisation, mainly due to the fact that non-verification tasks cannot preempt verification tasks allocated on the same core. FlexStep's flexibility allows for more efficient resource allocation, mitigating these issues.

Comparing Figs. 6(a), 6(b), and 6(c), it is evident that FlexStep achieves greater performance improvements when there are fewer verification tasks, as more non-verification tasks are required to share cores with verification tasks. As FlexStep allows flexible switching between verification and non-verification tasks, without requiring binding between main and checker cores, it enhances resource utilisation and improves system schedulability. The improvement persists with an increased number of cores or higher utilisations of individual tasks, as shown in Figs. 6(e) and 6(f). Comparing Figs. 6(b) and 6(d), the addition of triple-check tasks significantly increases resource demand, leading to varying degrees of performance degradation across all three methods. However, LockStep and HMR exhibit a sharper decline in performance compared to FlexStep, as their core binding between main and checker cores limits flexibility in scheduling. In contrast, FlexStep enables more flexible verification and scheduling, optimising resource utilisation to meet deadlines, even in more demanding scenarios.



(a) $m = 8, n = 160, \alpha = 6.25\%, \beta = 6.25\%$. (b) $m = 8, n = 160, \alpha = 12.5\%, \beta = 12.5\%$. (c) $m = 8, n = 160, \alpha = 25\%, \beta = 25\%$.

(d) $m = 8, n = 160, \alpha = 25\%, \beta = 0\%$. (e) $m = 16, n = 160, \alpha = 12.5\%, \beta = 12.5\%$. (f) $m = 8, n = 80, \alpha = 25\%, \beta = 25\%$.

Fig. 6: Percentage of schedulable task sets (y-axis) under LockStep, HMR, and FlexStep with increasing task set utilisations (x-axis) and varying system configurations: m (number of cores), n (number of tasks), α (percentage of double-check tasks), and β (percentage of triple-check tasks).

C. Error Detection Latency

Experiment setup. To evaluate detection latency, we injected errors in the forwarded data from the main core, e.g., memory access data of MAL and architectural register data of ASS, simulating the hardware faults without disrupting the main core's normal execution. For each workload, 5,000 - 10,000 faults were randomly generated, resulting in over 100,000 sample points in total to ensure the validity of results.

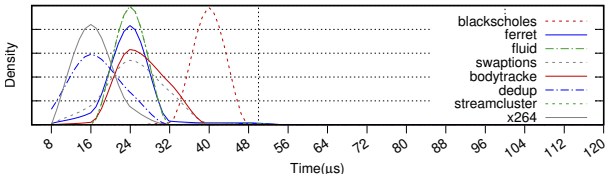
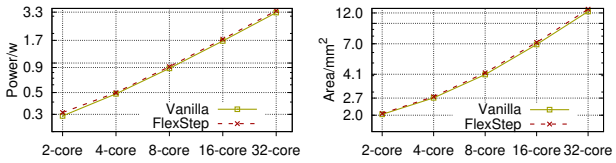


Fig. 7: Probability distribution of error detection latency of Parsec.

Results. Fig. 7 shows the density distribution of detection latency across different workloads. For most workloads, the majority of detection latencies are concentrated around 20 μs, while the maximum latency seen in blackscholes is 2 to 3 times higher, reaching up to 50 μs. Overall, FlexStep maintains an actual error detection latency of no more than 50μs in most cases, which is sufficient to cover over 99.9% of hardware faults and guarantee the system reliability.

D. Scalability

Experiment setup. To evaluate scalability, we increased the number of cores for both FlexStep and Vanilla (based on the original unmodified microarchitecture). We assessed their area and average power consumption using Design Compiler (v2019.12) for RTL synthesis and PrimeTime PX (v2019.12) for post-simulation analysis.



(a) Average Power. (b) Area.

Fig. 8: The average power and area overheads on the SoC with different core numbers for Vanilla and FlexStep, including L1$ and L2$.

Results. Fig. 8 presents comparison of average power (Fig. 8(a)) and area (Fig. 8(b)) for Vanilla and FlexStep across varying core counts, indicating that the increase in average power and area for FlexStep relative to Vanilla remains nearly linear, rather than exponential, as the SoC scales from dual-core to 32-core. This demonstrates that FlexStep offers strong scalability in multi/many-core systems.

E. Hardware Overheads

Experiment setup. We used the same setup as in scalability to evaluate the hardware overheads of FlexStep compared to Vanilla.

TABLE III: Average power & area of Vanilla and FlexStep (4 cores).

	Vanilla	FlexStep		
Core	Rocket	Rocket	**Overhead**	×
Tech. (nm)	28	28		×
Power (w)	0.485	0.499		2.89%
Area (mm^2)	2.71	2.77		2.21%

Results. With the reuse of homogeneous checker cores, FlexStep introduces minimal hardware overhead. Compared to Vanilla, FlexStep incurs the storage overhead per core of only 1614 bytes: 8 bytes for CPC, 518 bytes for ASS, and 1088 bytes for DBC. For a 4-core SoC shown in Tab. III, FlexStep occupies 2.77 mm^2 of area and consumes 0.499w of power, representing just 2.21% and 2.89% overhead, respectively, compared to Vanilla. This validates that the hardware overhead for implementing FlexStep is well within reasonable limits.

VII. Conclusion

In this work, we developed FlexStep, a homogeneous-core error-detection framework. Using a hardware-software co-design approach, we implemented FlexStep by adding lightweight modifications to existing cores and integrating simple OS-level scheduling codes. Evaluated on an FPGA prototype, FlexStep demonstrates microsecond-level error detection capabilities with low performance and hardware overheads, high scalability, and dynamic verification mode switching, making it an efficient solution for practical applications. In future works, FlexStep enables flexible task scheduling and unlocks significant potential for developing new scheduling algorithms to enhance system efficiency, improve real-time responsiveness, and adapt to dynamic reliability requirements in safety-critical applications.

VIII. Acknowledgment

We would like to thank the anonymous reviewers for the helpful feedback. This work is supported by National Key Research and Development Program (Grant No.2024YFB4405600), U.S. National Science Foundation (Grant No.CNS-2340171), National Natural Science Foundation of China (Grant Nos. 62472086, 92264203 and 62204036), the Basic Research Program of Jiangsu (Grants No. BK20243042), Key Research and Development Program of Jiangsu Province(Grant No.BE2023020-1), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

References

[1] V. Kumar, L. Singh, and A. K. Tripathi, "Reliability analysis of safety-critical and control systems: a state-of-the-art review," *IET Software*, vol. 12, no. 1, pp. 1–18, 2018.

[2] A. Maurya and D. Kumar, "Reliability of safety-critical systems: A state-of-the-art review," *Quality and Reliability Engineering International*, vol. 36, no. 7, pp. 2547–2568, 2020.

[3] P. Singh and L. K. Singh, "Reliability and safety engineering for safety critical systems: An interview study with industry practitioners," *IEEE Transactions on Reliability*, vol. 70, no. 2, pp. 643–653, 2021.

[4] V. Kumar, K. C. Mishra, P. Singh, A. N. Hati, M. R. Mamdikar, L. K. Singh, and R. R. Parida, "Reliability analysis and safety model checking of safety-critical and control systems: A case study of npp control system," *Annals of Nuclear Energy*, vol. 166, p. 108812, 2022.

[5] W. Ali and H. Yun, "Rt-gang: Real-time gang scheduling framework for safety-critical systems," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 143–155.

[6] V. P. Kumar and A. S. Pillai, "Dynamic scheduling algorithm for a utomotive safety critical systems," in *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE, 2020, pp. 815–820.

[7] D. Calvaresi, M. Marinoni, L. Lustrissimini, K. Appoggetti, P. Sernani, A. F. Dragoni, M. Schumacher, and G. Buttazzo, "Local scheduling in multi-agent systems: getting ready for safety-critical scenarios," in *Multi-Agent Systems and Agreement Technologies: 15th European Conference, EUMAS 2017, and 5th International Conference, AT 2017, Evry, France, December 14-15, 2017, Revised Selected Papers 15*. Springer, 2018, pp. 96–111.

[8] A. Hanafi, M. Karim, and A. E. Hammami, "Dual-lockstep microblaze-based embedded system for error detection and recovery with reconfiguration technique," in *2015 Third World Conference on Complex Systems (WCCS)*, 2015, pp. 1–6.

[9] X. Iturbe, B. Venu, E. Ozer, and S. Das, "A triple core lock-step (tcls) arm® cortex®-r5 processor for safety-critical and ultra-reliable applications," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, 2016, pp. 246–249.

[10] X. Iturbe, B. Venu, E. Ozer, J.-L. Poupat, G. Gimenez, and H.-U. Zurek, "The arm triple core lock-step (tcls) processor," *ACM Transactions on Computer Systems (TOCS)*, vol. 36, no. 3, pp. 1–30, 2019.

[11] X. Iturbe, B. Venu, J. Jagst, E. Ozer, P. Harrod, C. Turner, and J. Penton, "Addressing functional safety challenges in autonomous vehicles with the arm tcl s architecture," *IEEE Design & Test*, vol. 35, no. 3, pp. 7–14, 2018.

[12] S. Hukerikar, K. Teranishi, P. C. Diniz, and R. F. Lucas, "Redthreads: An interface for application-level fault detection/correction through adaptive redundant multithreading," *International Journal of Parallel Programming*, vol. 46, pp. 225–251, 2018.

[13] P. Bernardi, R. Cantoro, S. De Luca, E. Sánchez, and A. Sansonetti, "Development flow for on-line core self-test of automotive microcontrollers," *IEEE Transactions on Computers*, vol. 65, no. 3, pp. 744–754, 2015.

[14] D. S. Khudia and S. Mahlke, "Harnessing soft computations for low-budget fault tolerance," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014, pp. 319–330.

[15] A. Thomas and K. Pattabiraman, "Error detector placement for soft computation," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2013, pp. 1–12.

[16] G. A. Ismael, A. A. Salih, A. AL-Zebari, N. Omar, K. J. Merceedi, A. J. Ahmed, N. O. Salim, S. S. Hasan, S. F. Kak, I. M. Ibrahim *et al.*, "Scheduling algorithms implementation for real time operating systems: A review," *Asian Journal of Research in Computer Science*, vol. 11, no. 4, pp. 35–51, 2021.

[17] P. Agrawal, A. K. Gupta, and P. Mathur, "Cpu scheduling in operating system: a review," in *Proceedings of the Second International Conference on Information Management and Machine Intelligence: ICIMMI 2020*. Springer, 2021, pp. 279–289.

[18] M. Hamayun and H. Khurshid, "An optimized shortest job first scheduling algorithm for cpu scheduling," *J. Appl. Environ. Biol. Sci*, vol. 5, no. 12, pp. 42–46, 2015.

[19] F. Kluge, M. Gerdes, and T. Ungerer, "An operating system for safety-critical applications on manycore processors," in *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, 2014, pp. 238–245.

[20] C. El Salloum, M. Elshuber, O. Höftberger, H. Isakovic, and A. Wasicek, "The across mpsoc–a new generation of multi-core processors designed for safety–critical embedded systems," *Microprocessors and Microsystems*, vol. 37, no. 8, pp. 1020–1032, 2013.

[21] J. P. Cerrolaza, R. Obermaisser, J. Abella, F. J. Cazorla, K. Grüttner, I. Agirre, H. Ahmadian, and I. Allende, "Multi-core devices for safety-critical systems: A survey," *ACM Computing Surveys (CSUR)*, vol. 53, no. 4, pp. 1–38, 2020.

[22] J. Perez-Cerrolaza, J. Abella, L. Kosmidis, A. J. Calderon, F. Cazorla, and J. L. Flores, "Gpu devices for safety-critical systems: A survey," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[23] G. Durrieu, M. Faugère, S. Girbal, D. G. Pérez, C. Pagetti, and W. Puffitsch, "Predictable flight management system implementation on a multicore processor," in *Embedded Real Time Software (ERTS'14)*, 2014.

[24] M. Rogenmoser, Y. Tortorella, D. Rossi, F. Conti, and L. Benini, "Hybrid modular redundancy: Exploring modular redundancy approaches in risc-v multi-core computing clusters for reliable processing in space," *ACM Transactions on Cyber-Physical Systems*, 2023.

[25] F. Kempf, C. Kühbacher, C. Mellwig, S. Altmeyer, T. Ungerer, and J. Becker, "A holistic hardware-software approach for fault-aware embedded systems," in *2022 25th Euromicro Conference on Digital System Design (DSD)*. IEEE, 2022, pp. 704–711.

[26] F. Kempf, T. Hartmann, S. Baehr, and J. Becker, "An adaptive lockstep architecture for mixed-criticality systems," in *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 2021, pp. 7–12.

[27] F. Bas, S. Alcaide, R. Lorenzo, G. Cabo, G. Gil, O. Sala, F. Mazzocchetti, D. Trilla, and J. Abella, "Safede: a flexible diversity enforcement hardware module for light-lockstepping," in *2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS)*. IEEE, 2021, pp. 1–7.

[28] M. Barbirotta, F. Menichelli, A. Cheikh, A. Mastrandrea, M. Angioli, and M. Olivieri, "Dynamic triple modular redundancy in interleaved hardware threads: An alternative solution to lockstep multi-cores for fault-tolerant systems," *IEEE Access*, 2024.

[29] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[30] S. Ainsworth and T. M. Jones, "Parallel error detection using heterogeneous cores," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 338–349.

[31] S. Ainsworth, L. Zoubritzky, A. Mycroft, and T. M. Jones, "Paradox: Eliminating voltage margins via heterogeneous fault tolerance," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 520–532.

[32] S. Ainsworth and T. M. Jones, "Paramedic: Heterogeneous parallel error correction," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 201–213.

[33] A. Amid, D. Biancolin, A. Gonzalez *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, pp. 10–21, 2020.

[34] "28nm PDKs," https://www.tsmc.com/english/dedicatedFoundry/technology/logic/l_28nm.

[35] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra *et al.*, "Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 29–42.

[36] J. L. Henning, "Spec cpu2006 benchmark descriptions," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.

[37] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, 2008, pp. 72–81.

[38] M. Didehban and A. Shrivastava, "nzdc: A compiler technique for near zero silent data corruption," in *Proceedings of the 53rd Annual Design Automation Conference*, 2016, pp. 1–6.

[39] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Systems (Real-Time Syst.)*, vol. 30, no. 1-2, pp. 129–154, 2005.

Multi-Agent Yield Analysis For Circuit Design

Haiyan Qin*, Jing Kou*, Liang Zhang*, Wang Kang*§, Wei W. Xing†§
*National Key Laboratory of Spintronics, Hangzhou International Innovation Institute;
School of Integrated Circuit Science and Engineering, Beihang University, China
†School of Mathematical and Physical Sciences, University of Sheffield, Sheffield, United Kingdom
*{haiyanq, koujing, zhangliang24, wang.kang}@buaa.edu.cn, †w.xing@sheffield.ac.uk

***Abstract*—Semiconductor yield estimation presents a critical challenge in modern manufacturing, directly impacting production costs and market competitiveness. Traditional estimation methods, particularly Monte Carlo simulation, while reliable, become computationally prohibitive for complex modern circuits. Contemporary approaches, including importance sampling and machine learning techniques, face fundamental limitations in consistency across circuit topologies and practical validation. This work introduces YieldAgent, a novel Large Language Model (LLM)-powered framework that revolutionizes yield estimation through dynamic integration of multiple analytical strategies. YieldAgent employs a three-layer agent architecture to analyze circuit characteristics and historical data, optimizing estimation methods while balancing computational efficiency and precision. The framework incorporates Retrieval-Augmented Generation for domain knowledge integration and Tree-structured Parzen Estimators for dynamic hyperparameter optimization. Experimental validation across 12nm and 40nm technology nodes demonstrates that YieldAgent reduces computational overhead by up to 2.9× while maintaining or exceeding state-of-the-art accuracy. The system's ability to adapt across different circuit topologies and technology nodes establishes a new paradigm for scalable, intelligent yield estimation in electronic design automation.**

***Index Terms*—Yield analysis, Multi-agent system, LLM**

I. Introduction

Yield estimation serves as a cornerstone of semiconductor manufacturing, fundamentally impacting production optimization, cost management, and market competitiveness. The increasing complexity of modern integrated circuits, particularly in designs requiring high fault tolerance such as SRAM, has created unprecedented challenges in yield prediction. As process nodes continue to shrink and parameter spaces expand, traditional yield estimation methods face significant limitations in both accuracy and computational efficiency. The semiconductor industry's tight margins and competitive pressures make these limitations particularly acute, as even small improvements in yield estimation can translate to substantial financial benefits.

Traditional yield estimation methods, such as Monte Carlo (MC) simulation and Importance Sampling (IS), face growing challenges in modern semiconductor manufacturing. These methods struggle to balance efficiency and accuracy, especially with expanding parameter spaces and multiple failure regions [1]. As circuit complexity increases, their computational costs become prohibitive, demanding new solutions that reconcile accuracy with resource constraints.

§ Denotes Corresponding Authors.

This work was supported by Key R&D Program of Shandong Province, China2022CXGC020208, Beijing Municipal Natural Science Foundation (Z230004), the Beijing Natural Science Foundation (L223004), Natural Science Foundation of China (62274008) and Beijing MSTC Program (Z231100007423019).

Recent AI advancements have introduced data-driven surrogate models for yield estimation. Techniques like Bayesian Yield Analysis (BYA) and Absolute Shrinkage Deep Kernel Learning (ASDK) show promise in modeling performance functions and predicting yield [2], [3]. However, they face challenges such as extensive training data requirements, optimization instabilities, and limited adaptability across circuit topologies. While IS-based methods have improved through mixture importance sampling and adaptive algorithms [4], [5], they remain computationally intensive and lack dynamic adaptability.

The rise of Large Language Models (LLMs) offers a promising solution, leveraging their reasoning capabilities and few-shot learning [6]. LLMs' ability to generalize from limited data is particularly valuable in yield estimation, where extensive data collection is impractical. Their adaptability across circuit topologies aligns with industry needs for scalable solutions. Recent LLM applications in electronic design automation, from HDL code generation [7]–[9] to design optimization [10] and debugging [11], [12], show potential, though their use in yield estimation remains underexplored.

This work introduces YieldAgent, an innovative LLM-powered framework that transforms yield estimation through intelligent strategy selection and resource allocation. YieldAgent implements a sophisticated three-layer agent architecture that leverages LLMs' reasoning capabilities for dynamic method integration and optimization. The framework offers three fundamental capabilities that distinguish it from existing approaches: dynamic budget-aware resource allocation for optimizing computational efficiency, precision-guided estimation that adapts strategies based on accuracy requirements, and topology-aware method selection that customizes approaches for specific circuit characteristics. The primary contributions of this work encompass three key innovations:

- **LLM-Powered Multi-Agent Architecture:** YieldAgent introduces a novel three-layer architecture (Strategy, Coordination, and Execution) that leverages LLMs for intelligent yield estimation. Through the integration of Retrieval-Augmented Generation (RAG) for domain knowledge and Chain-of-Thought (CoT) reasoning for strategy selection, the system achieves dynamic adaptation across diverse circuit topologies. This architecture enables specialized agents to collaborate effectively, from high-level planning to tactical execution.
- **Dynamic Knowledge Integration and Optimization:** We present a comprehensive framework combining historical performance data with live analysis. The system employs Tree-structured Parzen Estimators (TPE) for hyperparameter tuning and implements a sliding

window convergence mechanism for precision control. This integration enables automated strategy selection and resource allocation based on circuit characteristics and performance requirements.

- **Comprehensive Cross-Technology Validation:** Through extensive experimentation across 12nm and 40nm nodes, we demonstrate YieldAgent's superior adaptability and efficiency. The framework achieves a 2.9× reduction in computational overhead while maintaining accuracy, validated through three key capabilities: budget-aware resource optimization, precision-guided estimation, and topology-aware method selection. This validation establishes YieldAgent's effectiveness across different technology nodes and circuit complexities.

II. Background

A. Problem Definition

The failure probability P_{fail} of a circuit is the likelihood that a performance metric Y falls into the failure region S:

$$P_{\text{fail}} = P(Y \in S) = \int I(x) f(x)\, dx \tag{1}$$

where $f(x)$ is the multivariate probability density function of the circuit's parameters x, and $I(x)$ is an indicator function:

$$I(x) = \begin{cases} 1, & \text{if } x \in S, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

This integral is generally intractable because $I(x)$ is not analytically available, making direct evaluation of P_{fail} computationally infeasible.

B. Sampling-Based Yield Analysis

Monte Carlo Simulation (MC). Monte Carlo simulation estimates P_{fail} by randomly sampling N points $\{X_i\}_{i=1}^{N}$ from $f(x)$. The estimated failure probability is:

$$\hat{P}_{\text{fail}} = \frac{1}{N} \sum_{i=1}^{N} I(X_i) \tag{3}$$

As N increases, $\hat{P}_{\text{fail}}$ converges to P_{fail}. However, MC is inefficient for rare events, requiring a large number of samples when P_{fail} is very small [1].

Importance Sampling (IS). IS improves efficiency by modifying the sampling distribution to oversample the failure region S. Using a biased distribution $g(x)$ instead of $f(x)$, the failure probability becomes:

$$P_{\text{fail}} = \int I(x) \frac{f(x)}{g(x)} g(x)\, dx \tag{4}$$

The IS estimator is $\hat{P}_{\text{IS}} = \frac{1}{N} \sum_{i=1}^{N} w(X_i) I(X_i)$, where $w(X_i) = f(X_i)/g(X_i)$ ensures correct weighting of samples from $g(x)$. The optimal $g_{\text{opt}}(x) = I(x)f(x)/P_{\text{fail}}$ is impractical since P_{fail} and $I(x)$ are unknown, so practical methods approximate $g_{\text{opt}}(x)$ by shifting $f(x)$ towards S or using optimization-based approaches [4], [5].

C. AI-Enhanced Yield Analysis

AI introduces data-driven surrogate models to approximate complex performance functions in yield estimation. Examples include LRTA [13], which uses PCE; BYA [2], employing GP with entropy reduction in active learning; and ASDK [3], replacing GP with a deep kernel method and feature selection. Deep learning techniques like Normalizing Flows [14] model failure distributions.

Enhancements to IS methods focus on designing proposal distributions that closely approximate the failure distribution. MNIS [15] centers a normal distribution on the failure sample closest to the origin. GIS [16] introduces an Optimal Mean Shift Vector (OMSV) via gradient descent; FSIS [17] uses sensitivity analysis for efficiency; AIS [18] iteratively updates OMSV with new data. To address multi-region failures, HSCS [5] and ACS [19] employ cluster-based models and iteratively update the proposal distribution.

Despite these advancements, significant challenges remain. Traditional and AI-based methods often lack flexibility across diverse circuit topologies and face high computational costs due to extensive simulations or large training datasets. They typically use fixed strategies incapable of dynamically integrating multiple estimation methods or balancing computational load with precision requirements, limiting practical adoption [1], [5]. These gaps necessitate a new paradigm in yield estimation.

D. LLMs in Electronic Design Automation

LLMs have demonstrated promising capabilities across various EDA tasks. In hardware design, LLMs have enabled automated HDL code generation [7]–[9], design optimization [10], and debugging assistance [11], [12]. These applications leverage various LLM capabilities, including fine-tuning, prompt engineering, and domain adaptation techniques.

A key advantage of LLMs is their implicit Bayesian inference via in-context learning (ICL), allowing them to leverage prior knowledge and generalize from limited examples [6], [20]–[23]. This is particularly beneficial for yield estimation, where data scarcity due to large parameter spaces and high simulation costs poses significant challenges. Additionally, LLMs' adaptability meets industrial needs for diverse circuit topologies and technology nodes.

However, applying LLMs to yield estimation faces unique challenges. Existing LLM frameworks lack specialized training data and struggle to integrate domain-specific knowledge, potentially leading to unreliable results in complex tasks. While LLMs excel at code generation, they often fail to produce functional methods for specialized applications. These limitations have historically impeded their effective use in yield estimation, especially where adaptability and computational efficiency are critical.

III. Methodology

A. System Overview and Design Principles

The fundamental challenge in yield estimation lies in balancing computational efficiency with estimation accuracy

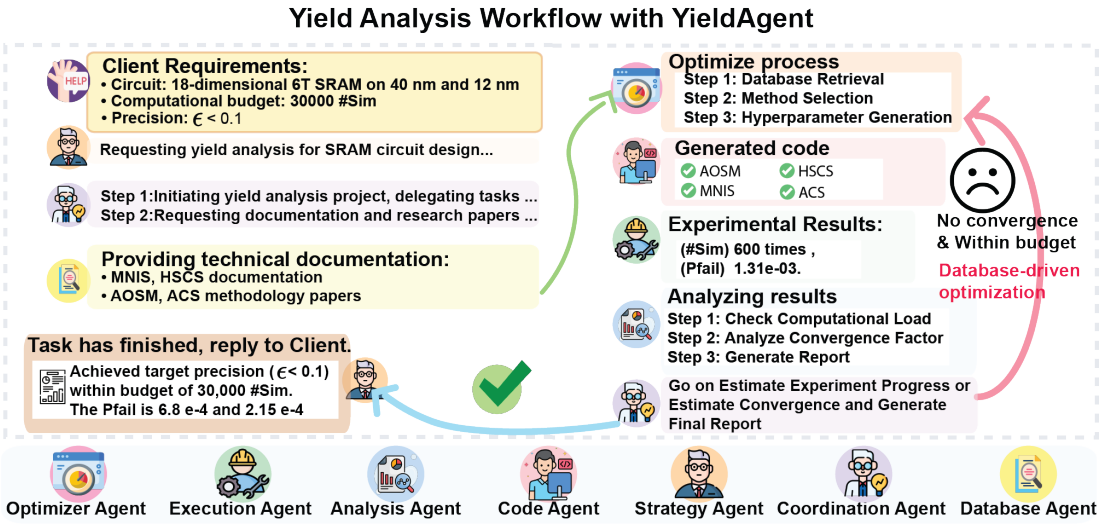


Fig. 1. Yield Analysis Workflow of YieldAgent

across diverse circuit topologies and technology nodes. YieldAgent addresses this challenge through three key design principles: adaptive resource allocation, precision-guided estimation, and topology-aware method selection. Fig. 1 illustrates the system architecture designed to implement these principles through a multi-agent framework.

The selection of a multi-agent architecture stems from the inherent complexity of yield estimation, where different stages of the process require distinct types of expertise and decision-making capabilities. Traditional approaches often treat yield estimation as a monolithic process, leading to inefficiencies in resource allocation and limited adaptability. By decomposing the process into specialized agents, YieldAgent enables dynamic adaptation to varying circuit characteristics while maintaining systematic coordination through LLM-based reasoning.

B. Multi-Agent Architecture

As illustrated in Fig. 2, YieldAgent implements a hierarchical three-layer architecture that separates strategic planning from tactical execution:

Strategic Layer: The Strategy Agent operates at the highest level, determining estimation goals and resource constraints. This agent leverages LLM reasoning capabilities to analyze circuit characteristics and historical performance data, developing high-level estimation strategies that align with specified precision requirements and computational budgets.

Coordination Layer: Two specialized agents operate at this level: the Coordination Agent manages task distribution and method selection, while the Database Agent handles knowledge integration and retrieval. This layer ensures coherent system-wide behavior by mediating between high-level strategies and concrete execution tasks.

Execution Layer: Four specialized agents handle specific tasks: (1) Code Generation Agent: Produces executable code for selected estimation methods, (2) Execution Agent: Manages simulation runs and resource utilization, (3) Analysis Agent: Evaluates results and computes convergence metrics, (4) Optimizer Agent: Tunes hyperparameters and adapts strategies in real-time.

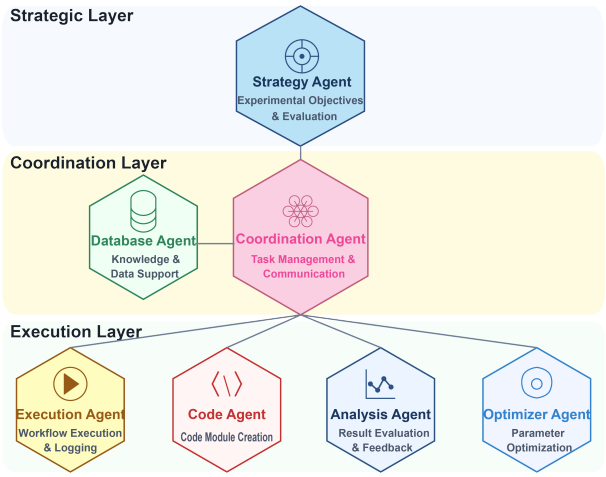


Fig. 2. YieldAgent Three-Layer Architecture Illustration.

C. Knowledge Integration and Method Selection

YieldAgent leverages a comprehensive knowledge base using Retrieval-Augmented Generation (RAG) to enhance its yield estimation capabilities. The knowledge base incorporates validated methodologies including MNIS [15], HSCS [5], AOSM [1], and ACS [19]. This integration addresses a critical limitation of LLMs: while they possess broad knowledge, they often lack specialized domain expertise required for technical tasks like yield estimation.

The Database Agent maintains performance profiles across different technology nodes and circuit topologies, enabling data-driven decision making. Through its implicit Bayesian inference capabilities [6], YieldAgent analyzes this historical data to adaptively select optimal strategies, functioning similarly to posterior updates in Bayesian methods. Validated across multiple technology nodes (e.g., Fig. 3), the database provides performance profiles that support informed, data-driven decisions.

The Coordination Agent combines database insights with real-time performance metrics to dynamically adjust estima-

tion strategies. This adaptive framework ensures robust and reliable yield estimation by: (1) Analyzing circuit characteristics to identify suitable estimation methods (2) Monitoring convergence rates to adjust computational resource allocation (3) Adapting strategy combinations based on real-time performance feedback.

D. Dynamic Hyperparameter Optimization

Empirical analysis reveals that direct transfer of hyperparameters across technology nodes often leads to suboptimal performance. As shown in Fig. 4, methods optimized for 40nm technology exhibit significant performance degradation when applied to 12nm circuits without adjustment. The failure probability (P_{fail}) curves demonstrate substantial gaps between 40nm-optimized configurations and reference results on 12nm nodes.

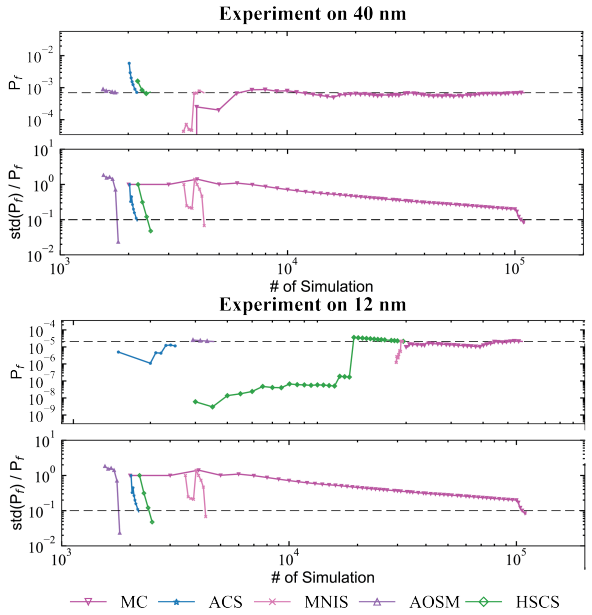


Fig. 3. Performance Comparison of Yield Estimation Methods Across 40nm and 12nm Nodes

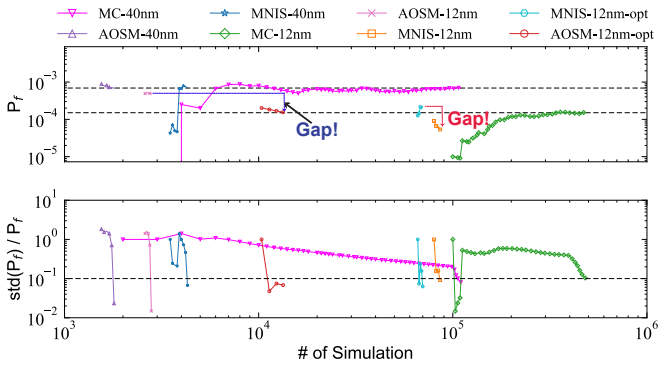


Fig. 4. Yield Analysis on 40nm and 12nm Nodes for MNIS and AOSM

YieldAgent addresses this challenge through a Tree-structured Parzen Estimator (TPE) that dynamically optimizes hyperparameters based on real-time performance metrics. The TPE algorithm models the relationship between hyperparameters and estimation performance using two probability density functions:

$$l(x) = p(x|y \leq y^*) \text{ and } g(x) = p(x|y > y^*) \tag{5}$$

where y^* denotes the performance threshold. New hyperparameters are selected by maximizing the expected improvement ratio:

$$x_{t+1} = \arg\max_x (l(x)/g(x)) \tag{6}$$

E. Statistical Convergence Analysis

YieldAgent employs a robust convergence analysis framework designed for dynamic, multi-model scenarios. The framework uses a sliding window statistical approach to assess the stability of the entire estimation process. For a window of size k, we calculate **Relative Variation:**

$$RV_k = \frac{\sigma_k}{\bar{x}} \tag{7}$$

where σ_k is the standard deviation and $\bar{x}$ is the mean of results within the window. Convergence is achieved when $RV_k < \epsilon$, where ϵ is the predefined precision threshold. This approach offers several advantages such as, (1) Robustness to local fluctuations, (2) Adaptive sensitivity through relative variation, (3) Computational efficiency via incremental updates. The sliding window approach provides a reliable convergence assessment than traditional metrics like Figure of Merit (FoM), particularly in multiple models and iterative refinement scenarios.

F. Integrated Workflow

YieldAgent leverages its modular multi-agent architecture to address yield estimation challenges by dynamically selecting methods, optimizing hyperparameters, and efficiently allocating computational resources. This is exemplified in the estimation of yield for a 6T-SRAM circuit at both 40nm and 12nm technology nodes, constrained by a computational budget of 30,000 simulations and a target precision of $\epsilon < 0.1$, as shown in Fig. 1.

The workflow begins with the user specifying circuit design parameters and requirements. The **Strategy Agent** analyzes this input to develop an initial estimation plan. This analysis encompasses circuit topology and complexity, technology node characteristics, precision requirements, and computational constraints. These factors form the basis for subsequent optimization decisions.

The **Database Agent** then retrieves relevant historical performance data for various estimation methods (MNIS, HSCS, AOSM, and ACS) across similar scenarios. Based on this information, the **Coordination Agent** selects optimal method combinations for each technology node through a systematic process. This process involves initial method selection based on historical performance, followed by resource allocation planning across selected methods, and concludes with the definition of intermediate evaluation points for strategy adjustment.

During the simulation phase, the **Code Generation Agent** produces executable code tailored to the selected methods.

This code incorporates method-specific implementations, initialization parameters, and convergence checks. The **Execution Agent** then manages simulation runs using initial hyperparameters optimized by the **Optimizer Agent**. The execution process follows a formal optimization structure:

1) For each method M_i in selected set:
   a) Initialize: $\theta_i = \text{TPE}_{\text{init}}(M_i, \text{historical_data})$
   b) Execute: $\text{results}_i = M_i(\theta_i, \text{circuit_params})$
   c) Update: $\theta_i = \text{TPE}_{\text{update}}(\text{results}_i, \theta_i)$

The **Analysis Agent** periodically evaluates intermediate results using the sliding window statistical approach to calculate relative variation (RV_k) and assess convergence:

$$RV_k = \frac{\sigma_k}{\bar{x}} < \epsilon \tag{8}$$

When convergence criteria are not met, the system initiates an iterative refinement process. The Optimizer Agent adjusts hyperparameters using TPE, while the Coordination Agent modifies method weights or combinations. Simultaneously, the Strategy Agent may reallocate computational resources to optimize performance.

The optimization process employs a dynamic feedback mechanism where hyperparameter configurations are adjusted to maximize expected improvements in precision:

$$\theta_{t+1} = \arg\max_{\theta} \mathbb{E}[\text{Improvement}(\theta|\mathcal{D}_t)] \tag{9}$$

where $\mathcal{D}_t$ represents the historical performance data up to iteration t. Methods exhibiting suboptimal performance undergo systematic enhancement through additional hyperparameter optimization, replacement with alternative methods from the database, or integration with complementary approaches. This adaptive strategy ensures continuous improvement of estimation accuracy while maintaining computational efficiency.

The integrated workflow demonstrates YieldAgent's capability to dynamically optimize estimation strategies while balancing computational efficiency with precision requirements. The system adapts seamlessly to diverse process nodes and circuit characteristics through systematic coordination of its multi-agent components. This comprehensive approach establishes a robust framework for yield estimation across varying technological contexts and precision demands.

The effectiveness of this workflow is validated through comprehensive experiments across different technology nodes and precision requirements, as detailed in Section IV. These experiments demonstrate quantitative improvements in both computational efficiency and estimation accuracy, establishing YieldAgent's practical utility in modern semiconductor manufacturing contexts.

IV. Experiments

The experimental validation of YieldAgent focuses on three fundamental capabilities: computational resource optimization, precision-guided estimation, and technology node adaptation. A 6T-SRAM circuit serves as the primary benchmark, chosen for its sensitivity to process variations and industrial relevance. The evaluation spans multiple technology nodes and precision requirements to demonstrate versatility and effectiveness.

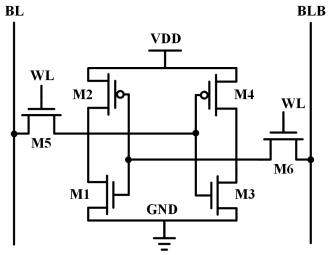


Fig. 5. Structure of the 6T-SRAM bit cell used in experiments.

A. Experimental Setup

YieldAgent's implementation utilizes Microsoft's AutoGen [24] multi-agent framework integrated with OpenAI's GPT-4o-mini model. The system operates on an AMD EPYC 7601 CPU platform with 62GB RAM, employing HSPICE for circuit simulation. All convergence evaluations utilize a sliding window size of $k = 5$ to ensure statistical stability.

The 6T-SRAM bit cell, illustrated in Fig. 5, serves as an ideal test vehicle for yield estimation due to its sensitivity to process variations and widespread deployment in semiconductor research. The experiments span TSMC 12nm and SMIC 40nm technology nodes to assess YieldAgent's adaptability across different manufacturing processes.

Process variations are modeled through three critical transistor parameters: oxide thickness (t_{ox}), threshold voltage (V_{th}), and carrier mobility (μ). These parameters significantly influence transistor performance metrics, including gate capacitance, switching characteristics, and drive current. SRAM read and write delay serve as key performance metrics for evaluating the impact of these variations.

B. Budget-Aware Capability Evaluation

The evaluation of YieldAgent's resource optimization capabilities employs three distinct computational budgets: Low (3,000 simulations), Medium (6,000 simulations), and High (10,000 simulations). The analysis compares YieldAgent against traditional Monte Carlo simulation and four state-of-the-art methods: MNIS [15], AOSM [1], ACS [19], and HSCS [5]. A reference failure rate of 6.91×10^{-4} is established through Monte Carlo simulation using 110,000 samples.

Table I presents comprehensive comparative results across budget levels. Under the Low budget constraint, YieldAgent achieves 4.1% relative error using 2,890 simulations through dynamic combination of MNIS, AOSM, and ACS methods. This performance significantly surpasses the best single-method approach (ACS: 8.6% error, 2,980 simulations).

In the Medium budget scenario, YieldAgent demonstrates superior method integration capabilities by combining MNIS with iterative AOSM refinements. This approach achieves 2.96% relative error using 5,550 simulations, while fixed methods exhibit errors ranging from 3.4% to 6.7%. The High budget level further showcases YieldAgent's strategic prowess, where its combination of MNIS and AOSM iterations yields 1.8% relative error with 9,200 simulations, compared to AOSM's 3.6% error using 9,950 simulations.

TABLE I
YIELD ANALYSIS WITH VARYING BUDGETS

Budget	Method	# Sim	Fail Rate	Rel. Err.
–	MC	110,000	6.91×10^{-4}	–
Low	MNIS	2,500	6.16×10^{-4}	10.8%
	AOSM	2,920	6.04×10^{-4}	12.5%
	ACS	2,980	6.32×10^{-4}	8.6%
	HSCS	2,050	5.81×10^{-4}	15.8%
	YieldAgent	**2,890**	$\mathbf{6.62 \times 10^{-4}}$	**4.1%**
Medium	MNIS	5,500	6.67×10^{-4}	3.4%
	AOSM	5,950	6.65×10^{-4}	3.7%
	ACS	4,320	6.59×10^{-4}	4.6%
	HSCS	5,900	7.37×10^{-4}	6.7%
	YieldAgent	**5,550**	$\mathbf{6.70 \times 10^{-4}}$	**2.96%**
High	MNIS	6,800	6.42×10^{-4}	7.1%
	AOSM	9,950	6.66×10^{-4}	3.6%
	ACS	9,680	6.58×10^{-4}	4.7%
	HSCS	9,750	7.18×10^{-4}	3.9%
	YieldAgent	**9,200**	$\mathbf{6.79 \times 10^{-4}}$	**1.8%**

Fail Rate: Estimated failure rate; **Rel. Err.:** Relative estimation error compared to MC method.

C. Precision-Aware Capability Evaluation

This phase of experimentation assesses YieldAgent's ability to meet varying precision requirements defined by convergence factors $\epsilon = \{0.1, 0.05, 0.01\}$. The evaluation employs relative error and simulation count as primary metrics, with convergence determined when the relative variation in yield estimates falls below the specified ϵ threshold across the sliding window.

Table II quantifies YieldAgent's performance across precision levels. At $\epsilon = 0.1$, the framework achieves 2.7% relative error using 5,550 simulations through ACS-MNIS method integration, while HSCS requires 16,100 simulations for comparable accuracy (2.6% error). Under stricter precision requirements ($\epsilon = 0.05, 0.01$), YieldAgent maintains superior efficiency through dynamic method selection, achieving relative errors of 1.0% and 0.4% with 8,760 and 15,600 simulations respectively. This performance represents a significant improvement over single-method approaches, which require 20,800 and 23,300 simulations for similar precision levels.

TABLE II
YIELD ANALYSIS WITH VARYING CONVERGENCE FACTORS

ϵ	Method	Fail Rate	# Sim	Rel. Err.
0.1	ACS	6.40×10^{-4}	5,580 (1.0×)	7.3%
	AOSM	6.61×10^{-4}	6,750 (1.22×)	4.3%
	HSCS	6.69×10^{-4}	16,100 (2.90×)	2.6%
	YieldAgent	$\mathbf{6.79 \times 10^{-4}}$	**5,550 (1×)**	**2.7%**
0.05	ACS	6.53×10^{-4}	7,140 (0.82×)	5.5%
	AOSM	7.03×10^{-4}	9,800 (1.12×)	1.7%
	HSCS	6.74×10^{-4}	20,800 (2.37×)	2.5%
	YieldAgent	$\mathbf{6.85 \times 10^{-4}}$	**8,760 (1×)**	**1.0%**
0.01	ACS	6.83×10^{-4}	13,120 (0.84×)	1.2%
	HSCS	7.01×10^{-4}	14,350 (0.92×)	1.4%
	MNIS	6.85×10^{-4}	23,300 (1.5×)	0.9%
	YieldAgent	$\mathbf{6.88 \times 10^{-4}}$	**15,600 (1×)**	**0.4%**

D. Topology-Aware Capability Evaluation

The final experimental phase examines YieldAgent's adaptation capabilities across TSMC 12nm and SMIC 40nm technology nodes, where process variations and failure mechanisms differ significantly. The 12nm node exhibits tighter threshold voltage variation ($\sigma_{V_{\text{th}}} = 30$mV) and more complex failure regions compared to 40nm ($\sigma_{V_{\text{th}}} = 45$mV), presenting distinct estimation challenges.

Table III presents comparative results across nodes and precision requirements. For the 40nm node, YieldAgent achieves superior estimation accuracy through strategic method combinations: ACS-MNIS for $\epsilon = 0.1$ (2.7% error, 20x simulation reduction), and ACS-HSCS for stricter requirements (1.0% and 0.4% error, with 12.67x and 7.12x reductions).

The framework adapts differently for 12nm technology, employing ACS-AOSM combinations to handle complex failure distributions. This strategy achieves relative errors of 7.6%, 3.4%, and 1.9% while reducing simulation counts by factors of 12.34x, 9.03x, and 5.84x respectively. The performance differential between nodes demonstrates YieldAgent's topology-aware capabilities, selecting different method combinations based on technology-specific characteristics: ACS-MNIS/HSCS for 40nm's wider variation spread, and ACS-AOSM for 12nm's complex failure regions.

TABLE III
YIELDAGENT'S ESTIMATION WITH TECHNOLOGY NODES AND CONVERGENCE FACTORS ϵ

Node	ϵ	Fail Rate	# Sim	Rel. Err.	Ensembled Method
40 nm	MC	6.91×10^{-4}	111,000	-	-
	0.1	6.79×10^{-4}	5,550(20×)	2.7%	ACS, MNIS
	0.05	6.85×10^{-4}	8,760(12.67×)	1.0%	ACS, HSCS
	0.01	6.88×10^{-4}	15,600(7.12×)	0.4%	ACS, HSCS
12 nm	MC	2.11×10^{-5}	569,000	-	-
	0.1	2.27×10^{-5}	46,000(12.34×)	7.6%	ACS, AOSM
	0.05	2.18×10^{-5}	63,000(9.03×)	3.4%	ACS, AOSM
	0.01	2.15×10^{-5}	87,400(5.84×)	1.9%	ACS, AOSM

V. CONCLUSION

This research addresses significant challenges in yield estimation for semiconductor manufacturing through the introduction of YieldAgent, the first LLM-powered multi-agent framework for dynamic yield estimation. Through its novel three-layer architecture integrating LLM reasoning capabilities with domain-specific knowledge, YieldAgent demonstrates superior adaptability and efficiency across diverse circuit topologies and technology nodes. Experimental validation across 12nm and 40nm nodes shows that YieldAgent reduces computational overhead by up to 2.9× while maintaining or exceeding state-of-the-art accuracy, achieved through three key innovations: dynamic budget-aware resource allocation, precision-guided estimation, and topology-aware method selection. The framework's success in balancing computational efficiency with estimation precision, while adapting to varying technological contexts, establishes a foundation for future innovations in semiconductor manufacturing, with potential extensions to other areas of electronic design automation and integration with emerging AI techniques for further enhancement of yield estimation capabilities.

REFERENCES

[1] J. Yao, Z. Ye, and Y. Wang, "An efficient sram yield analysis and optimization method with adaptive online surrogate modeling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1245–1253, 2014.

[2] M. Wang, F. Yang, C. Yan, X. Zeng, and X. Hu, "Efficient bayesian yield optimization approach for analog and sram circuits," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2017, pp. 1–6.

[3] S. Yin, G. Dai, and W. W. Xing, "High-dimensional yield estimation using shrinkage deep features and maximization of integral entropy reduction," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 283–289.

[4] L. Dolecek, M. Qazi, D. Shah, and A. Chandrakasan, "Breaking the simulation barrier: Sram evaluation through norm minimization," in *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2008, pp. 322–329.

[5] W. Wu, S. Bodapati, and L. He, "Hyperspherical clustering and sampling for rare event analysis with multiple failure region coverage," in *Proceedings of the 2016 on International Symposium on Physical Design*, 2016, pp. 153–160.

[6] S. M. Xie, A. Raghunathan, P. Liang, and T. Ma, "An explanation of in-context learning as implicit bayesian inference," *arXiv preprint arXiv:2111.02080*, 2021.

[7] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[8] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[9] H. Pearce, B. Tan, and R. Karri, "Dave: Deriving automatically verilog from english," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 2020, pp. 27–32.

[10] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," *arXiv preprint arXiv:2402.03375*, 2024.

[11] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language model," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[12] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," *arXiv preprint arXiv:2403.11671*, 2024.

[13] X. Shi, H. Yan, Q. Huang, J. Zhang, L. Shi, and L. He, "Meta-model based high-dimensional yield analysis using low-rank tensor approximation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[14] Y. Liu, G. Dai, and W. W. Xing, "Seeking the yield barrier: High-dimensional sram evaluation through optimal manifold," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[15] L. Dolecek, M. Qazi, D. Shah, and A. Chandrakasan, "Breaking the simulation barrier: Sram evaluation through norm minimization," in *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2008, pp. 322–329.

[16] T. Haine, J. Segers, D. Flandre, and D. Bol, "Gradient importance sampling: An efficient statistical extraction methodology of high-sigma sram dynamic characteristics," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 195–200.

[17] W. Hu, Z. Wang, S. Yin, Z. Ye, and Y. Wang, "Sensitivity importance sampling yield analysis and optimization for high sigma failure rate estimation," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 895–900.

[18] X. Shi, F. Liu, J. Yang, and L. He, "A fast and robust failure analysis of memory circuits using adaptive importance sampling method," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[19] X. Shi, H. Yan, J. Wang, X. Xu, F. Liu, L. Shi, and L. He, "Adaptive clustering and sampling for high-dimensional and multi-failure-region sram yield analysis," in *Proceedings of the 2019 International Symposium on Physical Design*, 2019, pp. 139–146.

[20] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng, "Large language models as general pattern machines," *arXiv preprint arXiv:2307.04721*, 2023.

[21] T. Liu, N. Astorga, N. Seedat, and M. van der Schaar, "Large language models to enhance bayesian optimization," *arXiv preprint arXiv:2402.03921*, 2024.

[22] S. Yang, H. Zhao, S. Zhu, G. Zhou, H. Xu, Y. Jia, and H. Zan, "Zhongjing: Enhancing the chinese medical capabilities of large language model through expert feedback and real-world multi-turn dialogue," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 17, 2024, pp. 19 368–19 376.

[23] G. Chen, K. Zhu, S. Kim, H. Zhu, Y. Lai, B. Yu, and D. Z. Pan, "Llm-enhanced bayesian optimization for efficient analog layout constraint generation," *arXiv preprint arXiv:2406.05250*, 2024.

[24] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu *et al.*, "Autogen: Enabling next-gen llm applications via multi-agent conversation."

BEVSA: A Real-Time Bird's-Eye-View Semantic Segmentation Accelerator for Multi-Camera System

Sangho Lee*, Jueung Jung*, Wuyoung Jang*, Jihyeon Hwang*, and Kyuho Lee†
*Graduate School of Artificial Intelligence / Department of Electrical Engineering, UNIST, Republic of Korea
†School of Electrical and Electronics Engineering, Yonsei University, Republic of Korea
lsh9901@unist.ac.kr / kyuho.jsn.lee@yonsei.ac.kr

***Abstract*— A bird's-eye-view (BEV) semantic segmentation accelerator (BEVSA) is proposed for real-time 3D space perception in multi-camera system (MCS). The view transformation from multi-camera-view to BEV obstructs the real-time operation on edge devices through 68.3 ms of time consumption during BEV pooling, which requires sorting and irregular memory access over wide searching space. Moreover, the 69.1% high average input activation sparsity of the segmentation process for the transformed BEV features results in excessive meaningless computations. For the real-time implementation of BEV semantic segmentation on edge platforms, this paper proposes two key features: 1) Block-decomposed hierarchical BEV pooling cluster that partitions the searching space in an MCS-suitable way and supports parallel pooling, achieving 43.2× speedup for BEV pooling over the edge computing platform; 2) Coarse-to-fine-grained zero skipping convolution cluster, which conducts coarse-grained zero skipping for all-zero channels and tile-wise fine-grained zero skipping, improving the convolution throughput by 1.61×. Implemented with 28nm technology, and evaluated on two representative MCS datasets, BEVSA finally achieves 23.1 frames-per-second of real-time BEV segmentation throughput with 167.4× higher energy-per-frame over the edge computing platform.**

Keywords*— *3D perception, BEV pooling, BEV semantic segmentation, heterogeneous cluster, multi-camera system, parallelization, sparsity exploitation.

I. Introduction & Background

Emerging autonomous robots require a comprehensive understanding of wide 3D space for flawless interaction with the surrounding environment. Bird's-eye-view semantic segmentation (BEV-SS) plays a crucial role as an effective solution for 3D perception through a multi-camera system (MCS), which is widely deployed in autonomous driving. The bird's-eye-view (BEV) resolves the scale variation and occlusion challenges inherent in vision-based 3D perception by transforming the ego-view of each camera into an integrated view. It facilitates the expansion of the autonomous system to multi-camera settings for high accuracy, being well-suited for 3D perception of objects and road elements for subsequent functions, such as planning and control in autonomous driving [1]. Recently Lift-splat [2] was proposed for BEV-SS, which generates the depth-estimated camera features using a convolutional neural network (CNN), followed by rasterization through BEV pooling for view transformation. By minimizing the interference among cameras during its training for depth estimation, Lift-splat [2] recovers from partial damage of cameras in inference and maintains high accuracy, enabling robust view integration even with the various configurations of MCS. Therefore, it has been widely adopted in BEV applications [3]-[5].

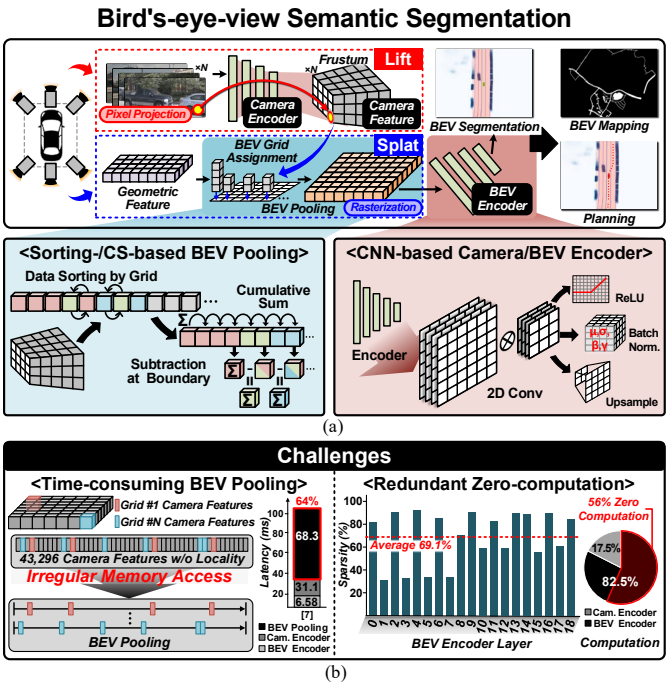
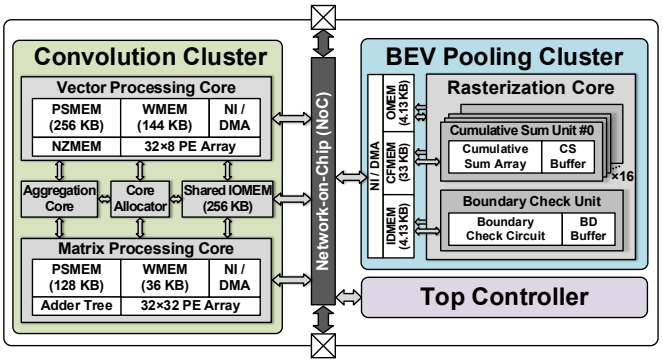


Fig. 1. Explanation of the bird's-eye-view semantic segmentation. (a) The entire process through lift and splat steps and (b) challenges of real-time implementation and motivation for hardware accelerator.

Fig. 1(a) illustrates the overall process of BEV-SS [2] with MCS, consisting of two steps: *lift* and *splat*. During the *lift* step, a CNN-based camera encoder performs pixel-wise depth estimation on each 2D image to project into a 3D camera features as a frustum. Then, at the *splat* step, the features in each frustum are assigned to the nearest BEV grid according to the corresponding geometric features. The assigned features are rasterized through BEV pooling within each grid, adopting cumulative sum (CS). After sorting the data according to its grid, CS is applied across the entire feature, followed by subtraction at each grid boundary. Afterward, the CNN-based BEV encoder takes the rasterized BEV features as input to produce segmentation outputs.



PSMEM: Partial-sum Memory, WMEM: Weight Memory, NZMEM: Nonzero Memory, IOMEM: I/O Memory, BD: Boundary IDMEM: Grid ID Memory, CFMEM: Camera Feature Memory, OMEM: Output Memory

Fig. 2. Overall architecture of the proposed BEVSA.

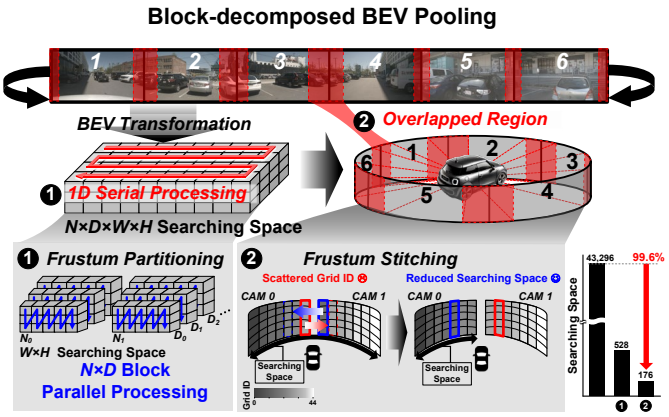


Fig. 3. The proposed frustum partitioning and stitching of block-decomposed BEV pooling for searching space reduction.

To ensure the safety of autonomous driving, satisfying the real-time constraint is imperative, at least 12 Hz (83.3 ms/frame) according to [6]. However, real-time implementation of BEV-SS on an edge computing platform [7] is unfeasible for two reasons, as illustrated in Fig. 1(b). First, the rasterization process utilizes time-consuming BEV pooling. Due to the irregular memory access in BEV pooling induced by extracting features with the same grid ID among 43,296 camera features [8], the BEV pooling accounts for 64% of total latency. Additionally, the BEV encoder, which accounts for 82.5% of convolution operations, incurs substantial computational load with redundant zero computations. The input of each layer in the BEV encoder exhibits an average sparsity of 69.1%, resulting from the sparse BEV features and ReLU activation function. Therefore, the redundant zero-computations constitute 55.7% of total convolution operations across the two encoders.

In this paper, a real-time BEV semantic segmentation accelerator, named BEVSA, for outdoor MCS is proposed. BEVSA is designed with a heterogeneous cluster architecture, facilitating parallel BEV pooling and zero skipping optimized for BEV-SS through the following SW-HW co-optimizations:

- A block-decomposed BEV pooling method is proposed for removing redundant searching operations and inter-block parallel BEV pooling without sorting.
- A hierarchical BEV pooling cluster (BPC) is designed for further reduction of pooling searching space by 87.5% and supporting intra-block parallelism of BEV pooling.
- A coarse-grained zero skipping method with on-the-fly pruning is introduced for 50.1% reduced computation in the BEV encoder by permanently removing meaningless channels.
- A fine-grained zero skipping method on hybrid sparse-dense convolution cluster (CC) is proposed for efficient CNN inference utilizing sparsity prediction through inter-frame similarity.

BEVSA is designed in RTL Verilog and implemented in 28nm CMOS technology to demonstrate its feasibility. 2.45× and 2.33× throughput improvements are achieved compared with baseline [7] for BEV-SS with [2] on nuScenes [6] and Lyft Level 5 [9] datasets. The rest of the paper is organized as follows. Section II introduces the overall architecture of the proposed BEVSA. Sections III and IV describe the hierarchical BPC with block-based algorithm optimization and the coarse-to-fine-grained zero skipping, supported by hybrid sparse-dense CC, respectively. Section V shows the evaluation results, followed by the conclusion in Section VI.

II. Overall Architecture

Fig. 2 illustrates the overall architecture of the proposed BEVSA. It comprises a CC for the camera encoder and BEV encoder, as well as a BPC for BEV pooling and a top controller interconnected through the network-on-chip (NoC). To facilitate efficient convolution processing for inputs with various sparsity of BEV encoder, the basis of CC rises in the hybrid architecture of [10] for sparsity exploitation. It consists of a vector processing core (VPC) for high-sparsity input and a matrix processing core (MPC) for dense input, with each employing 32×8 processing element (PE) array and 32×32 PE array, respectively. The aggregation core, core allocator, and 256 KB shared I/O memory are included between the pair of VPC and MPC. The BPC comprises a rasterization core, a camera feature memory, a grid ID memory, and an output memory. The rasterization core contains 16 CS units with the CS array and a boundary check (BC) unit with the BC circuit. The CS array iteratively accumulates input features, and the results are stored in the CS buffer. The boundary information obtained from the BC circuit of the BC unit is utilized for CS trick [2], which performs subtraction between boundary CS results for BEV pooling. Consequently, only the subtracted results corresponding to boundaries are stored in output memory with their grid IDs.

III. Block-Decomposed Hierarchical BEV Pooling Cluster

A. Block-Decomposed BEV Pooling

Irregular memory access is the most crucial factor in latency during the BEV pooling process, which is caused by sorting and BC over a wide searching space based on multiple camera features. To reduce the extensive searching space, previous 3D perception accelerators [11] and [12] divided input points into

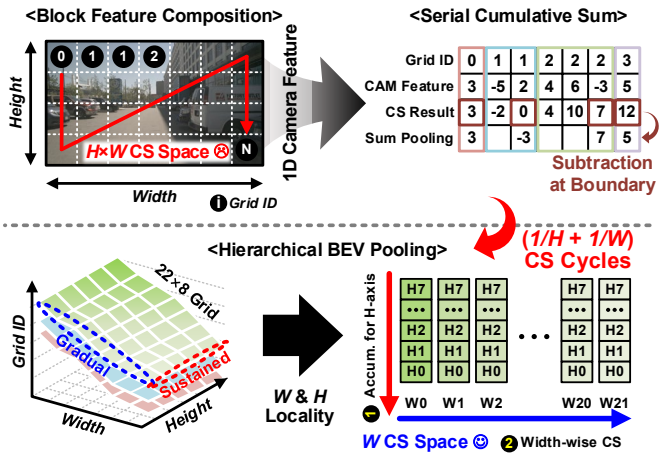


Fig. 4. The operation of hierarchical BEV pooling within a block based on grid ID locality across dimensions.

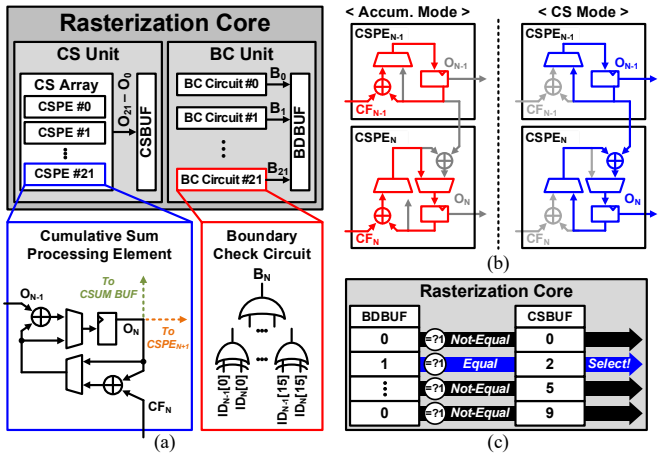


Fig. 5. (a) The architecture of cumulative sum processing element (CSPE) and boundary check circuit (BCC) in CS unit and BC unit, (b) two operating modes of CSPE, and (c) the CS result storage flow based on boundary condition.

distinct spaces. Reference [11] presented a *window-search algorithm* for monocular RGB-D data, while [12] proposed a *spherical-coordinate-based bin searching* for LiDAR data. However, none of them dealt with the overlapped regions between sensors because they leveraged only a single sensor. In MCS, overlapping regions among the sensors must be considered when partitioning the space, as illustrated in Fig. 3. This requirement hinders the direct adoption of previous single-sensor-based space partitioning methods. Therefore, this paper proposes a block-based BEV, which divides the MCS inputs while handling overlapping regions between multiple images.

The bottom of Fig. 3 illustrates the proposed block-decomposed BEV pooling, which reduces the searching space by utilizing frustum partitioning based on the grid ID locality across dimensions. After the frustum partitioning on each camera and depth dimensions in 1D camera features, grid IDs appear ordered within each block, enabling CS to be performed without additional sorting. Each partitioned block can be processed in parallel within a searching space reduced by 82× for locating and accumulating the features of a specific grid ID.

However, duplicated searching space occurs as grid IDs are partitioned across distinct blocks, which necessitates accessing camera features of neighboring blocks. The proposed frustum stitching eliminates redundant searches by reorganizing the features of each camera boundary. The overlapping regions are gathered based on the width of neighboring camera blocks, sorted, and then reallocated according to the grid ID distribution of each block. Through frustum partitioning and stitching, the searching space of BEV pooling is reduced by 99.6%, enabling flawless parallelization across each block.

B. *Hierarchical BEV Pooling Cluster*

Fig. 4 illustrates the hierarchical BEV pooling process within each block. The conventional CS trick requires serial computation on camera features encoded along the number of height (H) and width (W). This requires $H \times W$ registers to store the CS results along with the BC for the maximum grid IDs. On the other hand, the proposed hierarchical BPC enhances the computational parallelism by leveraging the locality of grid IDs with respect to the height and width. The grid IDs within a block exhibit a linear variation along the width axis while remaining constant along the height axis. Since variation in height has a minimal impact on the BEV grid ID, width-wise parallel accumulation of camera features is performed H times without grid ID comparison with only W registers. Subsequently, the CS is applied along the width axis during W cycles. Therefore, the hierarchical BEV pooling method reduces the amounts of CS cycles to $(1/H + 1/W)$ times that of the conventional methods. Given that BEV-SS typically adopts H and W values of 8 and 22, respectively, the BPC finally achieves 1.83× higher throughput.

Fig. 5(a) illustrates the detailed architecture of the rasterization core with CS unit and BC unit, containing a CS array and BC circuits. The CS array comprises 22 CS processing elements (CSPEs) and conducts hierarchical width-wise CS for 16 channels of input camera features. Fig. 5(b) describes two operating modes of CSPE: accumulation mode and CS mode. In accumulation mode, parallel accumulation of input camera features according to height axis is performed with a register of each CSPE. In CS mode, serial width-wise cumulative sums across the registers of neighboring CSPEs are conducted. Meanwhile, the BCU figures out the grid boundaries through bitwise XOR operations on each 16-bit grid ID, followed by a 16-bit OR operation. The CS results and boundaries are temporarily stored in the CS buffer and boundary buffer, respectively, and only the boundary CS results are transmitted to external memory, as illustrated in Fig. 5(c), generating BEV pooling output for each grid. As a result, the hierarchical BPC design maximizes the computational parallelism for BEV pooling by transforming serial $N \times D \times W \times H$ operations into parallel $W+H$ operations, and further reduces the searching space of block-decomposed BEV pooling by 87.5%.

IV. Coarse-to-Fine-grained Zero Skipping Convolution Cluster

A. *Coarse-grained Zero Skipping through On-the-Fly Pruning*

Redundant zero-computations, which account for 68% of the BEV encoder operations, can be skipped through sparsity

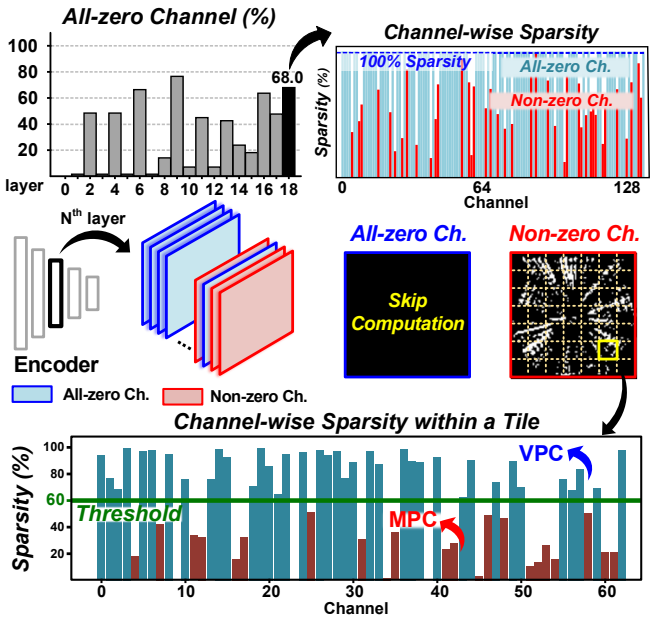


Fig. 6. Coarse-to-fine-grained zero skipping. On-the-fly pruning for coarse-grained zero skipping of meaningless channels, fine-grained zero skipping of non-zero channels on hybrid convolution cluster with sparsity-based core allocation.

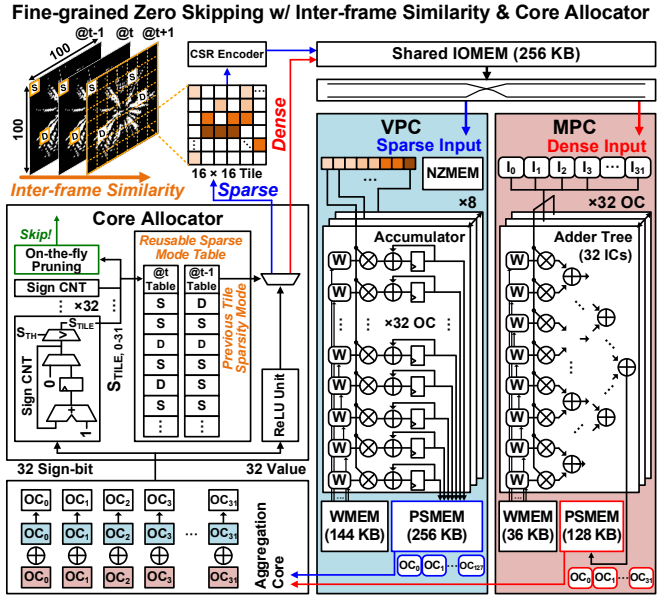


Fig. 7. The architecture of hybrid sparse-dense convolution cluster with VPC, MPC, aggregation core, and core allocator.

exploitation to enhance the throughput. Fig. 6 illustrates the input sparsity characteristics of the BEV encoder. As depicted in the upper side of Fig. 6, meaningless channels, where all input features of a channel are zero, constitute 30.7% of the entire BEV encoder on average. Furthermore, 99.97% of all-zero channels maintain their sparsity across consecutive frames. To leverage the consecutive sparsity, on-the-fly pruning is introduced to skip the meaningless channels. Once an all-zero channel is detected, it is permanently excluded from computations in subsequent frames. Therefore, on-the-fly pruning method reduces the computational load by 50.1% and 52.8% on the Lyft Level 5 [9] and nuScenes [6] datasets, respectively, with only 0.3% and 0.08% drop in mean intersection over union (mIoU).

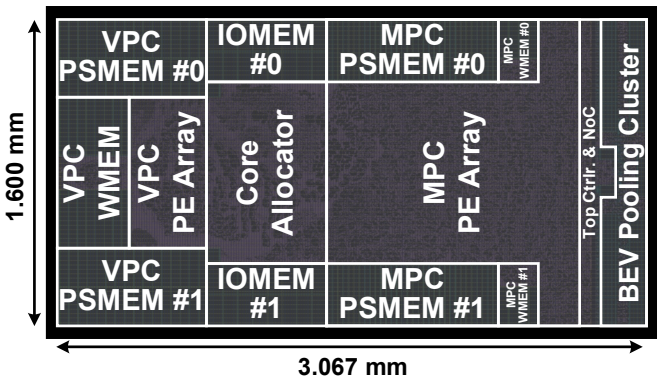
B. Tile-wise Fine-grained Zero Skipping on Hybrid Convolution Cluster

Sparsity remains in non-zero channels. The bottom of Fig. 6 illustrates the fluctuation in sparsity across the input channels within a 16 × 16 tile. To support the polarized sparsity, a hybrid CC inspired by [10] is utilized to separate operations based on the input sparsity. The hybrid CC consists of VPC, which adopts a sparsity exploitation architecture, and MPC, which employs a weight stationary design. As the computational efficiency of VPC declines when input sparsity falls below 60% [10], core allocation is determined based on a sparsity threshold of 60%. Sparse inputs are processed by the VPC for throughput increase by zero skipping. In contrast, dense inputs are processed by the MPC for memory access reduction through the enhanced reusability of the weight parameters.

Fig. 7 illustrates the hybrid sparse-dense CC. Input tiles are loaded from the shared I/O memory to VPC and MPC according to the sparsity of input data. Each core loads weight data from weight memory, operating with double-buffering. In each cycle, the VPC loads eight non-zero inputs encoded through compressed sparse row (CSR) encoding. The CSR parameters, such as row pointer and column index, are loaded to non-zero memory for decoding. Then, it performs MAC operations with 32 weights to generate outputs for 32 output channels (OCs). These results are transferred to the internal partial-sum memory, where the results of PE arrays are accumulated. The MPC processes 32 input channels and 32 OCs each cycle. Then, the resulting 32 OCs from adder trees are accumulated in partial-sum memory. Once computations for a given output tile are done, the results from each core are combined through the aggregation core. Subsequently, a core allocator determines the processing core of the output tile for the next layer according to the sparsity.

Reference [10] proposed a sampling-based task allocation method for hybrid cores, utilizing the sparsity of the sampled pixels' inference result during MLP-based MAC operations for individual pixels. However, in CNNs, since the output features depend on all neighboring pixels within the receptive field, it is unsuitable for CNN-based BEV-SS. Therefore, exploiting sparsity in BEV-SS requires a data sparsity analysis after the computation for an output tile. It causes either memory overhead for temporary tile data storage or significant latency for input activation reordering. To address these challenges, the fine-grained zero skipping method by reusing inter-frame similarity-based sparsity is proposed.

Given the inter-frame similarity, 98.2% of the sparsity mode for each tile remains consistent across adjacent frames. By leveraging this consistency, a reusable sparsity mode table (RSMT) can be obtained from the output of the previous frame to assign each output tile to the appropriate core based on sparsity. The core allocator accumulates sign bit values from the results of the aggregation core prior to the ReLU unit, as negative values are set to zero after the ReLU activation function. Through the accumulated values, channel sparsity for each output tile is assessed, and the RSMT is generated for the



		BEVSA (1 CC & 1 BPC)			
Technology		28nm CMOS			
Operating Condition		0.9 V, 500 MHz			
Area / Peak Power		4.91 mm^2 / 877.6 mW			
On-chip SRAM Size		861 KB			
Core Type of CC		VPC		MPC	
# of MAC		256 / Core		1,024 / Core	
Input Activation Sparsity*		0%	90%	0%	90%
Performance*** [GOPS]	0% All-zero**	256	2,560	1,024	
	40% All-zero	427	4,267	1,707	
Power [mW]	0% All-zero	228.8	278.6	286.6	
	40% All-zero	232.5	315.5		
Energy Efficiency [TOPS/W]	0% All-zero	1.12	9.19	3.57	
	40% All-zero	1.84	13.5	5.96	

**Sparsity within Tiles of BEV Encoder / **All-zero Channels of BEV Encoder*
**** 1 MAC = 2 OP*

Fig. 8. The photograph of the proposed BEVSA layout and the summary of its performance.

next frame. While the RSMT for the next frame is being generated, the values after the ReLU unit are allocated to either VPC or MPC for the following layer computations. Regardless of the sparsity calculation in the current frame, core allocation only considers the sparsity of the previous frame, referencing the RSMT. The sparse tiles assigned to the VPC are encoded through CSR encoding, with only the non-zero data stored in the shared I/O memory.

V. Evaluation

A. Experimental Setup

CAD Tools. For the area and energy consumption evaluation, the proposed accelerator is designed in RTL Verilog and implemented by Synopsys Design Compiler for synthesis and the IC Compiler II for place and route in a 28nm CMOS technology process. The BEVSA is evaluated through cycle-accurate RTL simulation and power dissipation simulation using Synopsys PrimePower after the implementation.

Datasets and Model. To validate the throughput of the proposed BEVSA, the experiment is conducted on Lift-splat [2]. Two representative MCS datasets, nuScenes [6] and Lyft Level 5 [9], are used for evaluation in multifaceted environments with different camera settings. The pre-trained network models are sourced from TorchVision and EfficientNet-PyTorch.

Baseline Platform. For comparison with the previous platforms, Lift-splat is executed on an edge computing device, NVIDIA Jetson AGX Orin Developer Kit [7]. The evaluation

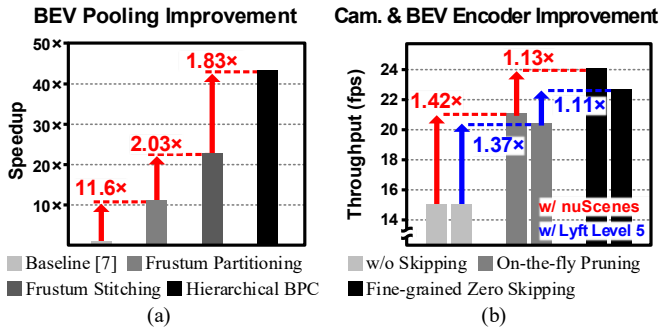


Fig. 9. Step-by-step performance improvement. (a) The speedup of BEV pooling, and (b) the throughput improvement of CNN-based encoders.

is conducted using the PyTorch [13] framework on 2048-core NVIDIA Ampere architecture GPU.

Implementation Results. Fig. 8 illustrates the post-layout of the proposed BEVSA and its performance summary. The accelerator occupies 4.91 mm^2 with a total of 861 KB on-chip SRAM and operates at a maximum frequency of 500 MHz at 0.9 V supply voltage, consuming 877.6 mW of peak power. The performance and energy efficiency are varied according to the sparsity of input activation in tile and the proportion of all-zero channels. The power consumption of the VPC slightly increases with higher input activation sparsity because its sparsity exploitation reduces accumulation latency, which results in more frequent partial-sum memory access during each unit operating time. However, the VPC significantly enhances performance at higher input activation sparsity by employing the proposed coarse-to-fine-grained zero skipping method. When comparing input activation sparsity ranging from 0% to 90%, the VPC improves energy efficiency by up to 8.21× despite a 21.8% increase in power consumption. Under conditions of 40% all-zero channels and 90% input activation sparsity within tiles, the VPC and MPC demonstrate peak performances of 4.267 and 1.707 TOPS with INT16 precision, respectively, achieving 12.1× and 1.67× higher energy efficiency than the non-sparsity condition.

B. Experimental Results

Improvement of BEV Pooling. Fig. 9(a) illustrates the step-by-step throughput improvement achieved through block-decomposed hierarchical BEV pooling. As discussed in Section III, the required searching space is lowered by 99.6% through the block-decomposed BEV pooling. After frustum partitioning, because of the decreased searching space and block-wise parallel computation, it improves the throughput by 11.6× compared to the baseline. Moreover, the overlapping regions between adjacent cameras are addressed by frustum stitching, achieving 2.03× higher throughput.

After the block decomposition, the hierarchical BPC facilitates parallel feature accumulation across height dimensions, further enhancing the pooling throughput by 1.83×. As a result, BEVSA with BPC achieves 43.2× higher BEV pooling throughput than the baseline. Furthermore, as the number of registers for the CS is diminished, the area of the CS array is decreased by 46.1% compared to 176 CSPEs for serial computation.

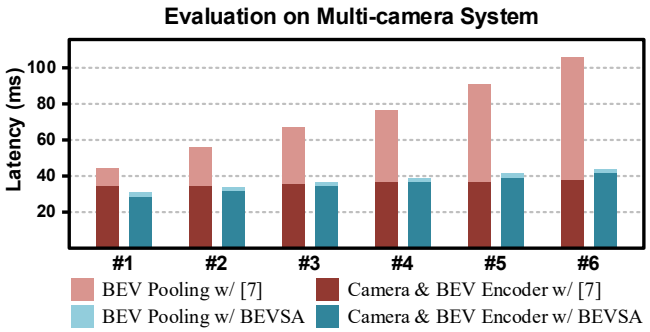


Fig. 10. The latency evaluation on multi-camera system with different number of camera images per frame.

Coarse-to-Fine-grained Zero Skipping. Fig. 9(b) illustrates the convolution throughput improvement by coarse-to-fine-grained zero skipping. The BEV features, input activation of the BEV encoder, are derived from depth estimation through the camera encoder and BEV pooling. The features exhibit high sparse data characteristic regardless of the camera rig, and the sparsity is maintained through the ReLU activation function. Specifically, the extracted BEV features with nuScenes and Lyft Level 5 datasets demonstrate high sparsity levels of 81.8% and 80.3%, which result in 69.1% and 45.7% of high average input activation sparsity in the BEV encoder, respectively. Therefore, almost 50% of computations of the whole BEV encoder are eliminated by on-the-fly pruning, and the throughput increases by 1.42× and 1.37× with each dataset, respectively. Subsequently, the fine-grained zero skipping through the hybrid CC further improves the throughput by 1.13× and 1.11×, achieving 1.61× and 1.52× of the entire throughput improvement with nuScenes and Lyft datasets, respectively, validating the applicability across various camera rigs.

Impact of Number of Cameras. The BEV pooling significantly contributes to latency on the edge computing platform due to the sorting and irregular memory access. Therefore, the searching space of BEV pooling has a significant impact on the latency within [7]. Fig. 10 illustrates the latency evaluation for the proposed accelerator over the number of camera images. As the number of input camera images per frame increases, the latency in [7] grows substantially with the expanded pooling searching space. In contrast, the proposed BEVSA shrinks this searching space by 99.95%, thereby minimizing the impact of BEV pooling on overall latency. Although the latency in BEVSA also increases with the number of input images through the higher computational load in the camera encoder, the overall change accounts for 6.59% of total encoder operations, resulting in a significantly lower latency increase rate than [7]. Consequently, BEVSA supports more efficient processing in MCS with a larger number of input camera images compared to [7].

Total Performance. Table I shows the results of the comparison with conventional platforms for BEV-SS. BEVSA achieved 2.45× throughput improvement compared to the edge computing platform [7], supporting 23.1 frames-per-second (fps) of real-time performance with 68.3× reduced power consumption. Although the CPU + GPU platform [14] satisfies the real-time constraint, its high power consumption is unsuitable for edge computing. BEVSA improved the energy-per-frame by 167.4× and 272.9× compared to [7] and [14].

TABLE I. PERFORMANCE COMPARISON TABLE

Device	# Cores	Technology	Area	Frequency	Throughput	Power	Energy-per-Frame
AGX Orin	12 CPU + 2,048 CUDA	8 nm	628.4 mm^2	1,300 MHz	9.43 fps	60 W	6,363 mJ/frame
Gold 6230 + RTX 2080 Ti	20 CPU + 4,352 CUDA	12 nm	754 mm^2	1,350 MHz	24.1 fps	250 W	10,373 mJ/frame
BEVSA w/ 1 CC	**1,632 PEs**	**28 nm**	**4.91 mm^2**	**500 MHz**	**23.1 fps**	**0.878 W**	**38.01 mJ/frame**
BEVSA w/ 2 CC	**2,912 PEs**	**28 nm**	**9.34 mm^2**	**500 MHz**	**44.6 fps**	**1.664 W**	**37.31 mJ/frame**

The proposed BEVSA minimizes the searching space of BEV pooling, making the overall latency mainly dependent on the camera and BEV encoder. Therefore, a range of hybrid CC configurations and clock frequencies can be implemented to meet specific application requirements. Increasing the number of CCs proportionally enhances throughput but incurs additional power and area overhead. For example, operating two CCs at a clock frequency of 500 MHz results in an additional power consumption of 0.786 W and an area overhead of 4.43 mm², while achieving a throughput of 44.6 fps. This performance meets the requirements of MCS as well as those of commercial cameras used in advanced autonomous driving systems (~36 Hz).

VI. CONCLUSION

In this paper, we present a real-time bird's-eve-view semantic segmentation accelerator (BEVSA) for 3D space perception in multi-camera system (MCS) with heterogeneous clusters. The block-decomposed hierarchical BEV pooling cluster reduces the searching space of sorting and irregular memory access by supporting inter- and intra-block parallel computations. It achieves a 43.2× speedup for BEV pooling over the edge computing platform. In addition, the hybrid convolution cluster with coarse-to-fine-grained zero skipping method is proposed to improve the throughput for sparse convolution neural network (CNN) inference caused by BEV transformation. It skips meaningless zero computations by utilizing channel-wise on-the-fly pruning and tile-wise inter-frame sparsity similarity, achieving 1.61× higher throughput on camera and BEV encoder. We evaluate the implemented BEVSA with 28nm CMOS technology compared to the edge computing platform by running two representative MCS datasets. As a result, the BEVSA achieved 23.1 frames-per-second of real-time BEV semantic segmentation throughput with 167.4× higher energy-per-frame over the edge computing platform in the MCS.

ACKNOWLEDGMENTS

This work was supported in part by the National Research Foundation (NRF) of Korea grant funded by the Korea government (MSIT) (RS-2023-00212222); and in part by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2024-00399817). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] H. Li, C. Sima, J. Dai, W, Wang, L. Lu and H. Wang, "Delving Into the Devils of Bird's-Eye-View Perception: A Review, Evaluation and Recipe," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 4, pp. 2151-2170, April 2024.

[2] J. Philion and S. Fidler, "Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d," in *Proc. of the European Conference on Computer Vision (ECCV)*, 2020, pp. 194-210.

[3] X. Gu, G. Song, I. Gilitschenski, M. Pavone and B. Ivanovic, "Producing and Leveraging Online Map Uncertainty in Trajectory Prediction," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024, pp. 14521-14530.

[4] S. Hu, L. Chen, P. Wu, H. Li, J. Yan and D. Tao, "ST-P3: End-to-end vision-based autonomous driving via spatial-temporal feature learning," in *Proc. of the European Conference on Computer Vision (ECCV)*, 2022, pp. 533-549.

[5] C. Ge, J. Chen, E. Xie, Z. Wang, L. Hong, H. Lu, Z. Li and P. Luo, "MetaBEV: Solving Sensor Failures for 3D Detection and Map Segmentation," in *Proc. of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023, pp. 8721-8731.

[6] H. Caesar, V. Bankiti, A. -H. Lang, S. Vora, V. -E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Bejibom, "nuScenes: A Multimodal Dataset for Autonomous Driving," in *Proc. of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11621-11631.

[7] NVIDIA. Jetson Orin Series. Accessed: Nov, 7, 2024. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/

[8] X. Feng, W. Sun, X. Lin, S. Fan, H. Yang and Y. Liu, "A 28nm 1.2GHz 5.27TOPS/W Scalable Vision/Point Cloud Deep Fusion Processor with CAM-based Universal Mapping Unit for BEVFusion Applications," in *Proc. of IEEE Custom Integrated Circuits Conference (CICC)*, 2024, pp. 1-2.

[9] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov and P. Ondruska, "One Thousand and One Hours: Self-driving Motion Prediction Dataset," in *Proc. of Conference on Robot Learning*, 2020, pp. 409-418.

[10] D. Han, J. Ryu, S. Kim, S. Kim, J. Park and H. -J. Yoo, "MetaVRain: A Mobile Neural 3-D Rendering Processor With Bundle-Frame-Familiarity-Based NeRF Acceleration and Hybrid DNN Computing," *IEEE Journal of Solid-State Circuits*, vol. 59, no. 1, pp. 65-78, Jan. 2024.

[11] D. Im, G. Park, J. Ryu, Z. Li, S. Kang and H. -J. Yoo, "DSPU: An Efficient Deep Learning-Based Dense RGB-D Data Acquisition With Sensor Fusion and 3-D Perception SoC," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, pp. 177-188, Jan. 2023.

[12] J. Jung, S. Kim, B. Seo, W. Jang, S. Lee, J. Shin, D. Han and K. J. Lee, "An Energy-Efficient Processor for Real-Time Semantic LiDAR SLAM in Mobile Robots," *IEEE Journal of Solid-State Circuits*, early access, Sep. 06, 2024, doi: 10.1109/JSSC.2024.3450314.

[13] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc.of International Conference on Neural Information Processing Systems (NIPS),* 2019, pp. 8026–8037.

[14] NVIDIA. (2080). GeForce RTX 2080 Ti. Accessed: Nov, 7, 2024. [Online]. Available: https://www.nvidia.com/en-us/geforce/20-series/

Accuracy Is Not Always We Need: Precision-Aware Bayesian Yield Optimization

Jing Kou*, Zidong Chen†, Liang Zhang*, Haiyan Qin*, Wang Kang*§, Wei W. Xing†§
*National Key Laboratory of Spintronics, Hangzhou International Innovation Institute;
School of Integrated Circuit Science and Engineering, Beihang University, China
†School of Mathematical and Physical Sciences, University of Sheffield, Sheffield, United Kingdom
*{koujing, zhangliang24, haiyanq, wang.kang}@buaa.edu.cn, †{zchen223, w.xing}@sheffield.ac.uk

***Abstract*—Integrated circuit yield optimization plays a vital role in ensuring reliable semiconductor manufacturing, directly impacting both product quality and production costs. Current approaches to yield optimization face two fundamental challenges that limit their practical effectiveness. First, yield estimation requires intensive computational resources. Second, traditional black-box optimization methods inefficiently allocate these resources across design candidates. Most existing approaches compound these issues by performing detailed yield estimations uniformly across all candidates, regardless of their potential quality. To address these limitations, we introduce a novel precision-aware yield optimization framework that intelligently adapts computational resource allocation based on each design candidate's predicted performance. Our approach moves beyond simple simulation counting by incorporating a Figure of Merit (FoM) as a continuous quality metric. By combining a Continuous AutoRegression model to characterize the relationship between true yield and precision levels with a sophisticated multi-fidelity acquisition strategy, our framework achieves optimal resource distribution. Experimental validation on four industry-standard benchmark circuits demonstrates that our method converges with fewer than 1,000 simulations, reducing simulation costs by over 10× while achieving better final designs and robustness than state-of-the-art high-fidelity approaches.**

***Index Terms*—Yield Optimization, Yield Estimation, Multi-fidelity Bayesian Optimization, Continuous Autoregression**

I. Introduction

As integrated circuit technology scales down toward the physical limits of devices, the impact of process variations on chip performance becomes increasingly severe, *e.g.*, doping fluctuation, intra-die mismatches, and threshold voltage variation. These variations can cause circuit performance to deviate from nominal design values, potentially failing to meet required specifications, which is particularly pronounced in analog and mixed-signal CMOS circuits. Therefore, designers must not only optimize nominal circuit performance but also ensure robustness against process variations, thereby giving rise to the yield optimization problem.

The complexity of yield optimization stems from two fundamental challenges. First, yield estimation requires extensive computational resources through circuit simulation. Second, the yield function's black-box nature prevents the application of efficient gradient-based optimization methods. Traditional yield optimization frameworks address these challenges by combining two key components: an estimator that evaluates circuit yield with reduced simulation cost, and an optimizer that performs efficient black-box optimization. This creates an iterative process where the optimizer selects design parameters based on prior estimation results, followed by yield estimation of the selected designs.

Recent research has primarily focused on developing accurate yet efficient yield estimation methods. Despite significant advances, Monte-Carlo (MC) simulation remains the gold standard for reliability, though its computational demands are substantial. For example, verifying a 99.9999% yield requirement for a 6T-SRAM cell necessitates at least 10^7 simulations—a computational burden that proves impractical for real-world applications.

Two main approaches have emerged to reduce yield estimation costs: importance sampling and surrogate modeling. Importance sampling enhances estimation efficiency by concentrating samples in failure regions, as demonstrated by methods such as Minimal Norm Importance Sampling [1], Hyperspherical Clustering and Sampling [2], Adaptive Importance Sampling [3], Optimal Manifold Importance Sampling [4], and Variational Importance Sampling [5]. Alternatively, surrogate modeling employs data-driven machine learning techniques to approximate SPICE simulation behavior, significantly reducing computational overhead. Notable approaches include Gaussian Process (GP) [6], [7], Conditional Normalizing Flow (CNF) [8], low-rank tensor approximation [9], Bayesian neural network [10], and radial basis function network [11].

Once the estimated yield is obtained, various black-box optimization algorithms can be used as yield optimizers, with Bayesian optimization (BO) being most widely used [6], [8], [10], [12]–[14]. However, a critical yet often overlooked observation is that the ultimate goal of yield optimization differs from that of accurate yield estimation—we primarily need to understand yield trends rather than exact values. This insight reveals that many existing methods are overdesigned, expending unnecessary computational resources on precise yield estimates for suboptimal or trend-irrelevant designs.

Several studies have attempted to address this inefficiency. [15] introduced an optimal computing budget allocation (OCBA) algorithm that adjusts resources based on MC variance. [12] and [13] employed two-stage MC estimators, allocating more simulations to promising designs. [16] developed a multi-fidelity GP approach for SRAM arrays, while [17] proposed varying-sigma sampling with multi-fidelity modeling. However, these approaches share several limitations: they often rely on estimation variance without considering its relative impact on yield, implement coarse precision control

§ Denotes Corresponding Authors.

This work was supported by the Natural Science Foundation of China (62274008), Beijing MSTC Program (Z231100007423019), Beijing Natural Science Foundation (L223004), Beijing Municipal Natural Science Foundation (Z230004) and Key R&D Program of Shandong Province, China (2022CXGC020208).

with limited discrete levels, inadequately capture relationships between precision levels, and frequently target specific circuits or estimators, limiting their scalability.

To address these challenges, we propose Precision-Aware Yield Optimization (PAYO), an efficient framework that conducts yield optimization based on low-precision estimation to reduce computational cost. The novelty of this work includes:

- We develop the first yield optimization framework with precision-aware resource allocation. As shown in Fig. 1, PAYO enables significant reduction in simulation costs through intelligent budget distribution.
- To achieve precise resource allocation, we design a novel precision-based autoregressive model that effectively captures the convergence behavior between true yield and estimation precision.
- The proposed framework is general-purpose and integrates seamlessly with existing sampling-based yield estimation methods through FoM-based precision control.
- We propose an advanced optimization strategy that encodes the quantitative relationship between FoM and simulation cost, balancing information loss and achieving optimal resource allocation for each candidate design.
- Experiments on four benchmark circuits, including a 12nm FinFET In-memory-Computing Vector Multiplier, demonstrate that PAYO reduces simulation costs by more than 10× while achieving better optimization results compared to other state-of-the-art (SOTA) methods.

II. Background

A. Problem Formulation: Yield Optimization

Let $\mathbf{x} = [x_1, x_2, \cdots, x_{d_x}]^T \in \mathcal{X}$ denotes design parameters which are controlled by designer, where $\mathcal{X}$ defines the feasible design space. And the inevitable manufacturing process variations are assumed fully captured by $\mathbf{v} = [v_1, v_2, \cdots, v_{d_v}]^T \in \mathcal{V}$. Through normalization, $\mathbf{v}$ can be modeled as a standard independent multivariate Gaussian distribution, where $p(v_i) = \exp\left(-v_i^2/2\right)/\sqrt{2\pi}$. Circuit performance metrics m_k depend on both design and variational parameters through the relationship $m_k = h_k(\mathbf{x}, \mathbf{v})$. A circuit is considered qualified when all K performance metrics meet their specified criteria $\mathbf{m}^0$, *e.g.*, $m_k \leq m_k^0$ for $k = 1, 2, \cdots, K$. We define an identity function $I : \mathcal{X} \times \mathcal{V} \rightarrow \{0, 1\}$ to indicate whether a design meets these criteria. For a given design $\mathbf{x}$, the yield represents the probability that a manufactured circuit will meet all specifications, expressed mathematically as:

$$g(\mathbf{x}) \triangleq \mathbb{E}(I(\mathbf{x}, \mathbf{v})) = \int_{\mathcal{V}} I(\mathbf{x}, \mathbf{v}) p(\mathbf{v}) d\mathbf{v} \tag{1}$$

The yield optimization problem aims to find the optimal design parameters $\mathbf{x}^*$ that maximize yield:

$$\mathbf{x}^* = \underset{x \in \mathcal{X}}{\operatorname{argmax}}\, g(\mathbf{x}) \tag{2}$$

This optimization presents two fundamental challenges: the yield function Eq. (1) requires extensive simulation for estimation, and the lack of gradient information $\nabla_{\mathbf{x}} g(x)$ prevents the use of efficient gradient-based optimization methods.

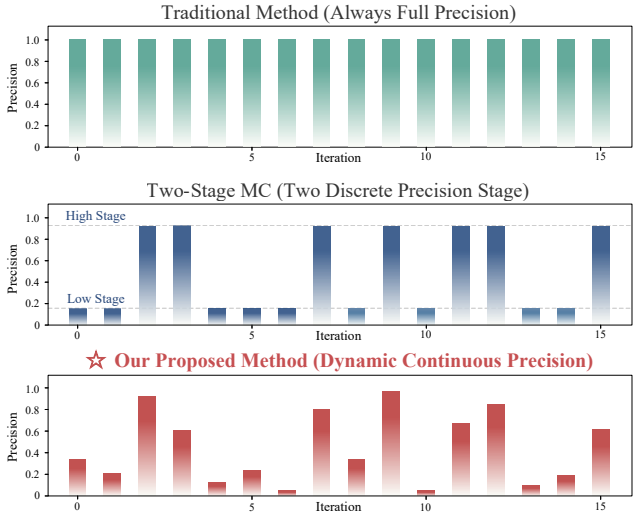


Fig. 1. Illustration of precision allocation among traditional method, Two-stage MC, and our proposed method.

B. Bayesian Yield Optimization

Bayesian optimization provides an efficient framework for optimizing $g(\mathbf{x})$ under these constraints. Since yield estimation relies on random sampling, each estimate contains noise, giving observations of the form $y_i = g(\mathbf{x}_i) + \epsilon_i$, where ϵ_i represents estimation noise. To enable optimization without gradient information, we model $g(\mathbf{x})$ using a GP surrogate model that updates throughout the optimization process. The GP places a prior distribution $g(\mathbf{x})|\boldsymbol{\theta} \sim \mathcal{GP}\left(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta})\right)$ with mean function $m(\mathbf{x}) = \mathbb{E}[g(\mathbf{x})]$ and covariance function $k(\mathbf{x}, \mathbf{x}'|\boldsymbol{\theta}) = \mathbb{E}[(g(\mathbf{x}) - m(\mathbf{x}))(g(\mathbf{x}') - m(\mathbf{x}'))]$. The hyperparameters $\boldsymbol{\theta}$ define the kernel function and are estimated at each iteration using maximum likelihood estimation. The GP model provides predictive mean, $\mu(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T \left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1} \mathbf{y}$ and variance $v(\mathbf{x}) = \sigma^2 + k(\mathbf{x}, \mathbf{x}) - \mathbf{k}^T(\mathbf{x}) \left(\mathbf{K} + \sigma^2\mathbf{I}\right)^{-1} \mathbf{k}(\mathbf{x})$. Given the high computational cost of yield estimation, selecting the next design point $\mathbf{x}_t$ for evaluation requires careful balance between exploiting regions of high observed yield and exploring uncertain regions. This balance is achieved through an acquisition function, with Upper Confidence Bound (UCB) being particularly effective:

$$\mathbf{x}_t = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}} \left(\mu(\mathbf{x}) + \beta^{\frac{1}{2}} v(\mathbf{x})\right) \tag{3}$$

where β controls the exploration-exploitation trade-off.

C. Multi-Fidelity Bayesian Optimization

Multi-fidelity Bayesian optimization (MFBO) extends traditional BO by leveraging evaluations of varying computational cost and accuracy. In yield optimization, higher fidelity evaluations require more simulations, providing more accurate yield estimates but at greater computational cost. Let function $f(\mathbf{x}, z)$ represent the estimated yield for design $\mathbf{x}$ at fidelity $z \in [z^0, z^1]$, where z^0 denotes the lowest fidelity (fewer simulations) and z^1 the highest fidelity (maximum simulations). The MFBO framework can be formally described as follows:

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}}\, f(\mathbf{x}, z^1) \text{ s.t. } \sum_{t=1}^{T} c(z_t) \leq B \tag{4}$$

where $c(z_t)$ represents the computational cost at fidelity level z_t (assuming independent of $\mathbf{x}$), T is the total number of

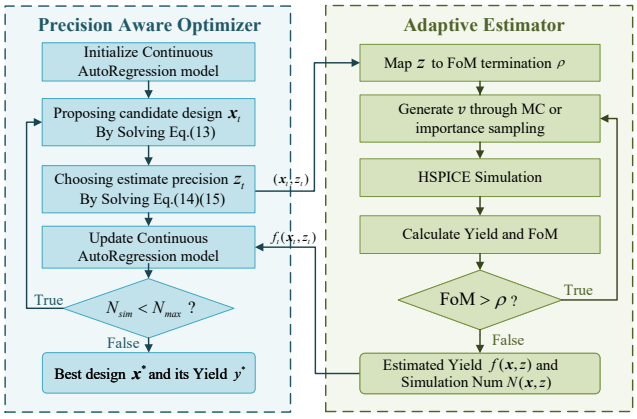


Fig. 2. The workflow of our proposed Precision-Aware Yield Optimization.

evaluations, and B is the total computational budget. Generally speaking, at each iteration, MFBO selects both the design point and fidelity level by maximizing a multi-fidelity acquisition function:

$$(\mathbf{x}_t, z_t) = \underset{\mathbf{x}\in\mathcal{X}, z\in\mathcal{Z}}{\operatorname{argmax}}\ \alpha(\mathbf{x}, z)/c(z) \tag{5}$$

where $\alpha(\mathbf{x}, z)$ is the acquisition function value and $c(z)$ represents the evaluation cost at fidelity z. Most existing MFBO frameworks focus on discrete fidelity levels due to their simpler implementation. However, this discretization can limit the efficiency of resource allocation, particularly in yield optimization where the relationship between simulation cost and estimation accuracy is inherently continuous. This limitation motivates our development of a continuous fidelity approach specifically designed for yield optimization.

III. Proposed Approach

To address the limitations of existing yield optimization approaches, we propose PAYO (Precision-Aware Yield Optimization), a framework that intelligently allocates computational resources by dynamically adjusting yield estimation precision. Our framework consists of three key components: (1) an adaptive yield estimator that provides continuous precision control through Figure of Merit (FoM), (2) a continuous autoregressive model that captures the relationship between different precision levels, and (3) a precision-aware optimization strategy that efficiently balances exploration and exploitation across the precision spectrum.

A. Framework Overview

Fig. 2 illustrates the PAYO workflow: In a single iteration, the precision-aware optimizer first selects a promising design candidate $\mathbf{x}_t$ and its ideal precision level z_t with potential maximum gain. Adaptive yield estimation is then conducted accordingly. All data are collected to train a continuous autoregressive model to build a relation between truth yield and low-precision yield analysis. With these three components integrated, yield analyses are conducted at a "just fine" precision to reduce computational cost in yield optimization.

B. Adaptive Yield Estimator with Continuous Precision

In precision-aware yield optimization, a critical challenge is mapping the desired precision level z_t to an appropriate simulation count N. Classic approaches often employ simple

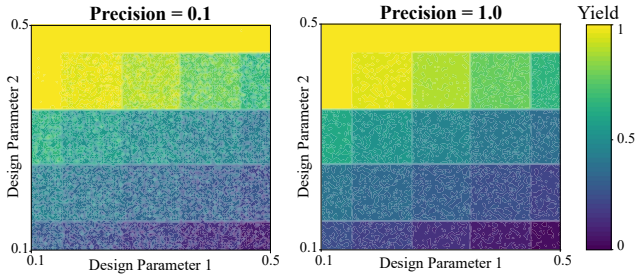


Fig. 3. Yield distribution for OTA circuit: Low precision estimation does not significantly influence the yield trend and is sufficient to guide optimization.

strategies which proved to be inadequate in practice. The most straightforward approach uses linear mapping: $N = N_{\max} z_t$, where $N_{\max}$ represents the maximum simulation count per iteration. However, this naive approach fails to account for the varying difficulty in estimating yields of different magnitudes. For instance, accurately estimating an extremely high yield (*i.e.*, very low failure rate P_f) requires substantially more simulations than estimating a moderate yield value. Several works [12], [13], [15] propose variance-based resource allocation using the approximation:

$$N \approx \frac{Y \cdot (1-Y) \cdot k_\gamma^2}{\Delta Y^2} \tag{6}$$

where ΔY defines the confidence interval and k_γ reflects the confidence level. However, this approach suffers from two fundamental limitations: (1) Its derivation assumes the binomial distribution can be approximated by a normal distribution when N is large enough—an invalid assumption for the typically small sample sizes used in yield optimization. (2) It treats estimation variance uniformly across different yield levels, ignoring that the same variance has drastically different implications for different failure rates (e.g., a variance of 0.1 is far more significant for $P_f = 10^{-8}$ than for $P_f = 0.5$).

To address these limitations, we adopt FoM as a more effective precision measure. For Monte Carlo methods, FoM quantifies the relative error in yield estimation:

$$\rho = \frac{\sqrt{Var(\hat{P}_f)}}{\hat{P}_f} = \sqrt{\frac{1}{N} \cdot \frac{1-\hat{P}_f}{\hat{P}_f}} \tag{7}$$

For importance sampling methods, FoM is defined as:

$$\rho = \frac{\sqrt{Var(\hat{P}_f)}}{\hat{P}_f} = \frac{1}{N} \cdot \sqrt{\frac{\sum_{i=1}^{N} w(\mathbf{v}_i)^2 I(\mathbf{x}, \mathbf{v})}{\hat{P}_f^2} - N} \tag{8}$$

where $\hat{P}_f$ represents the estimated failure rate and $w(\mathbf{v}_i)$ denotes importance sampling weights. Building on this FoM foundation, we propose an adaptive yield estimator that dynamically controls precision through continuous FoM monitoring. During estimation, we compute the current FoM after each sampling round and continue sampling until FoM drops below $1 - 0.9z_t$. This approach provides several key advantages:

Natural Resource Allocation: As evident from Eq. (7) and Eq. (8), designs with lower failure rates automatically receive more simulation resources, aligning with the need for higher precision during exploitation phases.

Efficient Low-Precision Evaluation: The quadratic relationship between FoM and simulation count means that low-

precision evaluations require minimal resources. As demonstrated in Fig. 3, these evaluations can still effectively capture yield trends, making them ideal for exploration phases.

General Applicability: The estimator seamlessly integrates with both MC and importance sampling methods through their respective FoM definitions, providing a unified framework for precision control.

This adaptive approach ensures that computational resources are allocated based on both the desired precision level and the inherent difficulty of yield estimation for each design point, enabling more efficient optimization compared to traditional fixed-allocation strategies.

C. Continuous AutoRegression for Precision Modeling

Traditional multi-fidelity yield optimization approaches [16], [17] typically treat fidelity as an independent input dimension in their GP models. While straightforward to implement, this approach fails to capture the inherent convergence behavior of yield estimation—as precision increases, estimates gradually converge to their true values. Furthermore, treating precision levels independently ignores the strong correlations between successive precision levels, leading to inefficient information utilization during optimization.

To address these limitations, we employ a Continuous AutoRegression model [18] with a specially designed kernel structure that captures the precision-dependent nature of yield estimation. Our key insight is that yield estimates at different precision levels exhibit a continuous evolution pattern that can be modeled through differential equations. We formulate this relationship using a differential equation:

$$\frac{\partial f(\mathbf{x}, z)}{\partial z} = \alpha(z) f(\mathbf{x}, z) + s(\mathbf{x}, z) \tag{9}$$

where $f(\mathbf{x}, z)$ represents the yield estimate at precision z, $\alpha(z)$ controls the convergence rate, and $s(\mathbf{x}, z)$ captures additional nonlinear relationships. By setting $\alpha(z)$ to a positive constant, we ensure the model reflects the natural convergence behavior of yield estimation. The solution to this differential equation is:

$$f(\mathbf{x}, z) = f(\mathbf{x}, z^0) e^{-\alpha(z - z^0)} + \int_{z^0}^{z} e^{-\alpha(z-\tau)} s(\mathbf{x}, \tau)\, d\tau \tag{10}$$

where z^0 represents the lowest precision level. We place GP priors on both the base function and the residual term: $f(\mathbf{x}, z^0) \sim \mathcal{GP}\big(0,\ k_0(x, x')\big)$ and $s(\mathbf{x}, z) \sim \mathcal{GP}\left(0,\ k_s((x, z), (x', z'))\right)$. This formulation leads to a precision-aware kernel structure:

$$\begin{aligned} &k\big((\mathbf{x}, z), (\mathbf{x}', z')\big) = e^{-\alpha(z - z^0)} k_0(\mathbf{x}, \mathbf{x}') \\ &+ \int_{z^0}^{z} e^{-\alpha(z-\tau)} \int_{z^0}^{z'} e^{-\alpha(z'-\tau')} k_s\big((\mathbf{x}, \tau), (\mathbf{x}', \tau)\big), d\tau, d\tau', \end{aligned} \tag{11}$$

which explicitly models the convergence behavior through the exponential terms, captures both design-space relationships (k_0) and precision-level correlations (k_s), and enables efficient information transfer across precision levels through the continuous formulation.

The combination of continuous autoregression and precision-aware kernel structure allows our model to better capture the yield estimation behavior across different precision levels, leading to more efficient resource allocation and optimization convergence.

D. Precision-Aware Bayesian Yield Optimization

With the autoregressive model capturing the relationship between different precision levels, we can now leverage this information for efficient yield optimization. Unlike traditional GP models that treat fidelity as an independent dimension, our autoregressive model provides rich information about the convergence behavior of yield estimates, enabling more informed decisions about precision selection. Multi-fidelity Bayesian optimization typically selects both design parameters and fidelity levels by maximizing an acquisition function augmented with fidelity:

$$\mathbf{x}_t, z_t = \underset{\mathbf{x} \in \mathcal{X}, z \in \mathcal{Z}}{\operatorname{argmax}} \Big(\mu(\mathbf{x}, z) + \beta(z)^{\frac{1}{2}} v(\mathbf{x}, z) \Big) \tag{12}$$

However, this approach has limitations for yield optimization: it doesn't account for the unique characteristics of yield estimation precision, and it may waste resources by evaluating designs at unnecessarily high precision levels. To address these limitations, we propose a novel optimization strategy that leverages our autoregressive model and FoM-based precision control inspired by [19]. First, we select promising designs based on their potential at full precision:

$$\mathbf{x}_t = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmax}} \Big(\mu(\mathbf{x}, z^1) + \beta_t^{\frac{1}{2}} v(\mathbf{x}, z^1) \Big) \tag{13}$$

where β_t increases with iterations to maintain exploration. Then, we determine the appropriate precision level through a novel selection strategy:

$$z_t = \underset{z \in \mathcal{Z}_t(\mathbf{x}_t) \cup z^1}{\operatorname{argmin}} N(\mathbf{x}_t, z) \tag{14}$$

subject to precision constraints:

$$\begin{aligned} \mathcal{Z}_t(\mathbf{x}_t) = \{ z \in \mathcal{Z} \setminus \{z^1\} : v(\mathbf{x}_t, z) > \sqrt{\kappa_0}\ \xi(z) \left(\frac{1 - 0.9 z^1}{1 - 0.9 z} \right)^{2q}, \\ \xi(z) > \beta_t^{-1/2} \|\xi\|_\infty \} \end{aligned} \tag{15}$$

where κ_0 is the length scale of kernel function. q is a constant dependent on kernel used for z, *e.g.*, for the squared exponential kernel, $q = 1/(d_x + 3)$.

This strategy incorporates several key innovations:

Information-Cost Balance: The first constraint balances information gain (measured by $\xi(z) = \sqrt{1 - \phi_z(z, z^*)^2}$ where ϕ_z is the kernel function of z) against computational cost, leveraging the Continuous AutoRegression model's understanding of precision relationships. Furthermore, the quadratic relationship between FoM and simulation count is taken into account to make more reasonable precision selection.

Efficiency Threshold: The second constraint prevents the selection of unnecessarily high precision levels by setting an exclusion zone near full precision.

Adaptive Resource Allocation: The framework automatically adjusts precision based on: (1) exploration needs (lower precision for initial search) (2) exploitation requirements (higher precision for promising regions) (3) convergence behavior captured by the Continuous AutoRegression model.

As shown in Algorithm 1, the complete PAYO algorithm integrates these components, providing a comprehensive framework for precision-aware yield optimization.

TABLE I
YIELD OPTIMIZATION FOR DIGITAL CIRCUITS WITH DIFFERENT NUMBERS OF TOTAL SIMULATIONS

		# Simulation=1,000				# Simulation=5,000				# Simulation=10,000			
Circuit	Method	Worst	Best	Mean	Std	Worst	Best	Mean	Std	Worst	Best	Mean	Std
SRAM (Fail rate)	AYO	4.14e-1	4.13e-1	4.13e-1	8.28e-5	2.50e-7	5.00e-8	1.00e-7	8.66e-8	1.50e-7	5.00e-8	8.00e-8	4.47e-8
	KDEBO	4.11e-1	3.77e-2	2.83e-1	1.49e-1	3.93e-1	4.73e-6	1.25e-1	1.58e-1	4.95e-6	1.92e-8	2.12e-6	2.49e-6
	MVEBO	4.14e-1	2.73e-3	3.31e-1	1.84e-1	4.13e-1	2.94e-8	9.31e-2	1.80e-1	2.50e-7	**1.03e-8**	7.04e-8	1.02e-7
	BYABO	4.13e-1	4.13e-1	4.13e-1	1.52e-4	1.85e-6	1.32e-8	1.30e-6	7.50e-7	1.70e-6	1.32e-8	5.43e-7	7.47e-7
	OPT	1.08e-1	1.08e-1	1.08e-1	6.22e-5	**9.30e-8**	**1.05e-8**	4.73e-8	3.49e-8	**9.30e-8**	1.05e-8	4.73e-8	3.49e-8
	PAYO-IS	**9.30e-8**	**1.61e-8**	5.12e-8	3.26e-8	**9.30e-8**	1.61e-8	5.12e-8	3.26e-8	**9.30e-8**	1.61e-8	5.12e-8	3.26e-8
	PAYO-MC	**9.30e-8**	**1.61e-8**	**4.52e-8**	**2.94e-8**	**9.30e-8**	1.61e-8	**4.52e-8**	**2.94e-8**	**9.30e-8**	1.61e-8	**4.52e-8**	**2.94e-8**
ADDER (Fail rate)	AYO	5.68e-3	5.63e-3	5.66e-3	1.65e-5	1.05e-7	2.94e-8	7.39e-8	2.99e-8	1.05e-7	1.92e-8	4.62e-8	3.46e-8
	KDEBO	8.20e-2	4.76e-8	4.94e-2	4.45e-2	5.88e-8	2.56e-8	4.52e-8	1.44e-8	5.88e-8	2.50e-8	3.85e-8	1.44e-8
	MVEBO	5.68e-3	5.63e-3	5.66e-3	1.69e-5	9.30e-8	1.56e-8	5.78e-8	3.12e-8	9.30e-8	1.56e-8	3.68e-8	3.19e-8
	BYABO	5.68e-3	5.64e-3	5.66e-3	1.76e-5	8.70e-8	2.94e-8	4.46e-8	2.50e-8	2.94e-8	1.75e-8	2.47e-8	6.42e-9
	OPT	7.15e-2	7.13e-2	7.14e-2	6.06e-5	3.49e-8	1.82e-8	2.84e-8	**6.13e-9**	3.49e-8	1.82e-8	2.84e-8	**6.13e-9**
	PAYO-IS	7.14e-8	1.82e-8	4.02e-8	1.92e-8	7.14e-8	1.82e-8	4.02e-8	1.92e-8	7.14e-8	1.82e-8	4.02e-8	1.92e-8
	PAYO-MC	**2.94e-8**	**8.70e-9**	**2.30e-8**	**8.37e-9**	**2.94e-8**	**8.70e-9**	**2.30e-8**	8.37e-9	**2.94e-8**	**8.70e-9**	**2.30e-8**	8.37e-9

Algorithm 1 Precision-Aware Bayesian Yield Optimization

Input: Adaptive yield estimator $f(\mathbf{x}, z)$, # Initial samples N_i, # BO iteration N_o

Sample $\{\mathbf{x}_i, z_i\}, i = 1, \cdots N_i$ to estimate the yield y_i
Set the initial observation set $\mathcal{D}_\prime \leftarrow \{y_i, \mathbf{x}_i, z_i\}, i = 1, \cdots N_i$
Initialize Continuous AutoRegression model with initial $\mathcal{D}$
for $t = 1$ to N_o **do**
  Propose candidate design $\mathbf{x}_t$ by solving Eq. (13)
  Select precision z_t by solving Eq. (14) and Eq. (15)
  Set initial FoM $\rho = \infty$
  while $\rho > 1 - 0.9z_t$ **do**
    Randomly sample variation parameters $\mathbf{v}$
    Obtain performance metric by SPICE simulation
    Update the yield y_t and FoM ρ
  end while
  $\mathcal{D}_t \leftarrow \mathcal{D}_{t-1} \cup \{y_t, \mathbf{x}_t, z_t\}$
  Update the Continuous AutoRegression model
end for
return Best design parameter $\mathbf{x}^*$

IV. EXPERIMENTAL RESULTS

To evaluate the efficiency and optimization performance of our proposed framework, we conducted experiments on four benchmark circuits: a 6T-SRAM, an Adder, an Operational Transconductance Amplifier (OTA), and an In-Memory-Computing Vector Multiplier (IMC-VM). These benchmark circuits encompass both digital and analog designs, as well as advanced and traditional process nodes, allowing for a comprehensive assessment of PAYO's scalability and robustness across different circuit paradigms and technology nodes. To ensure rigorous evaluation, we implement five SOTA methods as comparative baselines: AYO [12], KDEBO [14], MVEBO [13], BYABO [6], and OPT [8]. To eliminate the influence of randomness, each method was tested using five random seeds and the optimal design was validated using 10e7 MC simulations (IMC-VM using 10e3). All experiments are conducted on a Linux system equipped with an Intel Xeon Platinum 8352V CPU, and 60GB of memory.

A. Yield Optimization on Digital Circuit

The performance of our proposed framework in digital circuit yield optimization was validated using two circuits: 6T-SRAM and Adder, both of which are implemented using 45nm CMOS process. Fig. 4 (a) shows the schematic of the 6T-SRAM, which consists of a latch that can stably store 1bit data and two access transistors for read/write operation. We consider the read/write delay as the performance metric and the widths of the three symmetric transistor couples are chosen as the design parameters of interest. As depicted in Fig. 4 (b), Adder circuit consists of 28 transistors, the widths and lengths of transistors are chosen as the design parameters. We compute the yield by the delay of the output reaching a predefined threshold voltage through transient simulations.

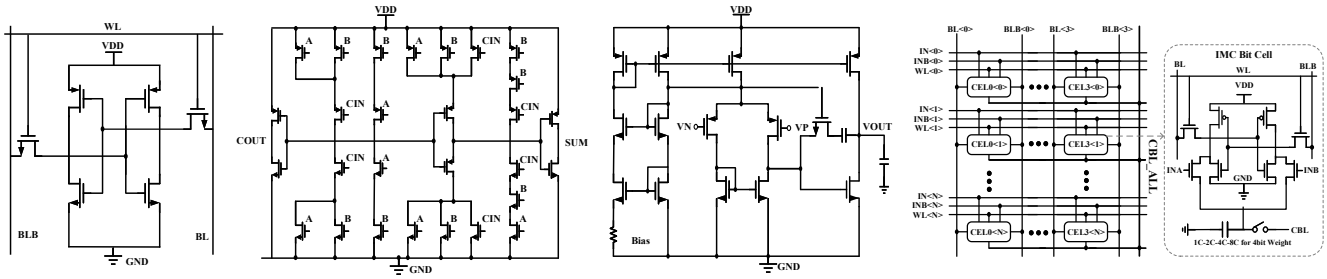
As shown in Table. I, it is evident that PAYO achieved the best optimization results with the minimal simulation cost. For 6T-SRAM, PAYO-MC achieves a performance improvement of 2.39e6×–9.14e6× with 1,000 simulations, 1.05×–2.77e6× improvement with 5,000 simulations, and 1.05×–46.90× improvement with 10,000 simulations. For Adder, PAYO-MC achieves 2.46e5×–3.10e6× improvement with 1,000 simulations, 1.23×–3.21× improvement with 5,000 simulations, and 1.07×–2.01× improvement with 10,000 simulations. To verify the compatibility of PAYO with importance sampling methods, we embedded Minimal Norm Importance Sampling (MNIS) [1] into the PAYO framework for experimentation (PAYO-IS). Due to the lower accuracy and reliability of MNIS compared to MC, PAYO-IS performs slightly worse than PAYO-MC. However, it still achieves better performance than other baselines.

B. Yield Optimization on Analog circuit

To validate the effectiveness of our proposed method for analog circuits, we conducted experiment on the OTA and IMC-VM circuit. The OTA, implemented in 180nm CMOS proces, consists of a bias circuit and a two-stage amplifier with miller compensation, as shown in Fig. 4 (c). The OTA's static current is considered as the target performance metric

TABLE II
YIELD OPTIMIZATION FOR ANALOG CIRCUITS WITH DIFFERENT NUMBERS OF TOTAL SIMULATIONS

		# Simulation=1,000				# Simulation=5,000				# Simulation=10,000			
Circuit	Method	Worst	Best	Mean	Std	Worst	Best	Mean	Std	Worst	Best	Mean	Std
OTA (Fail rate)	AYO	5.29e-1	5.29e-1	5.29e-1	1.00e-4	1.46e-2	8.82e-8	8.77e-3	8.00e-3	8.82e-8	**1.00e-8**	3.56e-8	3.05e-8
	KDEBO	5.82e-1	1.46e-2	3.18e-1	2.18e-1	3.99e-1	1.33e-5	1.22e-1	1.76e-1	5.88e-8	1.75e-8	3.47e-8	1.84e-8
	MVEBO	1.46e-2	7.14e-8	4.02e-3	6.08e-3	6.52e-8	1.35e-8	3.70e-8	1.99e-8	6.52e-8	1.25e-8	3.13e-8	2.15e-8
	BYABO	5.29e-1	5.29e-1	5.29e-1	4.55e-5	1.40e-5	2.00e-8	7.96e-6	7.24e-6	8.82e-8	1.89e-8	3.51e-8	2.99e-8
	PAYO-MC	**4.08e-8**	**1.15e-8**	**2.55e-8**	**1.18e-8**	**4.08e-8**	**1.15e-8**	**2.55e-8**	**1.18e-8**	**4.08e-8**	1.15e-8	**2.55e-8**	**1.18e-8**
IMC-VM (Yield)	AYO	93.9%	94.3%	94.2%	0.15%	95.4%	**99.7%**	98.0%	1.57%	**99.1%**	**99.7%**	99.3%	**0.20%**
	KDEBO	77.4%	99.2%	91.9%	7.69%	94.2%	99.6%	97.4%	1.99%	97.6%	**99.7%**	99.0%	0.76%
	MVEBO	94.2%	94.4%	94.3%	**0.08%**	94.3%	99.6%	98.0%	1.92%	98.7%	**99.7%**	99.3%	0.35%
	BYABO	82.1%	99.4%	92.1%	6.73%	96.7%	99.4%	98.5%	0.96%	98.1%	99.5%	99.0%	0.53%
	PAYO-MC	**97.6%**	**99.7%**	**98.8%**	0.72%	**98.9%**	**99.7%**	**99.4%**	**0.28%**	98.9%	**99.7%**	**99.4%**	0.28%



(a) 6T-SRAM (b) Adder (c) OTA (d) In-memory-Computing Vector Multiplier

Fig. 4. Simplified schematic for benchmark circuits: 6T-SRAM, Adder, OTA amd IMC Vector Multiplier

and the widths of current mirrors and bias transistors are selected as the design parameters for yield optimization. Fig. 4 (d) shows the structure of the IMC-VM proposed by [20], where each input is multiplied with the 4bit weight stored in the 9T-1C IMC cell and accumulated on the global charge bit line through charge sharing. Multibit-weight modulation is conducted through capacitance difference. Sum of each node's settling time of is selected as the performance metric. As the circuit is implemented in 12nm FinFET process, the number of fins and fingers of each FinFET in the 9T-1C cell is considered as the design parameter of interest.

Due to the poor robustness of OPT and MNIS in complex analog circuit yield analysis, We did not include OPT and PAYO-IS in the comparison. As shown in Table. II, our method also demonstrates the best performance among all baselines on the analog circuits. For the OTA, PAYO-MC achieved performance improvements of 1.58e5×–2.07e7× with 1000 simulations, 1.45×–4.78e6× with 5000 simulations, and 1.23×–1.40× with 10000 simulations. For the IMC Vector Multiplier, PAYO-MC achieved performance improvements of 4.77%–7.51%, 0.91%–2.05%, and 0.10%–0.40% with 1000, 5000, and 10000 simulations, respectively.

C. Ablation Study

To validate the effectiveness of each component in PAYO, we conducted ablation experiments on the Adder circuit. As shown in Fig. 5, we compared the number of simulations required by each method to first find a design with a failure rate less than 1e-7. Each method was set with $N_{\max} = 2000$. Compared to full precision, linear mapping $N = N_{\max} z_t$, and Two-Stage MC, our proposed FoM mapping method achieved improvement factors of 11.28×, 9.99×, and 4.54×, respectively. Furthermore, compared to GP regression (GPR), the Continuous AutoRegression (CAR) model we used improved optimization efficiency by 1.36×.

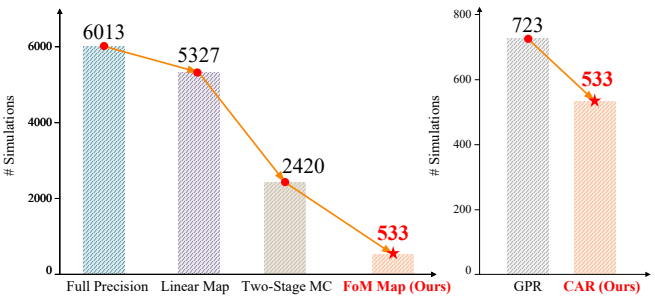


Fig. 5. Ablation study of each component.

V. CONCLUSION

This paper presents PAYO, the first precision-aware yield optimization framework that significantly reduces simulation costs (more than 10× reduction). Key innovations include FoM-based precision control, continuous autoregressive modeling for precision relationships, and an efficient resource allocation strategy. Current limitations include computational overhead from the autoregression model and the need for initial samples across different precision levels. Future work will explore extensions to multi-objective yield optimization and online learning strategies for more efficient precision control.

REFERENCES

[1] L. Dolecek, M. Qazi, D. Shah, and A. Chandrakasan, "Breaking the simulation barrier: Sram evaluation through norm minimization," in *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2008, pp. 322–329.

[2] W. Wu, S. Bodapati, and L. He, "Hyperspherical clustering and sampling for rare event analysis with multiple failure region coverage," in *Proceedings of the 2016 on International Symposium on Physical Design*, 2016, pp. 153–160.

[3] X. Shi, H. Yan, J. Wang, X. Xu, F. Liu, L. Shi, and L. He, "Adaptive clustering and sampling for high-dimensional and multi-failure-region sram yield analysis," in *Proceedings of the 2019 International Symposium on Physical Design*, 2019, pp. 139–146.

[4] Y. Liu, G. Dai, and W. W. Xing, "Seeking the yield barrier: High-dimensional sram evaluation through optimal manifold," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[5] Y. Liu, L. He, and W. W. Xing, "Beyond the yield barrier: Variational importance sampling yield analysis," *arXiv preprint arXiv:2407.00711*, 2024.

[6] S. Yin, X. Jin, L. Shi, K. Wang, and W. W. Xing, "Efficient bayesian yield analysis and optimization with active learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1195–1200.

[7] S. Yin, G. Dai, and W. W. Xing, "High-dimensional yield estimation using shrinkage deep features and maximization of integral entropy reduction," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 283–289.

[8] Y. Liu, G. Dai, Y. Cheng, W. Kang, and W. W. Xing, "Opt: Optimal proposal transfer for efficient yield optimization for analog and sram circuits," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[9] X. Shi, H. Yan, Q. Huang, J. Zhang, L. Shi, and L. He, "Meta-model based high-dimensional yield analysis using low-rank tensor approximation," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[10] N. Guo, F. Peng, J. Shi, F. Yang, J. Tao, and X. Zeng, "Yield optimization for analog circuits over multiple corners via bayesian neural networks: Enhancing circuit reliability under environmental variation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 1, pp. 1–17, 2023.

[11] J. Yao, Z. Ye, and Y. Wang, "An efficient sram yield analysis and optimization method with adaptive online surrogate modeling," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 7, pp. 1245–1253, 2014.

[12] M. Wang, W. Lv, F. Yang, C. Yan, W. Cai, D. Zhou, and X. Zeng, "Efficient yield optimization for analog and sram circuits via gaussian process regression and adaptive yield estimation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 10, pp. 1929–1942, 2017.

[13] S. Zhang, F. Yang, D. Zhou, and X. Zeng, "Bayesian methods for the yield optimization of analog and sram circuits," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 440–445.

[14] D. D. Weller, M. Hefenbrock, M. Beigl, and M. B. Tahoori, "Fast and efficient high-sigma yield analysis and optimization using kernel density estimation on a bayesian optimized failure rate model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 695–708, 2021.

[15] I. Guerra-Gomez, E. Tlelo-Cuautle, and L. G. de la Fraga, "Ocba in the yield optimization of analog integrated circuits by evolutionary algorithms," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2015, pp. 1933–1936.

[16] X. Wang, T. Gu, C. Yan, X. Wu, F. Yang, S.-G. Wang, D. Zhou, and X. Zeng, "An efficient and robust yield optimization method for high-dimensional sram circuits," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[17] X. Wang, C. Yan, F. Yang, D. Zhou, and X. Zeng, "An efficient yield optimization method for analog circuits via gaussian process classification and varying-sigma sampling," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 625–630.

[18] W. Xing, Y. Wang, and Z. Xing, "Continuar: continuous autoregression for infinite-fidelity fusion," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[19] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos, "Multi-fidelity bayesian optimisation with continuous approximations," in *International conference on machine learning*. PMLR, 2017, pp. 1799–1808.

[20] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, 2019.

Towards In-Situ Neuromorphic Computing Architecture for Event Stream Super-Resolution

Yihe Yu*, Bo Li*, Kexin Huang, Wei Liu, Jinghai Wang, Yue Liu, Zhiyi Yu, Shanlin Xiao†
Sun Yat-sen University, China
{yuyh25, libo93, huangkx36, wangjh335, liuy2389}@mail2.sysu.edu.cn; {liuw359,yuzhiyi, xiaoshlin}@mail.sysu.edu.cn

***Abstract*—Event-based cameras, with their unique event stream representation, effectively mitigate motion blur in high-speed, high-exposure scenarios but suffer from low spatial resolution. To address this, we propose a super-resolution hardware accelerator for event streams based on Spiking Neural Networks (SNNs). In terms of network architecture, we incorporate hardware-friendly algorithmic designs by simplifying neuron models and optimizing convolution operations. On the hardware side, the design adopts a hierarchical structure featuring a highly parallel computational array. Additionally, by proposing a Kernel-Channel-Timestamp-Row (KCTR) dataflow and dual-pipeline structure, the design achieves in-situ computing, eliminating intermediate storage within layers and significantly reducing inter-layer spike storage. Evaluations on the N-MNIST and ASL-DVS datasets demonstrate root mean square errors (RMSE) of 1.296 and 0.121 for reconstructed super-resolution event streams. In downstream applications, the classification accuracies reach 98.84% and 99.73%, respectively. The proposed accelerator, designed using a 28nm CMOS process, improves reconstruction speed by 95.6% compared to a GPU, operates at 500 MHz, and consumes only 0.546 pJ per synaptic operation.**

***Index Terms*—Spiking Neural Network, Hardware Accelerator, Super Resolution, Event-based Camera.**

I. Introduction

The event camera is a bio-inspired sensor that fundamentally differs from traditional cameras, which capture full scene frames at fixed intervals. Instead, event cameras, also known as Dynamic Vision Sensors (DVS) [1]–[4], respond asynchronously to brightness changes, outputting data as an event stream. When the light intensity at a pixel changes significantly beyond a predefined threshold, the pixel generates an event, indicating brightness increase or decrease (1 or 0). Unlike traditional cameras, event cameras eliminate frame-based redundancy, producing pixel-level event streams only during scene changes. Fig. 1 compares the imaging performance of two types of cameras for high-speed object. The event camera, which records only events and features higher temporal resolution compared to traditional frame-based cameras, effectively mitigates the issue of motion blur.

Event cameras offer distinct advantages over traditional frame-based cameras. They achieve microsecond-level temporal resolution [5], [6], far surpassing conventional frame rates,

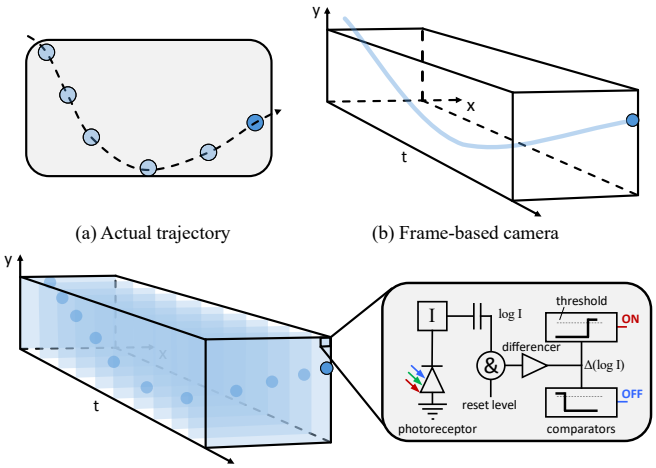


Fig. 1. (a) for the actual trajectory of object motion, (b) (c) illustrate the imaging results of a frame-based camera and an event-based camera for high-speed moving objects, and (c) shows the internal structure of DVS.

and their low latency effectively reduces motion blur. With independently and asynchronously operating pixels, they provide a dynamic range of up to 140 dB, compared to 60 dB in traditional cameras [2]. Additionally, their sparse data output, generated only during brightness changes, eliminates frame-based redundancy, streamlining processing and improving computational efficiency. These features significantly reduce computational and energy demands, making event cameras ideal for edge computing and real-time applications [7].

However, the spatial resolution of current event cameras remains lower than that of traditional cameras. Popular models like the DVS128 (128×128 pixels) [8] and DAVIS240 (240×180 pixels) [1] operate at relatively low resolutions due to their pixel-level event-triggering mechanism, where each pixel independently detects light intensity changes and generates events. As shown in Fig. 1(c), the DVS architecture integrates circuits like photoreceptors, differencers, and threshold comparators within each pixel [9], requiring substantial physical space and limiting resolution. Moreover, event cameras prioritize high-temporal-resolution data for compact edge devices, making it difficult to increase resolution within size constraints. Additionally, manufacturing costs and process limitations further restrict their spatial resolution [10].

Researchers have thus focused on super-resolution reconstruction of low-spatial-resolution images captured by event

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 62334014; in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2023B0303030004.
* These authors contributed equally to this work.
† Corresponding author.

cameras. The event streams generated by event cameras are asynchronous, sparse, and feature high temporal resolution, aligning well with the processing mechanisms of SNNs. As neural networks designed to handle discrete time-stamped data [11], SNNs can simultaneously process spatiotemporal information, making them particularly suitable for receiving and processing discrete event streams from event cameras [12]–[14]. However, current research on super-resolution for event cameras has largely focused on algorithms, with limited emphasis on hardware implementation [15]–[18]. To the best of our knowledge, this work is the first to integrate SNNs with event stream super-resolution within a algorithm-hardware co-design framework. We present an SNN-based hardware accelerator that generates high-resolution (HR) event streams from low-resolution (LR) inputs and adapts to diverse datasets. Our main contributions are as follows:

- We propose a hardware-efficient SNNs algorithm for event stream super-resolution image reconstruction. By simplifying the neuron model and optimizing computation order and channel count to exploit hardware parallelism.
- We propose a high-efficiency SNNs hardware architecture with hierarchical design. With the help of the Kernel-Channel-Timestamp-Row (KCTR) dataflow and a dual-pipeline strategy, in-situ computation is achieved, eliminating intermediate value storage within layers and reducing spike storage between layers.
- The performance of hardware-based super-resolution image reconstruction and the quality of the reconstructed images in downstream applications were systematically evaluated. Results show the proposed method reduces reconstruction error and significantly enhances downstream classification accuracy, confirming the broad potential of super-resolution in practical use.

II. Background

A. Spiking neuron model

Spiking neurons are the core of SNNs [19], with models like the Hodgkin-Huxley [20], Leaky Integrate-and-Fire (LIF) [21], and Izhikevich [22] proposed to mimic biological neurons. The LIF model, favored for its simplicity and low computational cost, is the most widely used but lacks accuracy for complex tasks [23]. To address this, the Spike Response Model (SRM) [24] improves on LIF by incorporating refractory response (REF) decay, boosting accuracy with minimal overhead [25], [26]. Thus, we adopt the SRM in our SNNs design.

Fig. 2 shows the dynamics of an SRM spiking neuron. Input spikes $s_{in}(t)$, weighted by synaptic strengths (indicated by spike lengths), are processed sequentially to modulate the post-synaptic potential (PSP). The membrane potential $u(t)$ (red curve) combines all PSPs and the REF (blue curve). When $u(t)$ exceeds the threshold V_{th}, the neuron emits an output spike $s_{out}(t)$, triggering a REF that temporarily suppresses the membrane potential, preventing premature firing and ensuring proper spike timing.

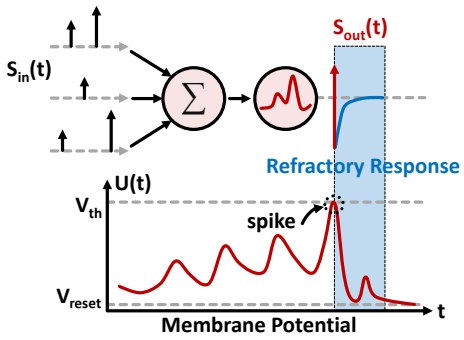


Fig. 2. Membrane potential dynamics of the SRM neuron. The red curve represents the neuronal membrane potential $U(t)$, which is the sum of all PSPs and the REF (blue curve).

The formula for calculating the membrane potential of SRM neurons is as follows [16]:

$$u(t) = (\gamma(t) * s_{out}(t)) + \sum_i w^i(\epsilon(t) * s_{in}^i(t)) \tag{1}$$

In this equation, $s_{in}(t)$ and $s_{out}(t)$ denote the input and output spikes of the SRM neuron, respectively. The input spike train for the i-th synapse is given by $s_{in}(t) = \sum_k \delta(t - t_k^i)$, where t_k^i represents the timestamp of the k-th spike in this sequence. Each input spike train is associated with a corresponding weight w^i. The membrane potential of the neuron, $u(t)$, generates an output spike when it exceeds the threshold voltage V_{th}. As shown in Equation 1, $u(t)$ is defined as the sum of all PSPs and REFs. Specifically, the PSP is obtained by convolving the input spike train $s_{in}(t)$ with the spike response kernel $\epsilon(t)$ and multiplying the result by the corresponding synaptic weight w^i. Similarly, the REF is derived by convolving the output spike train $s_{out}(t)$ with the REF kernel $\gamma(t)$ [27]. The mathematical definitions of the spike response kernel and the REF kernel are as follows, where τ_s and τ_r are the time constants of the respective kernels, and $\Theta(t)$ denotes the Heaviside step function.

$$\epsilon(t) = \frac{t}{\tau_s} e^{1-\frac{t}{\tau_s}} \Theta(t) \tag{2}$$

$$\gamma(t) = -2V_{th} e^{-\frac{t}{\tau_r}} \Theta(t) \tag{3}$$

B. Principle of event cameras

The event stream output of an event camera is a three-dimensional dataset $M(x, y, t)$, containing spatial coordinates and timestamps. At time t, the event stream output at coordinate (x, y) can be expressed as [28]:

$$M(x,y,t) = \begin{cases} 1, & \int_{t-\Delta t}^{t} E(t)\,dt > \phi, \\ 0, & \int_{t-\Delta t}^{t} E(t)\,dt < \phi \end{cases} \tag{4}$$

The photoreceptor in an event camera converts brightness changes into an electrical signal $E(t)$, which is integrated and compared to a threshold ϕ. A spike is emitted, and the integrator resets to zero when the accumulated signal exceeds ϕ, indicating a detectable brightness change. If the threshold is not reached, the signal continues accumulating. In the model,

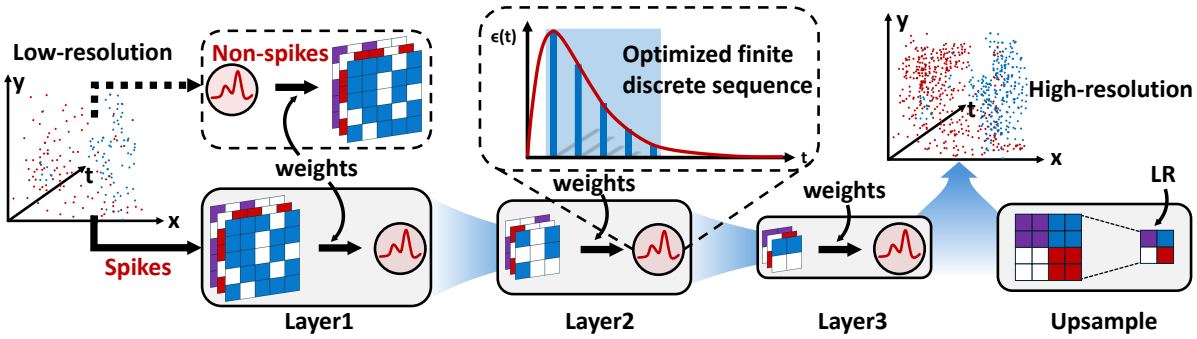


Fig. 3. The overall network architecture for event stream super-resolution. The LR event stream passes from left to right through two convolution layers, one deconvolution layer, and three neuron layers, then combines with the output of the upsampling layer to ultimately produce the HR event stream.

a "1" represents a spike at pixel (x, y, t), generating an event, while "0" indicates no event [29].

The left plot in Fig. 3 shows the discrete event stream output of the event camera, where red and blue pixels represent positive and negative polarities, corresponding to brightness increases and decreases. Event camera data contains spatial and temporal information along the x, y, and t axes, abstracted as a 3D space for processing. Positive and negative polarity events are separated into two input streams, processed independently by the SNNs.

III. Event Stream Super-Resolution SNNs Algorithm

To achieve efficient algorithm-hardware co-design, we propose a hardware-friendly SNNs model for event stream super-resolution image reconstruction. The overall network architecture is illustrated in Fig. 3. The following sections provide a detailed description of its components.

A. Overall architecture

Our network model includes three components: a convolutional module for 2D spatial processing, a neuronal module for 1D temporal processing, and an interpolation module for upsampling. The convolutional module consists of two convolutional layers and one deconvolutional layer, forming a convolutional spiking neural network. The first layer uses a 5×5 kernel, stride 1, and 2-pixel padding; the second uses a 3×3 kernel, stride 1, and 1-pixel padding. The deconvolutional layer has a 2×2 kernel, stride 1, and no padding. Both input and output event streams have 2 channels, while internal channels are set to $n = 4$. The input and output channel dimensions for the three-layer network are $2n$.

The neuron module uses SRM neurons as the network core. The input is a low-resolution event stream from an event camera, with 2 channels representing positive and negative polarity events, corresponding to pixel brightening and darkening. The SRM's membrane potential is calculated using Equation 1, with time constants for the response kernels and REF kernels set to $\tau_s = \tau_r = 1, 2, 4$ across the network's three levels.

The interpolation module enhances spatial details in LR images to reconstruct a high-resolution (HR) event stream, providing higher-quality input for subsequent tasks. Two convolutional layers extract features from the LR event stream without increasing spatial resolution. The deconvolutional layer enlarges feature maps via transposed convolution, while the upsampling module restores spatial details using interpolation. The outputs of these components are summed to generate the final HR event stream, achieving a resolution four times higher than the original.

B. Optimization of the spatial convolution

One advantage of SNNs is their use of discrete spikes as input data [30], enabling convolution's multiply-accumulate operations to be reduced to simple accumulation, which significantly decreases computational complexity.

As illustrated in the dashed box on the left of Fig. 3, the original network architecture applies a 1D convolution (accumulation) with the spike response kernel in the temporal dimension, followed by a 2D convolution (multiply-accumulate) with the spatial kernel. To further simplify the 2D convolution, we reordered these operations, performing 2D convolution before 1D convolution. This adjustment reduces the complex 2D convolution to simple accumulation, effectively lowering the network's computational complexity. For the three-layer structure, the neuron computation in layer l is formulated as:

$$\mathbf{u}^{(l+1)}(t) = \epsilon(t)(\mathbf{W}^{(l)} * \mathbf{s}^{(l)}(t)) + (\gamma(t) * \mathbf{s}^{(l+1)}(t)) \tag{5}$$

$$\mathbf{s}^{(l+1)}(t) = \sum_{t_k \in \{t | u^{(l+1)}(t) = V_{th}\}} \delta(t - t_k) \tag{6}$$

C. Optimization of the temporal neuron

In SRM neuron model, the spike response and REF kernels are exponential functions, which are challenging to implement in hardware. To address this, we discretized these functions by sampling them at discrete time points $t = 0, 1, 2, \ldots$, converting them into 1D convolution sequences [31].

Given that the response kernel decays rapidly after $t = 1$ and falls below 0.01 for $t > 6$, values with absolute magnitudes below 0.01 were discarded, further reducing the kernel to a compact finite sequence. This method transformed the spike response and REF kernels in all three network layers into discrete, finite 1D convolution sequences. This method

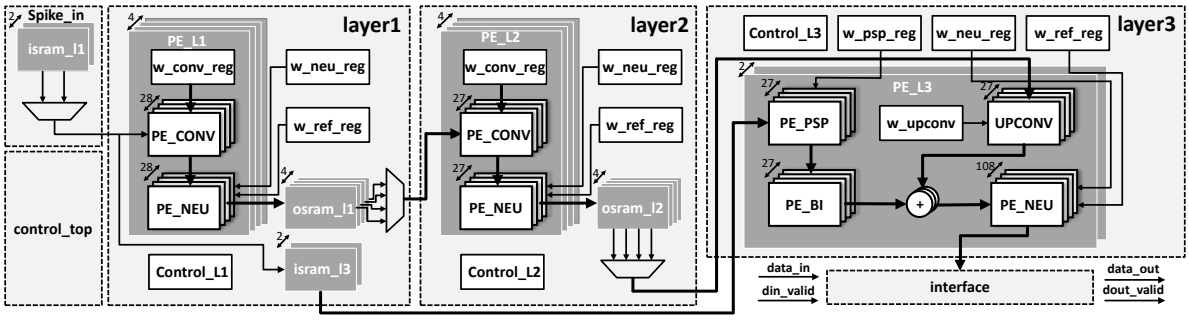


Fig. 4. The overall architecture of the hardware accelerator for event stream super-resolution. The hardware architecture consists of three hierarchical levels to enable highly parallel computation, employing a combination of KCTR data flow and dual-pipeline design to achieve in-situ computing.

not only eliminates the need for complex exponential computations in hardware implementation but also preserves the original network's performance to the greatest extent possible.

D. Optimization of the upsampling interpolation

Common upsampling methods, such as bilinear and nearest-neighbor interpolation, achieve similar model performance. Nearest-neighbor interpolation directly maps the original image to the new image, minimizing computational overhead [32]. Additionally, it reads each row of data only once, significantly reducing memory access frequency.

Therefore, we choose nearest-neighbor interpolation for the upsampling layer, while the floating-point operations in the model are quantized to fixed-point representations. Model weights are scaled by 4 using shift operations and rounded to integers in the range [-128, 127], representing signed 8-bit hardware data. Consequently, the outputs of both 2D and 1D convolutions are scaled down by right-shifting 2 bits. Similarly, the discrete 1D convolution sequences for the spike and REF kernels are quantized like the weights, with their convolution results also right-shifted by 2 bits.

IV. Event Stream Super-Resolution Hardware

A. Hierarchical hardware architecture

The overall hardware architecture of the SNNs accelerator is shown in Fig. 4, with the dashed box representing the first hierarchy. This includes the top-level control module (control_top), interface module (interface), input spike module (spike_in), and three computational layers (layer1-3).

The control_top module oversees initialization, process control, and termination. The interface module handles signal transmission across clock domains. The spike_in module, comprising two SRAMs, stores input spikes from two channels. Layers 1-3 implement the computational logic of the neural network's three layers.

The second hierarchy, shown as overlapping gray boxes, includes identical multi-channel processing modules (PE_L1, PE_L2, PE_L3), spike storage modules, and control units with weight registers for each layer. Input spikes are processed row by row through 2D convolution units, with results passed to neuron computation units to calculate PSP and REF, generating output spikes stored in output memory (OSRAM). To enable parallel processing across four output channels, each

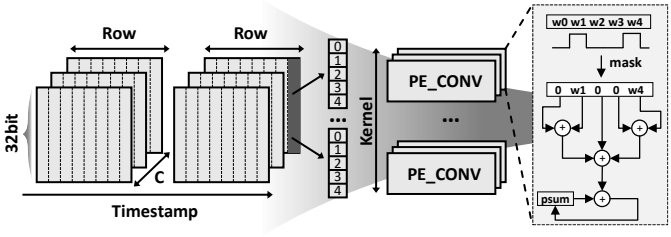


Fig. 5. Kernel-Channel-Timestamp-Row dataflow.

channel uses separate 2D convolution kernel weights, requiring each PE_L1 (or PE_L2) to have its own convolution weight register (w_conv_reg). In contrast, the 1D convolution kernel weights for PSP and REF calculations are shared across all four channels, so all PE_L1 (or PE_L2) modules use common 1D kernel registers (w_neu_reg and w_ref_reg).

The second hierarchy's internal structure comprises parallel processing units, forming the third hierarchy. Its input spikes come partly from the second hierarchy's output spikes and partly from the initial input spikes temporarily stored in the first hierarchy.The processing flow is as follows: output spikes from the second hierarchy are processed by the deconvolution unit (PE_UPCONV), while initial input spikes pass through the 1D convolution unit (PE_CONV) and the interpolation unit (PE_BI). The results from both paths are summed and sent to the neuron unit (PE_NEU), which computes PSP and REF to generate the final output spikes.

B. Key technologies of in-situ computing

In our SNNs hardware architecture, traversing convolution kernels, channels, timestamps, and matrix rows involves extensive computations and frequent memory access to update membrane potentials and spike weights, leading to increased latency and energy consumption [33]. To address this, we optimized the dataflow and proposed a novel KCTR dataflow with a dual-pipeline control structure. This design maximizes neuron reuse, enhances parallelism, and reduces memory data migration. By eliminating intermediate value storage within layers and minimizing inter-layer spike storage, it significantly improves hardware efficiency.

1) KCTR Dataflow: As shown in Fig. 5, the implementation process of the KCTR dataflow is demonstrated. Input spikes are 32 bits wide, and image data is segmented into 32-bit groups with zero-padding for incomplete parts. The KCTR

dataflow transmits data sequentially in the order of Kernel, Channel, Timestamp, and Row.

During computation, the 32-bit input spike is distributed across four output channels for separate processing. Within each channel, it is grouped based on the convolution kernel size k and stride s. In the first layer, the input spike is divided into 28 groups of 5-bit sequences using a sliding stride of $s = 1$. These sequences are transmitted to 28 parallel 2D convolution units, while the weight register sends 5 corresponding 2D convolution weights to enable parallel processing.

In a single 2D convolution unit, input spikes are masked, multiplied by weights, and accumulated, with the results stored in the 2D convolution result register (psum). These operations align strictly along the four dimensions: Kernel, Channel, Timestamp, and Row. Each computation unit processes one row of input spikes per clock cycle, while generating a set of output spikes involves $kernel_size \times input_channels$ computations.

The top-level control module uses multiple counters to manage layer-wise computations sequentially. Each layer's control module comprises four counters: kernel (cnt_kernel), channel (cnt_channel), timestamp (cnt_timestamp), and output row (cnt_row), which form the core of the KCTR dataflow control. These counters ensure orderly data transmission and task execution within each layer. Under the KCTR framework, precise synchronization between the top-level and layer control modules is achieved through counter increments, enabling accurate data flow, efficient pipelining, and parallel computation.

The first-layer convolution unit processes input data across four dimensions: Kernel, Channel, Timestamp, and Row. The kernel counter spans 0–4, the channel counter 0–1, the timestamp counter 0–T (where T is the dataset's number of timestamps), and the output row counter depends on the matrix size. The input spike memory's read address is dynamically computed based on these counters, enabling row-wise reading of five input rows per channel. The control module sequentially activates the 2D convolution unit, accumulating results over 10 iterations in the 2D convolution result register. Once a row of output is generated, the psum data is transferred to the neuron computation unit, and the register is reset for the next cycle.

2) Dual-Pipeline: As shown in Fig. 6, we designed a dual-pipeline architecture for the SNNs hardware accelerator that, combined with the KCTR dataflow, enables high parallelism and in-situ data processing, reducing storage overhead. Within each layer, a MicroPipeline coordinates the parallel operation of computing units for efficient task execution. Between layers, a MacroPipeline optimizes computational efficiency, this allows concurrent operation of the three hardware modules, greatly enhancing overall performance.

The MicroPipeline operates with a 12-clock-cycle period, enabling all computational units to complete a full process within each cycle. During single-layer computations, tasks collaborate in a pipeline: as the 2D convolution unit transfers a partial sum (psum) to the neuron unit, neuron computation starts immediately while the convolution unit begins processing the next row. Within a single task, stages like address generation, spike reading, and psum accumulation also execute in a pipelined manner.

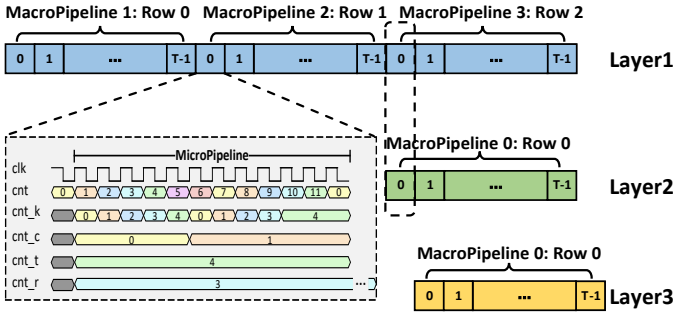


Fig. 6. The MicroPipeline, responsible for intra-layer control, and the MacroPipeline, responsible for inter-layer control.

In the hardware architecture, module states are governed solely by the MicroPipeline counter and enable pulse signals. In the first layer, the KCTR counters in the control module are tightly synchronized with the top-level MicroPipeline counter, with both increment and reset operations driven by it. When the counter reaches 11, pvalid is asserted in the next cycle, signaling the completion of 2D convolution and triggering the neuron computation unit. All related signals propagate seamlessly across stages according to computation clock cycles, eliminating the need for additional control logic and ensuring efficient coordination.

The MacroPipeline facilitates efficient inter-layer pipelining and minimizes inter-layer spike storage by integrating closely with the MicroPipeline. Output spikes from each layer are stored row by row in SRAM, with all T timestamps for a row saved before proceeding to the next. Computation is also row-wise, processing T timestamps sequentially per row. Thus, the MacroPipeline cycle spans T MicroPipeline cycles, requiring $T \times 12$ clock cycles in total.

After three MacroPipeline cycles, the first layer generates output spikes for the first three rows, each with T timestamps. In the following MicroPipeline cycle, the first layer processes the 0th timestamp of the 3rd row, while the second layer reads the 0th timestamp spikes from the 0th, 1st, and 2nd rows of the first layer's output to compute the 0th timestamp of the 0th row for the third layer.

In the next MicroPipeline cycle, the first layer processes the 1st timestamp of the 3rd row. Simultaneously, the second layer reads the 1st timestamp spikes from the first layer's output and computes the 1st timestamp of the 1st row for the third layer. Meanwhile, the third layer reads the 0th timestamp spike of the 1st row from the second layer's output to compute the final output spikes for the 0th and 1st rows, both for the 0th timestamp.This design synchronizes inter-layer computations, minimizing storage for intermediate spike outputs.

V. Experiments

A. Performance evaluation of super-resolution reconstruction

To evaluate the super-resolution image reconstruction accuracy of the designed SNNs hardware accelerator, we conducted

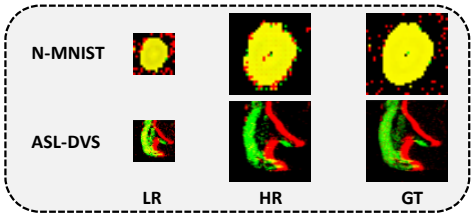


Fig. 7. Reconstruction results of images from different datasets.

TABLE I
COMPREHENSIVE EVALUATION OF RECONSTRUCTION PERFORMANCE.

		Neurocomputing [15]	ICCV [16]	This work
Implementation		GPU	GPU	ASIC
RMSE	N-MNIST	0.757	0.272	1.296
	ASL-DVS	0.229	0.179	**0.121**
Acc	N-MNIST	97.80%	99.70%	98.84%
	ASL-DVS	98.00%	99.90%	99.73%
Speed	N-MNIST	Na	38.99ms	**1.69ms**
	ASL-DVS	Na	164.39ms	**22.37ms**
Power		Na	30W	**649mW**

hardware-based tests using the spike-based N-MNIST [34] and ASL-DVS [35] datasets. The N-MNIST dataset comprises 70,000 handwritten digit images (0–9) captured from the MNIST dataset using a moving sensor, while the ASL-DVS dataset includes 100,800 gesture samples of 26 English letters (a–z) recorded in real-world scenarios.

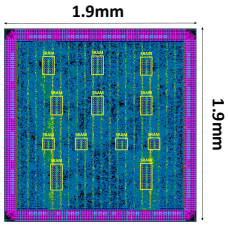
Fig. 7 shows super-resolution results for two datasets: a handwritten "0" and the hand gesture for "c." From left to right are the low-resolution input (LR), super-resolved output (HR), and ground truth (GT). In both cases, the HR closely matches the GT, clearly identifying the targets. Notably, HR images also exhibit reduced edge noise and improved sharpness compared to GT.

Currently, hardware implementations of SNNs for event stream super-resolution image reconstruction remain unexplored. To fill this gap, we compare our hardware-based SNNs with prior software-based ones. Table I summarizes metrics including RMSE, task accuracy, speed, and energy efficiency. While software SNNs (typically GPU-based with floating-point precision) offer higher accuracy, they require much more time and energy.

Compared to studies [15] and [16], the proposed design achieves lower reconstruction error on the ASL-DVS dataset but slightly higher error on the N-MNIST dataset, primarily due to the nearest-neighbor interpolation method, which is more error-prone on datasets with smaller sample sizes.

The SNNs hardware accelerator achieves 98.84% and 99.73% accuracy on N-MNIST and ASL-DVS datasets, respectively, outperforming software-based SNNs in [15] and trailing [16] by less than 1%. It consumes only 664mW, much lower than the 30W of GPU-based implementations. This peak power, measured at max frequency, can be reduced by lowering frequency, enabling flexible performance-power trade-offs ideal for energy-sensitive edge applications.

Processing times are also markedly improved: 1.69 ms per image for N-MNIST (vs. 38.99 ms on GPU) and 22.37 ms for ASL-DVS (vs. 164.39 ms), representing speed gains of 86.4% and 95.6%, respectively.



Technology	28nm
Chip size	3.61mm^2
Voltage	0.9V
Memory	21.45kB
Frequency	500MHz
Energy	0.546pJ/SOP

Fig. 8. Hardware accelerator layout and key technical parameters.

TABLE II
COMPARED WITH STATE-OF-THE-ART NEUROMORPHIC ACCELERATORS.

	VLSI [36]	ISSCC [37]	JSSC [38]	TCAS-I [39]	This work
Technology	40nm	28nm	28nm	40nm	28nm
Area	0.84*0.84	0.67*0.67	0.93*1.77	0.95*0.92	1.9*1.9
Neurons	Na	256-256-16	1024-512-10	Na	289-2312-4624-9248
Neuron density(/mm^2)	Na	748	949	Na	**4563**
Memory(kB)	53.84	138	266.5	130.25	**21.45**
Weight bitwidth	8/16bit	8bit	8bit	8/16bit	8bit
Frequency (MHz)	259	115	210	150	**500**
Energy(pJ/SOP)	1.89@0.65V	5.3@0.5V	1.5@0.56V	9.9@0.75V	**0.546@0.9V**
Application	Image Classification	Gesture Recognition	Keyword Recognition	Image Classification	Event stream Super-resolution

B. Performance evaluation of the SNNs hardware accelerator

The proposed hardware accelerator, implemented using a 28nm CMOS process, as shown in Fig. 8. We compare it with SNNs hardware accelerators designed for other applications, as no dedicated SNNs hardware exists for image super-resolution reconstruction. Table II summarizes the results.

In terms of hardware resources, larger SNNs with more layers and neurons enhance parallelism and speed but consume more hardware resources. We implement the largest network scale within a moderate chip area, achieving higher neuron density than [36]–[39]. Memory usage is minimized via the KCTR dataflow and dual-pipeline design, which avoid storing intermediate values and reduce inter-layer spike storage.

The SNNs hardware accelerator designed in this study operates at a maximum frequency of 500 MHz, achieving computational efficiency far exceeding that of studies [36]–[39], while maintaining exceptionally low power consumption. Measured by average energy per SOP, the proposed hardware achieves an energy efficiency of 0.546 pJ/SOP under high-frequency, highly parallel operation, reducing energy consumption by 64% to 90% compared to other SNNs accelerators.

VI. CONCLUSION

Event cameras use event stream representation to effectively reduce motion blur and overexposure but are limited by their low spatial resolution. To address this, We propose the first event stream super-resolution hardware accelerator utilizing SNNs. The architecture features hardware-friendly algorithms via neuron model simplification and convolution optimization. A hierarchical design enables highly parallel computation, utilizing KCTR dataflow and dual-pipeline mechanisms for in-situ processing, which eliminates intermediate value storage and minimizes inter-layer spike storage.

REFERENCES

[1] C. Brandli, R. Berner, M. Yang, S.-C. Liu, and T. Delbruck, "A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor," *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014.

[2] G. Gallego, T. Delbrück, G. Orchard, C. Bartolozzi, B. Taba, A. Censi, S. Leutenegger, A. J. Davison, J. Conradt, K. Daniilidis *et al.*, "Event-based vision: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 1, pp. 154–180, 2020.

[3] X. Zheng, Y. Liu, Y. Lu, T. Hua, T. Pan, W. Zhang, D. Tao, and L. Wang, "Deep learning for event-based vision: A comprehensive survey and benchmarks," *arXiv preprint arXiv:2302.08890*, 2023.

[4] T. Delbruck and M. Lang, "Robotic goalie with 3 ms reaction time at 4% CPU load using event-based dynamic vision sensor," *Frontiers in neuroscience*, vol. 7, p. 223, 2013.

[5] H. Kim, S. Leutenegger, and A. J. Davison, "Real-time 3D reconstruction and 6-DoF tracking with an event camera," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI 14*. Springer, 2016, pp. 349–364.

[6] M. Bouvier, A. Valentian, and G. Sicard, "Scalable Pitch-Constrained Neural Processing Unit for 3D Integration with Event-Based Imagers," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 385–390.

[7] A. Amir, B. Taba, D. Berg, T. Melano, J. McKinstry, C. Di Nolfo, T. Nayak, A. Andreopoulos, G. Garreau, M. Mendoza *et al.*, "A low power, fully event-based gesture recognition system," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7243–7252.

[8] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 2, pp. 566–576, 2008.

[9] C. Posch, D. Matolin, and R. Wohlgenannt, "A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 1, pp. 259–275, 2010.

[10] B. Son, Y. Suh, S. Kim, H. Jung, J.-S. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo *et al.*, "A 640x 480 dynamic vision sensor with a 9 m pixel and 300meps address-event representation," 2017.

[11] P.-Y. Chuang, P.-Y. Tan, C.-W. Wu, and J.-M. Lu, "A 90nm 103.14 TOPS/W Binary-Weight Spiking Neural Network CMOS ASIC for Real-Time Object Classification," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[12] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *International Journal of Computer Vision*, vol. 113, pp. 54–66, 2015.

[13] D. Gehrig, M. Gehrig, J. Hidalgo-Carrió, and D. Scaramuzza, "Video to events: Recycling video datasets for event cameras," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 3586–3595.

[14] D. Sharma, A. K. Kosta, and K. Roy, "Lightning Talk: A Perspective on Neuromorphic Computing," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023, pp. 1–2.

[15] H. Li, G. Li, and L. Shi, "Super-resolution of spatiotemporal event-stream image," *Neurocomputing*, vol. 335, pp. 206–214, 2019.

[16] S. Li, Y. Feng, Y. Li, Y. Jiang, C. Zou, and Y. Gao, "Event stream super-resolution via spatiotemporal constraint learning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 4480–4489.

[17] X. Xiang, L. Zhu, J. Li, Y. Wang, T. Huang, and Y. Tian, "Learning super-resolution reconstruction for high temporal resolution spike stream," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 33, no. 1, pp. 16–29, 2021.

[18] A. I. Maqueda, A. Loquercio, G. Gallego, N. García, and D. Scaramuzza, "Event-based vision meets deep learning on steering prediction for self-driving cars," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5419–5427.

[19] G. Wulfram and M. K. Werner, "Spiking neuron models: Single neurons, populations, plasticity," *Encyclopedia Neurosci*, vol. 4, pp. 277–280, 2002.

[20] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952.

[21] W. Gerstner and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[22] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[23] P. Dayan and L. F. Abbott, *Theoretical neuroscience: computational and mathematical modeling of neural systems*. MIT press, 2005.

[24] Gerstner, "Statistical physics, plasmas, fluids, and related interdisciplinary topics," *American Physical Society. (2000). Physical review. E*, 1995.

[25] R. Jolivet, T. J. Lewis, and W. Gerstner, "Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy," *Journal of neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004.

[26] H. Fang, B. Taylor, Z. Li, Z. Mei, H. H. Li, and Q. Qiu, "Neuromorphic Algorithm-hardware Codesign for Temporal Pattern Learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 361–366.

[27] W. M. Kistler, W. Gerstner, and J. L. van Hemmen, "Reduction of the Hogkin-Huxley equations to a single-variable threshold model," *Neural Computation*, vol. 9, no. 5, pp. 1015–1045, 1997.

[28] T. Serrano-Gotarredona and B. Linares-Barranco, "A 128 × 128 1.5% Contrast Sensitivity 0.9% FPN 3 µs Latency 4 mW Asynchronous Frame-Free Dynamic Vision Sensor Using Transimpedance Preamplifiers," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 3, pp. 827–838, 2013.

[29] P. Bardow, A. J. Davison, and S. Leutenegger, "Simultaneous optical flow and intensity estimation from an event camera," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 884–892.

[30] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: opportunities and challenges," *Frontiers in neuroscience*, vol. 12, p. 409662, 2018.

[31] D. R. Kunkle and C. Merrigan, "Pulsed neural networks and their application," *Computer Science Dept., College of Computing and Information Sciences, Rochester Institute of Technology*, 2002.

[32] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[33] K. Roy, A. Jaiswal, and P. Panda, "Towards spike-based machine intelligence with neuromorphic computing," *Nature*, vol. 575, no. 7784, pp. 607–617, 2019.

[34] G. Orchard, A. Jayawant, G. K. Cohen, and N. Thakor, "Converting static image datasets to spiking neuromorphic datasets using saccades," *Frontiers in neuroscience*, vol. 9, p. 437, 2015.

[35] Y. Bi, A. Chadha, A. Abbas, E. Bourtsoulatze, and Y. Andreopoulos, "Graph-based object classification for neuromorphic vision sensing," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 491–501.

[36] P.-Y. Tan and C.-W. Wu, "A 40-nm 1.89-pJ/SOP Scalable Convolutional Spiking Neural Network Learning Core With On-Chip Spatiotemporal Back-Propagation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[37] C. Frenkel and G. Indiveri, "ReckOn: A 28nm sub-mm2 task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 1–3.

[38] J. Zhang, D. Huo, J. Zhang, C. Qian, Q. Liu, L. Pan, Z. Wang, N. Qiao, K.-T. Tang, and H. Chen, "ANP-I: A 28-nm 1.5-pJ/SOP Asynchronous Spiking Neural Network Processor Enabling Sub-0.1- µJ/Sample On-Chip Learning for Edge-AI Applications," *IEEE Journal of Solid-State Circuits*, 2024.

[39] C. Liu, Z. Yang, X. Zhang, Z. Zhu, H. Chu, Y. Huan, L.-R. Zheng, and Z. Zou, "A Low-Power Hybrid-Precision Neuromorphic Processor With INT8 Inference and INT16 Online Learning in 40-nm CMOS," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 70, no. 10, pp. 4028–4039, 2023.

GraphAccel: An In-Storage Accelerator for Efficient Graph-Based Vector Similarity Search Using Page Packing and Speculative Search Optimization

Yoonyoung Kwon*
Sungkyunkwan University
Suwon, Korea
kyy0307@g.skku.edu

Yunjong Boo*
Sungkyunkwan University
Suwon, Korea
qndbswhd1@g.skku.edu

Hyungmin Cho
Sungkyunkwan University
Suwon, Korea
hyungmin.cho@skku.edu

***Abstract*—Graph-based search for approximate vector similarity is essential in AI applications, such as retrieval-augmented generation. To support large-scale searches, vector search graphs are often stored on storage devices like SSDs. In this paper, we introduce GraphAccel, an in-storage accelerator optimized for efficient graph-based vector similarity search. Our architecture incorporates an optimized page packing mechanism to reduce SSD page accesses per query, alongside a speculative search scheme that maximizes utilization of idle SSD chips and channels. Through these optimizations, GraphAccel achieves notable performance improvements over existing SSD-based graph search solutions, including DiskANN and DiskANN++.**

***Index Terms*—In-Storage Processing, SSD Parallelism, Vector Search.**

I. Introduction

The rapid growth of unstructured data, such as images [1], text [2], and audio [3], has created a need for processing methods that preserve semantic information. Semantic vectors address this challenge by transforming data into high-dimensional representations that capture the relationships between data points. A critical operation on large-scale vector databases is finding similar vectors for a given query. Exact similarity search often relies on brute-force methods, which can be computationally expensive. As a result, many applications adopt approximate nearest neighbor search (ANNS) [4], [5], which is widely used in fields such as recommendation systems [6], computer vision [7], and retrieval-augmented generation (RAG) [8].

Many ANNS algorithms utilize graph-based approaches, which organize vectors into a navigable graph structure for indexing [9], [10]. When a query is issued, the search algorithm traverses this graph in two primary stages: *node expansion* and *distance calculation*. During the node expansion stage, the algorithm selects the next node to expand based on its proximity to the query, retrieving its vector and neighboring node information. In the distance calculation stage, the distance between the expanded node's vector and the query vector is computed, and candidate nodes are updated accordingly. By iteratively repeating these stages, the algorithm identifies the nodes in the graph that are closest to the query vector.

As dataset sizes continue to grow, large-scale vector search poses unique challenges: they are both data-intensive and exhibit complex, unpredictable access patterns across varying queries that complicate efficient processing. These characteristics demand high-throughput and low-latency solutions, especially as ANNS graphs are frequently stored on storage devices to support scalability.

Storing a vector database on flash-based storage devices, such as Solid-State Drives (SSDs), presents significant challenges. The inherently slow access latency of flash storage hampers search performance, as it involves repeated, unpredictable access to flash pages. To address this, effective strategies are needed to minimize the number of page accesses during a search. Additionally, the unpredictable read latency of SSDs necessitates dynamic mechanisms to maximize SSD utilization and reduce overall search latency.

In this paper, we introduce *GraphAccel*, an in-storage accelerator designed to address these challenges by optimizing graph-based vector similarity searches on SSDs. First, we propose a new packing mechanism that groups nodes that are accessed together frequently into the same flash page, reducing flash access roundtrips during query processing. Additionally, GraphAccel incorporates a search algorithm that leverages In-Storage Processing (ISP) capabilities by monitoring hardware resource status to maximize SSD parallelism, sending additional speculative requests to idle channels and chips. This approach minimizes total page accesses during queries, maximizes hardware resource utilization, and ultimately achieves high accuracy with low latency.

We evaluated GraphAccel's search performance using large-scale vector datasets, including Turing100M [11], SIFT1B [12], and DEEP1B [13]. Compared to its foundational systems, DiskANN [9] and DiskANN++ [14], GraphAccel achieves up to 80.5% and 73.4% reductions in search latency, respectively, while maintaining equivalent levels of recall quality.

II. Background

A. SSD Preliminary

SSDs store data using individual cells that operate through electrically controlled mechanisms for data operations. These cells adjust their stored charge to establish specific voltage levels, which are sensed to decode the stored data. While this electrical design enables efficient data handling, it also introduces reliability challenges such as *read disturbance* and *retention loss* [15], [16].

Read disturbance occurs when repeated read operations on a cell cause slight voltage shifts in adjacent cells, potentially altering their threshold voltage levels, risking data corruption. In contrast, retention loss refers to the gradual leakage of electrons from cells over time, which lowers the stored voltage levels and degrade data accuracy.

To mitigate these issues and maintain reliability, modern SSDs employ Error Correction Codes (ECC) and *read-retry* mechanisms [17]–[19]. When the number of error bits exceeds the ECC's correction capability, the read-retry mechanism is triggered. This mechanism works by adjusting the voltage thresholds during read operations to reduce misread cells, thereby decreasing the error bit count to within ECC's corrective capacity. Once reduced, ECC corrects the remaining errors, restoring data integrity.

* Both authors contributed equally to this work.

While read-retry is critical for ensuring reliability, it introduces trade-offs. Each retry involves re-performing the read operation, increasing overall read latency. The number of retries required depends on factors such as the severity of read disturbance and the extent of retention loss. Consequently, read-retry operations add variability and unpredictability to SSD read latency.

B. DiskANN Search

TABLE I: Notations used in this work.

Symbol	Description
B	Beamwidth
$\mathcal{C}$	Maxheap for search list
$\mathcal{R}$	Maxheap for Top-k results
v	Node in the search graph
$N(v)$	Set of v's neighbor nodes
$\vec{v_{pq}}$	PQ vector of v
$\vec{v_{full}}$	Full vector of v

DiskANN's graph search algorithm identifies the k-closest nodes to a query by iterative *node expansions* [9]. It maintains a search list $\mathcal{C}$ of size L, from which the closest node to the query is selected for expansion. Node expansion involves reading the neighbors of the selected node, calculating their distances to the query, and adding them to $\mathcal{C}$ as potential candidates. This process repeats until all nodes in $\mathcal{C}$ are expanded, and the algorithm reports the top k results in $\mathcal{C}$.

DiskANN further optimizes performance by utilizing two distance calculation methods. When a node is initially added to the search list as a neighbor of an expanded node, the algorithm calculates an approximate distance using product quantization (PQ) [20], a technique that reduces vector dimensionality and computational overhead. Nodes selected for expansion are subsequently re-evaluated using their full vectors for greater accuracy, while their neighbors continue to be assessed with PQ vectors. Fully expanded nodes evaluated with their full vectors are stored in a separate top-k result heap, $\mathcal{R}$.

Algorithm 1: DiskANN Vector Graph Search

1 **Input:** Query vector q, k of top-k, Search List size L, Graph G
2 **Output:** k nearest neighbors of a query q
3 $v_c \leftarrow$ central vertex of G;
4 maxHeap $\mathcal{C}$ max capacity with L;
5 maxHeap $\mathcal{R}$ max capacity with k;
`// Insert` v_c `with computed PQ distance`
6 $\mathcal{C}.insert(v_c, PQ_dist(\vec{v_{c_{pq}}}))$;
7 **while** $\mathcal{C} - \mathcal{R} \neq \emptyset$ **do**
8 $v_s \leftarrow$ extract closest unexpanded node from $\mathcal{C}$;
`// SSD I/O request`
9 read page $\mathcal{P}$ containing $\vec{v_{s_{full}}}$, $N(v_s)$ from SSD;
`// insert` v_s `with computed full distance`
10 $\mathcal{R}.insert(v_s, full_dist(\vec{v_{s_{full}}}))$;
`// insert` $N(v_s)$ `with computed PQ distance`
11 **for** v ***in*** $N(v_s)$ **do**
12 read page $\mathcal{P}'$ containing $\vec{v_{pq}}$ from SSD;
13 $\mathcal{C}.insert(v, PQ_dist(\vec{v_{pq}}))$;
14 **end**
15 **end**
16 **return** $\mathcal{R}$

In the original DiskANN, PQ vectors are kept in memory, with only full vectors stored on disk. While feasible for smaller graphs due to PQ compression ($\frac{1}{4}\times$ in DiskANN), larger graphs may exceed memory limits. To support extremely large graphs as well, we assume *even PQ vectors reside on storage*.

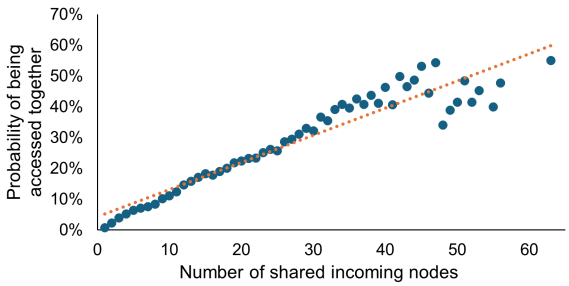


Fig. 1: Probability for a pair of nodes being accessed together based on the number of sharing incoming nodes.

Algorithm 1 and Table I summarize DiskANN's operation. The algorithm initializes with the graph's central node (line 3) and uses two max-heaps: $\mathcal{C}$ (capacity L) for PQ-evaluated candidates and $\mathcal{R}$ (capacity k) for top-k full-vector results (lines 4–5). The central node is inserted into $\mathcal{C}$ with its PQ distance (line 6). Node expansion is repeatedly applied (lines 8–14). The search proceeds while $\mathcal{C}$ contains candidates not yet in $\mathcal{R}$ (line 7).

III. Packing Algorithm of GraphAccel

The page size, the minimum granularity of read operations in flash storage, is typically larger than the size of a vector node. To optimize query performance, it is crucial to implement an efficient packing mechanism that groups vector nodes likely to be accessed together on the same page. Without such a mechanism, neighboring nodes spread across different pages would require additional I/O requests, leading to increased SSD roundtrips and degraded search performance.

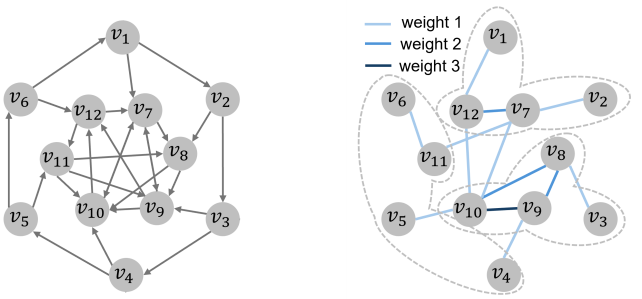
A. Starpacking

DiskANN++ addresses the packing issue by introducing an algorithm called *Starpacking*. Starpacking aims to co-locate each node with its $(b-1)$ nearest neighbors on the same page, where b represents the page capacity in terms of the number of vectors. The packing algorithm operates iteratively: starting with an unexpanded node, it groups the node with its $(b-1)$ closest unexpanded neighbors onto a single page. Once nodes are assigned to a page, they are marked as expanded, and the process continues with the remaining unexpanded nodes until all nodes are packed.

However, as Starpacking follows a greedy approach, the resulting packing is not globally optimal. The quality of the packing depends heavily on the order in which nodes are expanded. For nodes processed later in the sequence, where most neighboring nodes have already been marked as expanded, only a few of their neighbors can be placed on the same page. Consequently, query vectors that examine these nodes may require accessing a larger number of pages, resulting in increased I/O overhead and reduced search efficiency.

B. GraphAccel-Packing

The motivation for our GraphAccel-Packing algorithm stems from the insight that nodes frequently accessed together are not necessarily direct neighbors but rather nodes that share the same predecessor (incoming) node. Even if two nodes are not direct neighbors, if they share a common predecessor, they effectively become neighboring nodes relative to that predecessor. When the predecessor node is expanded, both of these nodes are accessed simultaneously.



(a) (b)

Fig. 2: GraphAccel-Packing example. (a) Directed graph with DiskANN indexing. (b) Transformed undirected graph with weighted edges.

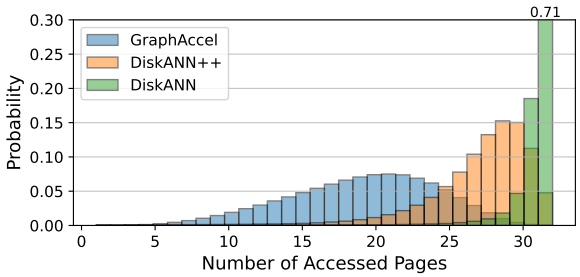


Fig. 3: Number of accessed pages when expanding a node with 32 neighbors.

To support this intuition, Fig. 1 illustrates the probability of two nodes being accessed together during query processing as a function of the number of shared incoming nodes. The probability is evaluated during search operations on the SIFT1M dataset using DiskANN indexing. The results demonstrate a strong positive correlation: the greater the number of shared incoming nodes, the higher the likelihood of the two nodes being accessed together.

Building on this motivation, we propose a packing mechanism that reframes the vector packing problem as a graph partitioning problem. Specifically, we transform the original directed graph into an undirected graph, where the edge weights are determined by the number of shared incoming nodes between connected nodes. We then partition this transformed undirected graph into sub-graphs each containing up to b nodes, and assign each sub-graph to a page.

Figure 2 illustrates the process of our packing method. The starting point is a directed graph constructed using DiskANN indexing, as shown in Fig. 2a. This graph is then converted into an undirected graph by connecting nodes that share common incoming nodes. The edge weights in the converted graph represent the number of shared incoming nodes between two connected nodes.

For example, while there is no direct edge between v_2 and v_7 in the original graph, both are neighbors of v_1 and will be accessed together when expanding v_1. Thus, in the converted graph, we introduce an edge between v_2 and v_7, enabling them to be packed on the same page. Similarly, for nodes sharing multiple incoming nodes, higher edge weights are assigned to reflect their higher likelihood of being accessed together. For instance, the edge weight between v_9 and v_{10} is three, as they share v_7, v_8, and v_{11} as incoming nodes.

Once the graph is transformed, we utilize PulP [21], a tool designed for large-scale graph partitioning, to partition the graph while minimizing the cut weights.

Before presenting a full evaluation of our packing mechanism in Sec. V, we provide a preview of its advantages over Starpacking in Fig. 3. The figure compares the number of pages accessed when expanding a node's neighboring nodes, using the SIFT1M dataset where each node has up to 32 neighbors. In DiskANN, which employs no specialized packing mechanism, neighboring nodes are distributed arbitrarily across pages. This results in up to 32 pages being accessed per node, causing a significant number of SSD I/O requests for each access.

While DiskANN++ reduces the number of accessed pages using Starpacking, nodes still require an average of 27.23 pages. This limitation arises because Starpacking maintains strong connectivity for nodes processed earlier in the packing process, but connectivity becomes fragmented for nodes processed later.

In contrast, our GraphAccel-Packing method significantly minimizes the number of pages required to access a node's neighbors. On average, our approach reduces the number of accessed pages per query by approximately 58.4% compared to DiskANN++ and by 72.8% compared to DiskANN, demonstrating its superior efficiency in packing and query performance.

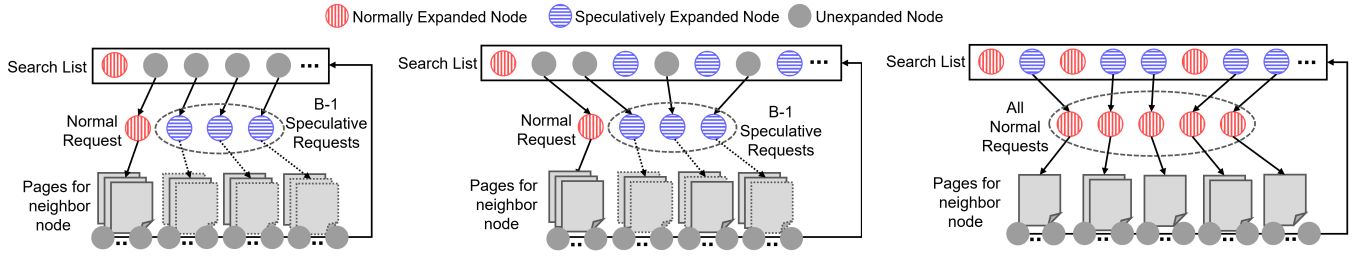
A downside of our approach is the requirement for large-scale graph partitioning. To enable partitioning on large-scale search graphs, we divide a large graph into multiple subgraphs, disregarding inter-subgraph connections. While this approach may not fully retain the original graph's structure, it only impacts the packing quality, leaving search accuracy unaffected. When splitting the graph into subgraphs, fewer but larger subgraphs preserve more of the original graph structure. However, larger subgraphs also increase computation time for calculating edge weights due to the greater number of nodes within each subgraph. To balance the trade-off between preserving graph information and managing computational overhead, we set the subgraph size to 50 million nodes. With this configuration, the calculation and storage of edge weights for each subgraph, as well as the subsequent packing process, required an average of 12 hours per subgraph. To expedite the process, all subgraph partitioning operations are executed in parallel.

IV. GraphAccel-Search

Disk-based ANN graph search is inherently constrained by the performance of disk access. To improve search performance, it is essential to exploit the abundant parallelism available in SSDs at both the channel and chip levels. However, the basic search mechanism, as outlined in Algorithm 1, fails to take full advantage of this parallelism due to its inherently sequential nature. Specifically, it computes and selects the next closest node for expansion only after completing disk accesses for the current closest node.

To address this limitation, DiskANN introduces the *Beamsearch* mechanism, which partially leverages SSD parallelism by expanding multiple nodes simultaneously. However, Beamsearch has inherent constraints, as it treats all node expansions equally, lacking the ability to prioritize critical accesses over non-critical ones.

In this work, we present *GraphAccel-Search*, a novel mechanism designed to fully utilize SSD parallelism through speculative search executions. By distinguishing between critical and speculative node expansions and dynamically managing SSD resources, GraphAccel-Search ensures that critical requests are prioritized while idle SSD resources are opportunistically used for speculative expansions. This approach overcomes the limitations of Beamsearch and delivers



(a) (b) (c)

Fig. 4: Phases of node expansion in GraphAceel-Search. (a) Initial phase after expanding the entry node. (b) Middle phase of the search. (c) Batch re-examination phase.

significant performance improvements for disk-based ANN graph search.

A. Limitations of Beamsearch

Beamsearch enhances the search algorithm by expanding a group of additional nodes (referred to as the *beam*) alongside the closest node, with the goal of improving SSD utilization. The beamwidth B refers to the number of nodes in a beam. However, the extra nodes selected for expansion are not always critical to the search. For example, the neighbors of the closest node may have greater relevance to the query than the nodes chosen for beam expansion. In such scenarios, sequentially expanding one node at a time may be more efficient, avoiding unnecessary use of SSD resources on irrelevant nodes.

Beamsearch introduces an inherent delay since the expansion of the next closest node occurs only after the current beam request is fully processed. If the beam includes irrelevant nodes, this delay is compounded, further slowing search traversal. Additionally, SSDs are susceptible to read retries caused by read disturbances, leading to unpredictable latency. These variations exacerbate inefficiencies in node processing, resulting in increased overall search latency.

B. GraphAccel-Search High-level Overview

GraphAccel's search algorithm leverages the abundant parallelism in SSDs by employing speculative search executions. Similar to the Beamsearch mechanism used in DiskANN, GraphAccel-Search issues additional node expansion requests. However, these additional requests are handled as *speculative* rather than essential. The key distinction of GraphAccel-Search is that the SSD controller categorizes requests into two types:

1) **Normal requests**, associated with the closest unexpanded node, are processed without dropping.
2) **Droppable requests**, associated with speculative node expansions, are processed opportunistically.

If the requested SSD resources—-such as the corresponding channels or flash chips-—are occupied when a droppable request is issued, the SSD controller cancels the request. This ensures that critical resources are preserved for processing normal requests.

Since speculative requests may be partially dropped, nodes expanded during speculative executions can remain in an incomplete state. To maintain accuracy, GraphAccel-Search includes a *re-examination phase*, analogous to the verification or correction step in speculative execution pipelines in CPUs. This phase enhances the quality of results while maximizing parallelism in earlier stages.

The re-examination phase is performed in two scenarios:

1) **Natural re-examination** occurs when a speculatively expanded node becomes the closest unexpanded node during subsequent steps of the search.
2) **Batch re-examination** occurs when all nodes in the search list are either fully expanded or speculatively expanded. At this stage, the algorithm revisits the remaining speculatively expanded nodes to resolve any incomplete states. During this process, all requests are treated as normal to ensure accuracy.

Although the number of nodes requiring re-examination can be large, especially with a larger Beamwidth, the impact on performance is mitigated. This is because most neighbor nodes are already accessed during speculative executions, requiring only a few additional flash accesses during the re-examination step.

C. GraphAccel-Search Details

Figure 4 illustrates how GraphAccel-Search handles node expansions during different phases of query processing.

Figure 4a shows the status after the initial entry node has expanded. The initial entry node is marked as expanded, and its neighbors are added to the search list. Among these neighbors, the one closest to the query, as determined by the PQ distance, is selected for non-droppable expansion. Meanwhile, the next B-1 closest neighbors are selected for speculative expansion, which can be dropped if necessary. Droppable requests, indicated by dotted lines in the figure, may be canceled by the SSD controller during flash access processing. Once the neighbor PQ vectors are fetched, these neighbors are added to the search list with their PQ distances. Upon completing the batch of droppable and non-droppable requests, the closest node is marked as fully expanded (red), while the B-1 speculatively expanded nodes are marked for re-examination later (blue).

Figure 4b shows the middle phase of the search process. At this stage, the search list contains fully expanded nodes, speculatively expanded nodes, and unexpanded nodes. For the next requests, the closest unexpanded or speculatively expanded node is selected for non-droppable expansion. If a speculatively expanded node is chosen, this effectively re-examines the earlier speculation. Alongside the non-droppable request, B-1 unexpanded nodes are selected for speculative expansion. Importantly, nodes that have already undergone speculative expansion are not selected for speculation again.

Figure 4c depicts the phase when all nodes are either fully expanded or speculatively expanded. In this phase, B^* nodes are expanded in a batch, all treated as non-droppable requests. This batch expansion focuses on re-examining speculative nodes to finalize their evaluation. Since most pages have already been accessed by this stage, fewer flash page requests are generated, allowing for a

larger batch size to maximize SSD parallelism. In our experiments, we empirically set B^* to 100 to fully exploit available parallelism.

The search concludes when all nodes in the search list are fully expanded. If batch re-examination reintroduces unexpanded nodes to the search list, the algorithm repeats the speculative expansion and batch re-examination phases until full expansion is achieved.

Note that not all speculatively expanded nodes proceed to the re-examination phase. If a speculatively expanded node is removed from the search list of size L by closer nodes to the query, it is not re-examined. This selective re-examination enhances the efficiency of our opportunistic evaluation strategy for speculative expansion, as opposed to uniformly exploring all B nodes in the beam.

D. GraphAccel-Search SSD Structure

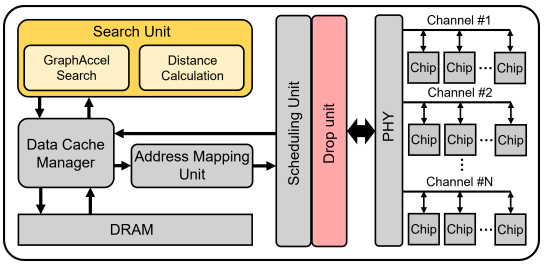


Fig. 5: High-level view of the components of GraphAccel.

Figure 5 illustrates the high-level interactions between the components of GraphAccel. To enhance vector search performance using ISP capabilities, two new components are introduced: the *Search Unit*, which executes the search algorithm and performs distance calculations, and the *Drop Unit*, which manages the cancellation of speculative droppable requests at the time of request issuance.

A dedicated data path between the Search Unit and the *Data Cache Manager* (DCM) enables efficient communication. If the requested data is located in DRAM, the DCM retrieves it and forwards it directly to the Search Unit for computation, avoiding unnecessary fetch operations from the flash chip.

When requests are submitted to the Scheduling Unit for chip roundtrips, the Drop Unit evaluates each request to determine whether it is droppable. For droppable requests, the Drop Unit verifies the status of the target chip and channel, processing the request only if both are idle, thereby optimizing resource utilization.

V. Evaluation

A. Experimental Setup

Implementation. We implement GraphAccel within MQSim [22], a widely used simulator for modern NVMe SSDs. We model the specification of SSD as below Table II. The SSD internal DRAM is set to 1 GB, with a storage capacity of 1 TB.

The newly integrated Search Unit is designed to operate using the same microcontroller as the flash translation layer of the SSD controller. We estimate its compute latency by profiling execution times on an ARM Cortex-A9 core processor running at a clock frequency of 200 MHz.

To model SSD read retries, we utilize measurement data from the literature [17], focusing on three retention age scenarios: 0 months, 3 months, and 6 months.

Datasets. We selected three datasets, Turing100M, SIFT1B, and DEEP1B, for evaluating large-scale vector searches, each containing over 100 million nodes. The details of these datasets are summarized in Table III. The search performance is evaluated by measuring the recall rate (R@k) against the total latency for processing queries from each dataset. This performance is compared with existing vector search algorithms, including DiskANN and DiskANN++, all evaluated under the same ISP environment as GraphAccel.

TABLE II: System Configurations

System	Config	Value
Search Unit	Clock Frequency	200MHz
Memory	Capacity	1GB
	Bandwidth	2GB/s
Storage	Topology	8 Channels, 4 Chips / Channel
	Capacity	1TB
	Read Latency	50us per single page

TABLE III: Dataset Information

Dataset	# Node	# Dimension	Type	Size (GB)
Turing100M [11]	100,000,000	100	float32	38
SIFT1B [12]	1,000,000,000	128	uint8	119
DEEP1B [13]	1,000,000,000	96	float32	358

B. Experimental Results

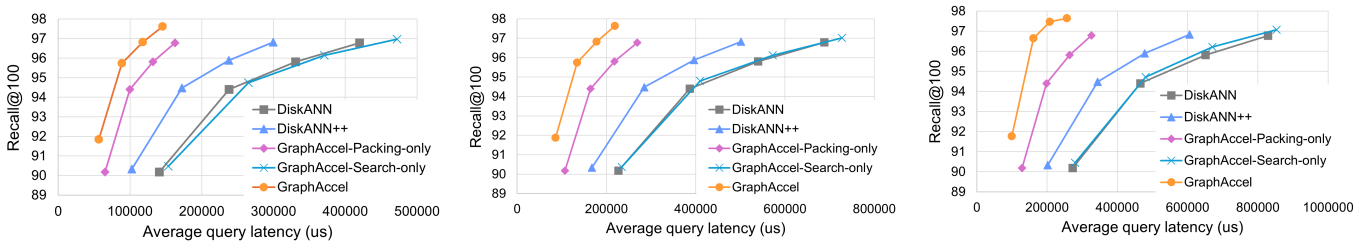
Resistance to Variable Read Latencies. Figure 6 compares the performance of GraphAccel, DiskANN, and DiskANN++ using the SIFT1B dataset, with the beamwidth of 16. We varied the search list length, L, from 500 to 2,000 in increments of 500 to analyze the trade-offs between search latency and recall.

GraphAccel demonstrates superior performance, achieving higher recall values with shorter overall latency. For example, at a recall level of approximately 97%, GraphAccel achieves a search latency reduction of 2.54×, 2.28×, and 2.36× compared to DiskANN++ for retention ages of 0, 3, and 6 months, respectively. With more read retries, GraphAccel excels by efficiently scheduling requests to maximize hardware utilization. When certain chips experience delayed latency, GraphAccel dynamically mitigates the impact of these delays through effective resource management.

In addition to reducing latency, GraphAccel-Search also improves recall. This improvement can be attributed to the increased number of node expansions in GraphAccel for the same search list length. By assigning a lower priority to additional node expansions, GraphAccel accelerates transitions to subsequent iterations of node expansion. As a result, GraphAccel-Search more frequently expands the most relevant portions of the search graph using the same resources, rather than waiting to fill the search list with less important nodes.

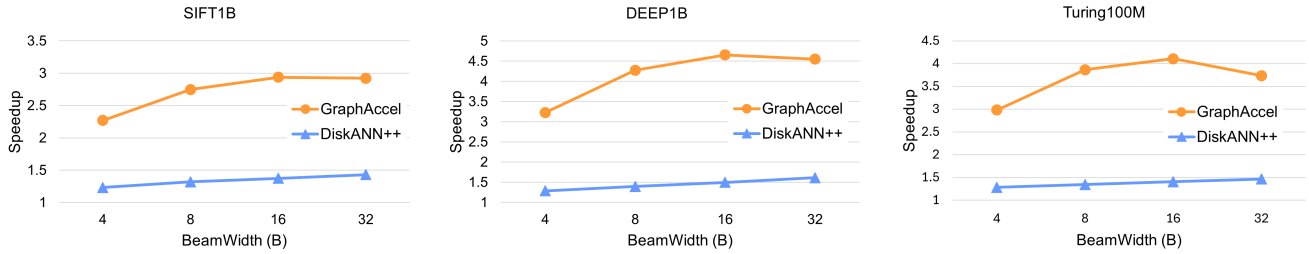
Effect of Beamwidth. As shown in Fig 7, as the beamwidth increases, more requests are processed simultaneously, allowing GraphAccel to significantly outperform DiskANN and DiskANN++. The slight decline in performance speedup observed as the beam size increases from 16 to 32 is attributed to the limitations of hardware resources. Our simulation setup includes 8 channels, each configured with 4 chips, resulting in a total of 32 chips available to handle concurrent requests. When the beam size remains within the hardware's parallelism limitations, increasing the beam size enables our GraphAccel to achieve significant performance improvements across all datasets. This improvement is attributed to its ability to effectively prioritize primary requests while discarding additional speculative requests that might otherwise increase the overall latency.

Page Accesses. Figure 8 depicts the average number of accessed pages per query normalized to DiskANN across different datasets, including SIFT1B, DEEP1B, and Turing100M. While DiskANN++



(a) (b) (c)

Fig. 6: Latency (microseconds) vs. Recall@100 for DiskANN, DiskANN++, and GraphAccel: (a) Retention age of 0 months, (b) Retention age of 3 months, (c) Retention age of 6 months.



(a) (b) (c)

Fig. 7: Speedups achieved with varying Beamwidth (B) compared to DiskANN across datasets: (a) SIFT1B, (b) DEEP1B, (c) Turing100M

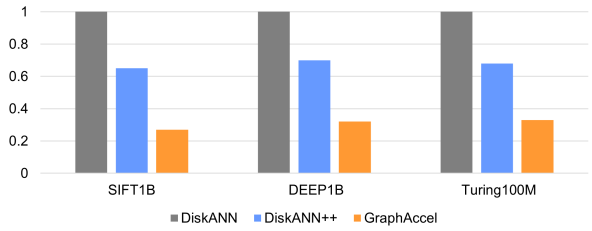


Fig. 8: Normalized Average Number of Flash Page Accesses per Query (Relative to DiskANN)

outperforms DiskANN due to its implementation of Starpacking, it still requires accessing a substantial number of pages. In contrast, GraphAccel demonstrates approximately 69.4% fewer accessed pages per query across all three datasets compared to DiskANN, up to 58.4% to DiskANN++. This result highlights the versatility of our GraphAccel-packing method, demonstrating its effectiveness across datasets with diverse characteristics.

VI. Related Work

VStore is an in-storage, graph-based vector search accelerator that leverages hardware optimizations, query orchestration, and result reuse strategies [23]. However, its performance heavily depends on temporal locality between multiple queries. In scenarios where queries are less predictable or exhibit poor locality, its effectiveness may be significantly reduced.

Pyramid is an in-storage ANN search acceleration platform with a hierarchical memory architecture [24], offering more than an order-of-magnitude speedup compared to CPU- or GPU-based systems. However, its design requires storing the compacted graph in memory, limiting the graph size to the capacity of the main memory. In contrast, GraphAccel stores all graph data on storage, eliminating graph size constraints except those imposed by storage capacity.

Kim et al. and Tian et al. have developed ANN search systems utilizing computational storage devices, specifically SmartSSDs [25], [26]. These systems achieve scalability by distributing the search workload across multiple SmartSSD units. However, due to their reliance on commercial platforms with fixed SSD controllers, their scheduling lacks the aggressive optimizations in GraphAccel's speculative search mechanism.

VII. Conclusion

In this paper, we introduced GraphAccel, an in-storage accelerator tailored for efficient graph-based vector similarity search. To address key challenges associated with large-scale searches on SSDs, GraphAccel incorporates an optimized page packing mechanism to reduce SSD page accesses and a speculative search strategy to fully exploit SSD parallelism. These techniques collectively minimize search latency while maintaining high recall accuracy, even under conditions of variable SSD read latency.

Our evaluation on large-scale datasets, including Turing100M, SIFT1B, and DEEP1B, demonstrates the significant performance gains of GraphAccel over state-of-the-art methods like DiskANN and DiskANN++. Moreover, GraphAccel's dynamic resource utilization and speculative execution strategies ensure resilience to retention-age-induced read disturbances, further underscoring its suitability for large-scale and latency-sensitive applications.

Acknowledgments

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grants funded by the Korea government (MSIT) (No. RS-2022-II221199, No. RS-2024-00398745, and No. RS-2024-00454666) and in part by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00251287).

REFERENCES

[1] A. Sharif Razavian *et al.*, "CNN features off-the-shelf: An astounding baseline for recognition," in *CVPR Workshops*, 2014.

[2] D. Cer, "Universal sentence encoder," *arXiv preprint arXiv:1803.11175*, 2018.

[3] T. Bertin-Mahieux *et al.*, "The million song dataset," 2011.

[4] J. Jo *et al.*, "PANENE: A progressive algorithm for indexing and querying approximate k-nearest neighbors," *IEEE TVCG*, vol. 26, no. 2, pp. 1347–1360, 2020.

[5] W. Li *et al.*, "Approximate nearest neighbor search on high dimensional data - experiments, analyses, and improvement," *IEEE TKDE*, vol. 32, no. 8, pp. 1475–1488, 2020.

[6] Q. Wang *et al.*, "Fast-adapting and privacy-preserving federated recommender system," *The VLDB Journal*, 2022.

[7] J. Wang *et al.*, "Scalable k-NN graph construction for visual descriptors," in *CVPR*, 2012.

[8] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *NeurIPS*, 2020.

[9] S. J. Subramanya *et al.*, "DiskANN: Fast accurate billion-point nearest neighbor search on a single node," in *NeurIPS*, 2019.

[10] Y. A. Malkov *et al.*, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE TPAMI*, vol. 42, no. 4, p. 824–836, Apr. 2020.

[11] H. Zhang *et al.*, "Generic intent representation in web search," in *SIGIR*, 2019.

[12] M. D. Hervé Jégou, Romain Tavenard *et al.*, "Searching in one billion vectors: re-rank with source coding," in *ICASSP*, 2011.

[13] A. B. Yandex *et al.*, "Efficient indexing of billion-scale datasets of deep descriptors," in *CVPR*, 2016.

[14] J. Ni *et al.*, "DiskANN++: Efficient page-based search over isomorphic mapped graph index using query-sensitivity entry vertex," 2023, arXiv:2310.00402.

[15] Y. Cai *et al.*, "Read disturb errors in mlc nand flash memory: Characterization, mitigation, and recovery," in *DSN*, 2015.

[16] Y. Luo *et al.*, "Improving 3d nand flash memory lifetime by tolerating early retention loss and process variation," *Proc. ACM Meas. Anal. Comput. Syst.*, 2018.

[17] J. Park *et al.*, "Reducing solid-state drive read latency by optimizing read-retry," in *ASPLOS*, 2021.

[18] P. Kumari *et al.*, "Radiation-induced error mitigation by read-retry technique for mlc 3-d nand flash memory," *IEEE TNS*, 2021.

[19] M. Ye *et al.*, "Achieving near-zero read retry for 3d nand flash memory," in *ASPLOS*, 2024.

[20] C. S. Hervé Jégou, Matthijs Douze, "Product Quantization for Nearest Neighbor Search," in *IEEE TPAMI*, 2011.

[21] G. M. c *et al.*, "Partitioning trillion-edge graphs in minutes," in *IPDPS*, 2017.

[22] A. Tavakkol *et al.*, "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices," in *FAST*, 2018.

[23] S. Liang *et al.*, "VStore: in-storage graph based vector search accelerator," in *DAC*, 2022.

[24] Z. Zhu *et al.*, "Processing-in-hierarchical-memory architecture for billion-scale approximate nearest neighbor search," in *DAC*, 2023.

[25] J.-H. Kim *et al.*, "Accelerating large-scale graph-based nearest neighbor search on a computational storage platform," *IEEE TC*, vol. 72, no. 1, pp. 278–290, 2023.

[26] B. Tian *et al.*, "Scalable billion-point approximate nearest neighbor search using SmartSSDs," in *USENIX ATC 24*, 2024.

RADiT: Redundancy-Aware Diffusion Transformer Acceleration Leveraging Timestep Similarity

Youngjun Park†, Sangyeon Kim‡, Yeonggeon kim‡, Gisan Ji‡, and Sungju Ryu*
†Department of Semiconductor Engineering, ‡Department of Electronic Engineering,
*Department of System Semiconductor Engineering,
Sogang University, Seoul, Republic of Korea
{youngjun, sangyeonkim, yeonggeonkim, gisanji, sungju}@sogang.ac.kr

***Abstract*—Diffusion Transformers (DiTs) have demonstrated unprecedented performance across various generative tasks including image and video generation. However, a large amount of computations on the inference process and iterative sampling steps in the DiT models result in high computational costs, leading to substantial latency and energy consumption challenges. To address these issues, we propose a redundancy-aware DiT (RADiT), a novel software-hardware co-optimization accelerator for DiTs that minimizes redundant operations in the iterative sampling stages. We identify data redundancy by evaluating blockwise input features and skip redundant computations by reusing results from consecutive timesteps.**

Furthermore, to minimize accuracy degradation and maximize computational efficiency, the Dynamic Threshold Scaling Module (DTSM) and Compress and Compare Unit (CCU) are employed in the redundancy detection process. This approach enables DiTs to achieve up to 1.8× and 1.7× faster speeds for image and video generation, respectively, without compromising quality, along with 41% and 45.5% reductions in energy consumption. Our RADiT scheme improves throughput by 1.67× and 1.76× for image and video generation tasks, respectively, while maintaining output quality and significantly reducing energy consumption.

***Index Terms*—Diffusion model, transformer, neural network, timestep similarity, hardware accelerator.**

I. Introduction

Diffusion models [1], [2] have achieved significant advancements in generative models and are widely adopted in various tasks including image [3], text-to-image [4], and video generation [5], [6]. These models generate high-quality samples by progressively denoising data during the inference process. Diffusion models have advantages over traditional generative models, such as generative adversarial networks (GANs) [7] and variational autoencoders (VAEs) [8] in terms of stable training and high sample quality, thereby leading to active research and increased attention in the generative models.

Recently, diffusion transformers (DiTs) [9] such as OpenAI's Sora [10] utilize transformers [11] as a backbone by integrating transformer-based architectures into diffusion models to enable more sophisticated and complex generative tasks instead of convolutional approaches. However, the computational complexity of DiT models is much larger than previous U-net based models [1], [4], [12], requiring substantial processing costs for large-scale data, and hence the real-time computation is one of the most important challenges. Recent studies on accelerating diffusion models [13]–[16] focus on leveraging the similarity in iterative sampling processes. However, conventional approaches have been targeting U-net based diffusion models and not well-suited for emerging DiT architectures. To overcome such challenges, we propose a software-hardware co-design method called RADiT which leverages timestep similarity in the DiT neural networks. Our RADiT first evaluates the block-level redundant computation on the input features in the consecutive timesteps. For the similar output features in the consecutive timesteps, we only compute once and reuse the result of the previous timestep for the following timestep computation. However, there is still a concern for the accuracy drop with such an approximate computing method. To minimize the accuracy drop, dynamic threshold scaling module (DTSM) flexibly adjusts the redundancy threshold criteria for every input data in real time. Furthermore, we design the compress and compare unit (CCU) to reduce the computing overhead of timestep similarity using low-bit compression of the feature map buffer. Our main contributions are as follows.

1) We analyze the redundancy in block-level operations across consecutive timesteps in DiTs and propose a redundancy-aware algorithm to skip the redundant computations.
2) To dynamically adjust the thresholding level for the evaluation of the timestep similarity, We propose a dynamic threshold scaling which efficiently accesses the redundancy across various timesteps, transformer blocks, and input data.
3) We introduce a compress and compare scheme to enable fast redundancy detection and minimize the computing overhead with low-bit data.
4) We finally propose a novel hardware architecture, RADiT, optimized with the proposed redundancy detection algorithm to accelerate DiTs effectively.

II. Background and Motivation

A. Diffusion Transformers

The diffusion model's fundamental idea is to gradually add noise to the original image. Then, the original image can be sampled by removing the noise step by step. Adding noise to

*Corresponding Author

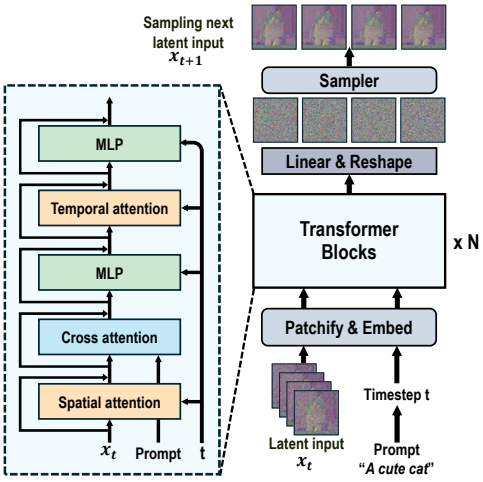


Fig. 1. DiT architecture for image and video generation.

the original image x_0 over T timesteps is called the forward process. The forward process is defined as follows.

$$q(x_t|x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I\right) \quad (1)$$

Where β_t is a variable that controls the noise, increasing as t approaches T. Conversely, removing the added noise step by step is called the reverse process. The reverse process is defined as follows.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)\right) \quad (2)$$

μ_θ and Σ_θ represent the mean and variance of the predicted noise as estimated by the model. Previously, a U-Net-based architecture was adopted as the model for predicting noise. However, transformer-based architecture DiT has emerged, achieving state-of-the-art performance across various generative models.

Fig. 1 illustrates the DiT architecture for video generation [17]. DiT comprises three main components: an embedding module for input data, timesteps, and text prompts; transformer blocks for noise prediction at specific timesteps; and a sampler that generates the latent input for the next timestep based on the predicted noise distribution. The transformer blocks are composed of attention blocks and multi-layer perceptrons (MLPs) to capture relationships between tokens embedded at the patch level. Through transformer blocks, DiT can capture global relationships in the data, which helps it recognize overall patterns. However, since DiT uses an attention mechanism, it has higher computational complexity and memory requirements compared to U-Net-based models. Therefore, we have focused on reducing computational load and accelerating the processing speed of DiT.

B. Observations on the Redundancy of DiT

The most important characteristic of the diffusion model is that it iterates the model over multiple timesteps. In other words, because it removes noise gradually, we predicted that features in the same block would be similar across adjacent timesteps. In this work, we analyzed whether this similarity exists during the inference process of DiT. We examined how much the input features of each block differs from (similar to) the input of previous timestep at all timesteps. The results are shown in Fig. 2a. Additionally, we analyzed the attention scores of two blocks at adjacent timesteps with similar input features (Fig. 2b). From this analysis, we observed the following characteristics. 1) During the inference process of the DiT models, redundancy in the input of blocks appears across adjacent timesteps. 2) This redundancy varies depending on the block and timestep, as well as on the characteristics of the input data. 3) Attention/MLP blocks with similar input tend to have similar output.

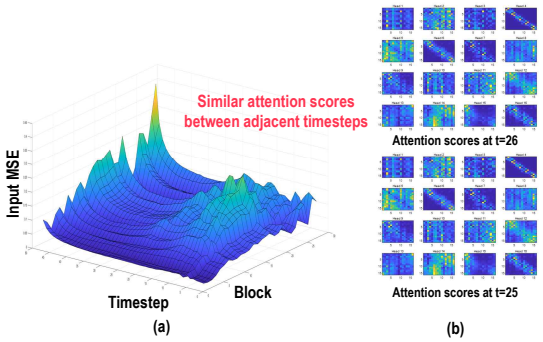


Fig. 2. The results of analyzing the redundancy inherent in DiTs. (a) Input MSE of transformer blocks over timestep. (b) Attention scores across adjacent timesteps.

Based on these observations, we aim to accelerate DiT-based models by skipping computation where each block's input is similar to that of the previous timestep and by reusing the results from the previous timestep. However, simply skipping and reusing in regions with frequent similar inputs significantly degrades model performance. This degradation is caused by errors from reusing previous results, which are amplified by the residual connections in DiT as the inference progresses. To mitigate such an accuracy drop, it is necessary to check the similarity of each block and verify if there are any errors caused by the reuse of the result data. As a result, we introduce software-hardware co-optimization approach called RADiT which will be explained in the following Section III.

III. Proposed Work: RADiT

In this section, we introduce our proposed method called RADiT, software-hardware co-design approach. First, we provide an overview of the overall operation of RADiT in Section III-A. Next, we explain the detailed methods employed by RADiT for redundancy-aware network compression in Section III-B. Finally, in Section III-C, we describe the overall hardware architectural design of RADiT.

A. Design Overview of RADiT

Fig. 3 illustrates the overall operation of RADiT. The main concept of RADiT is to reduce redundant computations during iterative sampling by analyzing the input to a DNN operation block such as self-attention, cross-attention, and MLPs. If the input is considered similar to the previous timestep, the computation is skipped, and the output from the block in the previous timestep is reused. This approach minimizes redundant operations and accelerates the sampling process. To enable this functionality, RADiT satisfies the following two requirements. First, we need to set a threshold to determine

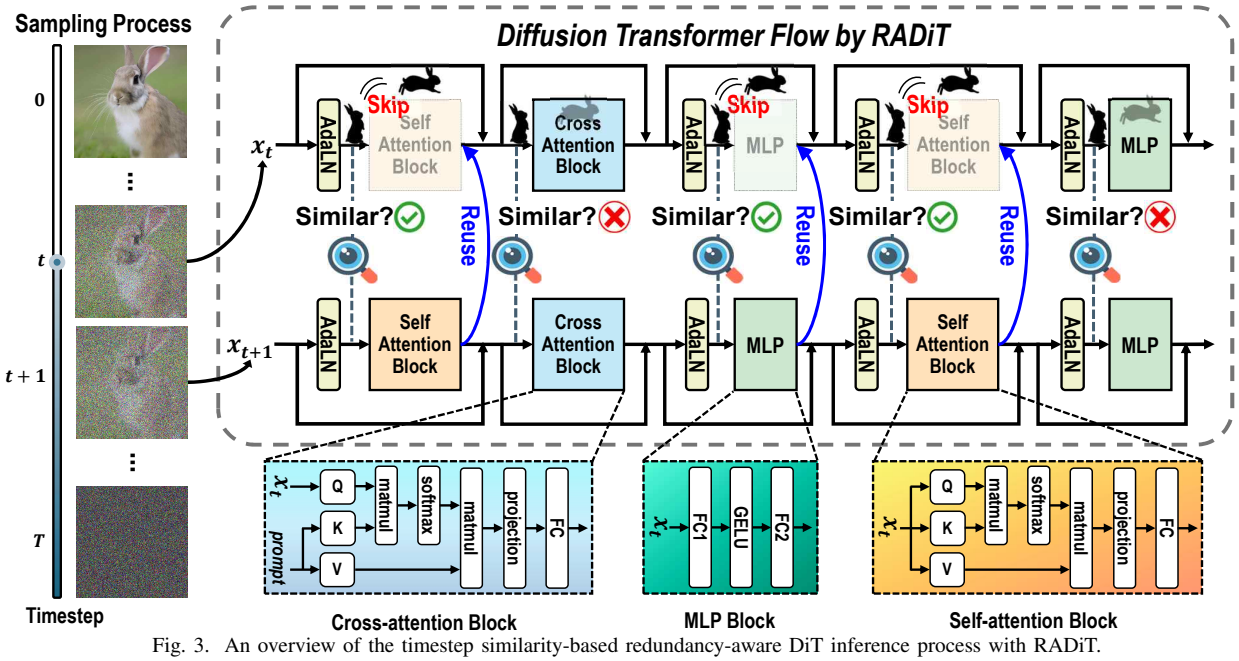


Fig. 3. An overview of the timestep similarity-based redundancy-aware DiT inference process with RADiT.

whether the input is similar to the previous timestep. As shown in Fig. 2a, differences in input across adjacent timesteps vary for every block, and these differences change dynamically during each inference depending on the input data. Therefore, it is crucial to dynamically adjust the similarity threshold for each block during inference to ensure consistent samples that closely resemble the original. This aspect will be explained in more detail in Section III-B2. Second, the comparison between the current and previous timestep inputs must be fast enough. This comparison occurs during the DNN operation at each timestep of the diffusion models. If the latency of the comparison process is significantly large, the overhead may outweigh the benefits of skipping computations. Therefore, directly loading raw previous timestep inputs and calculating differences using metrics like mean squared error (MSE) is not practical. Instead, RADiT employs a method that first compresses the data bits, and then it detects differences by comparing the compressed lower bits. This approach reduces the size of the previous timestep data that needs to be loaded from memory by 4× for FP16. Further details on this mechanism are provided in Section III-B3.

B. Redundancy Aware Network Compression

This Section introduces our various techniques for compressing diffusion models through the redundancy-aware algorithm optimized by RADiT.

1) Redundancy-Aware Algorithm: We analyze the redundancy in the DiT models and propose a redundancy-aware algorithm (Algorithm 1) to minimize redundant computations.

Let the input to the n-th DNN operation block at an arbitrary timestep t be denoted as x_t^n and its output as $F(x_t^n)$, and please note that the denoising process of the diffusion model starts from timestep T to 0. If the input x_{t+1}^n to the same n-th block at the previous timestep $t+1$ is similar to x_t^n, we can approximate $F(x_t^n)$ to $F^*(x_t^n) = F(x_{t+1}^n)$. A key aspect to note here is that we skip not individual operations like matrix multiplication but entire blocks, such as attention and MLP blocks. Thus, the redundancy-aware algorithm significantly reduces latency by skipping DNN computation, weight loading required for computation, memory read/write operation for activation and attention scores, and additional functions like reshaping and normalization.

Algorithm 1: Redundancy-Aware Computing

Input: Total timesteps T, Depth of block N, Block input x_t^n, Block threshold θ^n, DNN operation block $F(\cdot)$, Timestep difference function $f(\cdot)$
Output: Updated outputs $F(x_t^n)$ for all t and n

```
for t ← T to 1 do
    for n ← 1 to N do
        Δ(x_t^n) ← f(x_{t+1}^n, x_t^n);
        if Δ(x_t^n) < θ^n then
            Reuse: F(x_t^n) ← F*(x_t^n) = F(x_{t+1}^n);
        else
            Compute: F(x_t^n) ← F(x_t^n);
        end
    end
end
```

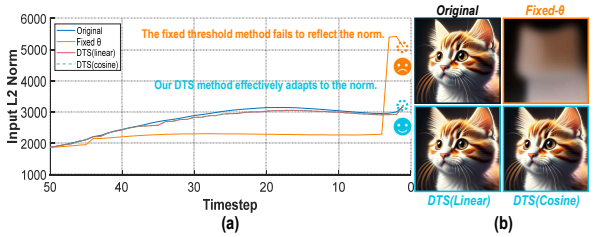


Fig. 4. (a) Changes in input L2 norm across timesteps. (b) The image sampling results by each method. θ is threshold value.

2) Dynamic Threshold Scaling Method: Reusing computation results from the previous block is a highly effective method for reducing latency, but it introduces errors in similarity evaluation, which worsen with increasing block depth.

Algorithm 2: Dynamic Threshold Update for Timestep Differences

Input: Total timesteps T, Depth of blocks N, Timestep differences Δx_t^n, Parameter α_t
Output: Updated thresholds θ^n for all n
for $t \leftarrow T$ ***down to*** 1 **do**
  for $n \leftarrow 1$ ***to*** N **do**
    if θ^n *is* ***None*** **then**
      Initialize: $\theta^n \leftarrow \Delta x_t^n$;
    else
      $\Delta(\Delta x_t^n) \leftarrow \Delta x_t^n - \Delta x_{t+1}^n$;
      if $\Delta(\Delta x_t^n) > 0$ ***and*** $\Delta(\Delta x_{t+1}^n) < 0$ **then**
        Update: $\theta^n \leftarrow \Delta x_{t+1}^n + \alpha_t \cdot \Delta(\Delta x_t^n)$;
      else
        Keep: θ^n;
      end
    end
  end
end

Therefore, using a fixed threshold exacerbates this issue in later blocks (Fig. 4), leading to excessive reuse and compounding errors. To address this, we propose the dynamic threshold scaling (DTS) method, as described in Algorithm 2.

For subsequent timesteps, the algorithm calculates the change in timestep difference, $\Delta(\Delta x_t^n) = \Delta x_t^n - \Delta x_{t+1}^n$. If the change indicates a significant variation ($\Delta(\Delta x_t^n) > 0$) and the previous difference is negative (($\Delta(\Delta x_{t+1}^n) < 0$), (this condition identifies a local minimum in Δx_t^n) the threshold is updated.

By using DTS, we can mitigate errors caused by reuse of block output. If significant reuse-induced errors occur in the previous step, the threshold is lowered, enforcing stricter similarity criteria in the subsequent timestep. Conversely, if errors are small, the threshold is increased, allowing for more aggressive data reuse. The parameter α_t in the DTS algorithm starts at 0 and increases to 1 as the timestep t decreases from T to 0. This design is inspired by β_t (Equation 1), which determines the amount of noise added during the forward process. At the initial timestep ($t = 0$), less noise is added, and as t approaches T, the noise decreases further. Similarly, during the inference (reverse process), redundancy is generally lower at the beginning and increases toward the end as in our analysis (Fig. 4). Based on this observation, we define α_t using two approaches: linear scheduling (Equation 3) and cosine scheduling (Equation 4).

$$\alpha_t = \frac{t - T}{T} \tag{3}$$

$$\alpha_t = \frac{1}{2}\left(1 - \cos \pi \left(\frac{T - t}{T}\right)\right) \tag{4}$$

Fig. 5a shows the timestep difference when similarity is computed using a **fixed threshold** in the later blocks. Due to the reuse of previous block results with the fixed threshold, the step difference becomes small in most timesteps, leading to excessive reuse. However, as shown in Fig. 4, this results in a significant loss of original consistency, causing a severe degradation in the accuracy.

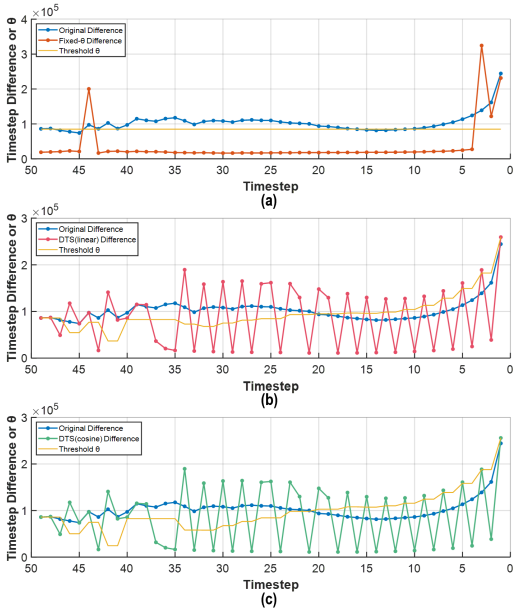


Fig. 5. Variation in differences of block outputs between adjacent timesteps. Original and reused results using (a) fixed threshold, (b) linear scheduling, and (c) cosine scheduling. θ is threshold.

On the other hand, Figs. 5b-c illustrate the timestep difference of the inputs when each of the two **dynamic threshold** scaling method is applied depending on the input similarity. These dynamic thresholding approaches track the original difference better than the fixed thresholding method. Consistently, Fig. 4 demonstrates that the DTS method preserves the original consistency, allowing for faster inference without significantly compromising model performance.

3) Efficient Bit Compression and Comparing: To apply the algorithms discussed so far, the input x_{t+1}^n of the previous timestep must be read before each DNN operation, leading to a latency overhead. To minimize this overhead, we propose an effective compression method for DiTs, which reduces input bits to 4 bits while maintaining comparability.

The adaptive layer normalization [18] in front of each DNN operation block of DiTs (Fig. 3) normalizes the input data and scales and shifts it based on the current timestep information. Due to the normalization layers, our evaluation (Fig. 6a) shows that input activation values range from -16-to-16. Based on the analysis, we propose an efficient compression method tailored to the distribution of input values.

Fig. 6b illustrates the proposed compression method. Assuming the input data is in FP16 format as baseline DiT used, the first and second bits of the exponent field are used as indicator bits. Based on these indicator bits, the 16-bit data is compressed into 4 bits using one of the following three cases: **Case 1 (Indicator Bits = 00):** When the indicator bits are 00, the compressed result is derived from input_data$[14:12]$, and the most significant bit (MSB) is set to 0. This case uses 2 compressed levels: 2^{-11}-to-2^{-7} are mapped to 0001, and values below 2^{-11} are mapped to 0000. **Case 2 (Indicator Bits = 01):** When the indicator bits are 01, the compressed result is

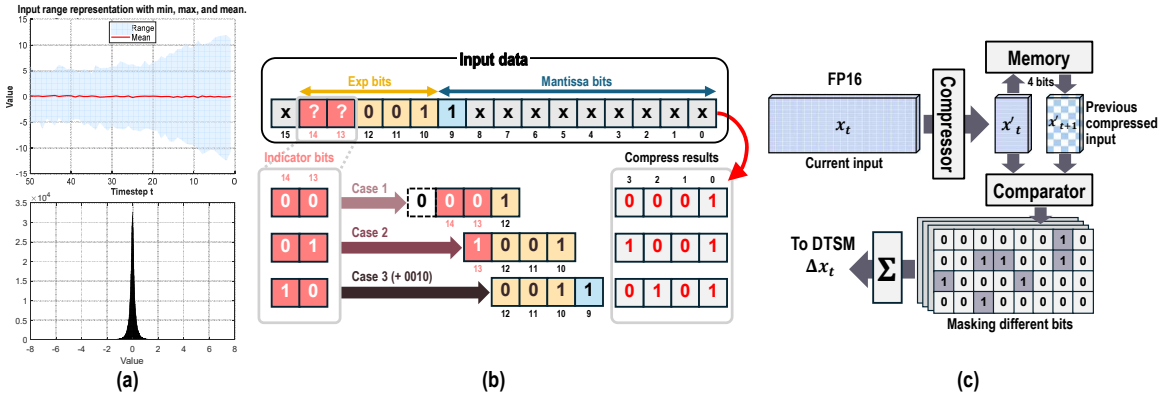


Fig. 6. Input compression method and comparing flow. (a) The range and distribution of input after adaptive layer normalization. (b) A method to compress input by dividing cases based on indicator bits. (c) A description of the process to compress the input, perform a comparison, and estimate the difference.

derived from input_data[13 : 10]. This case uses 8 compressed levels: 2^{-7}-to-2^{-6}, 2^{-6}-to-2^{-5}, $\cdots$, 2^{0}-to-2^{1} are mapped to 1000, 1001, $\cdots$, 1111, respectively. **Case 3 (Indicator Bits = 10):** When the indicator bits are 10, the compressed result is derived from input_data[12 : 9]+0010, where the input_data[9] is MSB of the mantissa bits. Therefore, this case uses 6 compressed levels: 2^{1}-to-3, 3-to-2^{2}, 2^{2}-to-6, 6-to-2^{3}, 2^{3}-to-12, and 12-to-2^{4} are mapped to 0010, 0011, 0100, 0101, 0110, and 0111, respectively. Sign is not considered, as it is rare for the sign to change significantly between adjacent timesteps.

Fig. 6c demonstrates the process of calculating the difference based on the proposed bit compression technique. The input x_t at the current timestep t is compressed into x'_t, and it is compared with the compressed x_{t+1} from the previous timestep. The differing bits are masked as 1, while the same bits are set to 0. By summing all masked 1s, the difference Δx_t between the inputs of two adjacent timesteps is obtained.

C. RADiT Hardware Accelerator Design

Fig. 7 illustrates the architecture of the DiT accelerator, RADiT, which incorporates the methods discussed above. RADiT consists of the following key components: 1) A *dynamic threshold scaling module (DTSM)* that applies the DTS method to appropriately scale the threshold value. 2) A *compress and compare unit (CCU)* that efficiently compresses input data and compares feature similarity between adjacent timesteps. 3) A systolic array [19] consisting of 64×64 processing elements (PEs) for performing DNN operations. These components work together to accelerate DiT operations by achieving speedup, reducing computational overhead, and maintaining

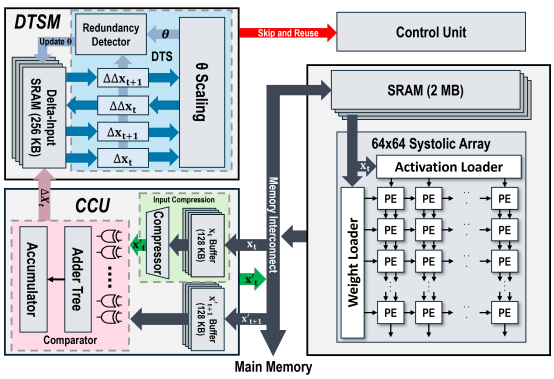


Fig. 7. Hardware architecture of RADiT.

TABLE I
COMPARED TO THE PREVIOUS U-NET-BASED DIFFUSION ACCELERATORS, OUR DESIGN SUPPORTS DIT ACCELERATION AND MULTIPLE TASKS.

Previous Work	Supporting Model	Evaluation Task
ISSCC'24 [15]	U-Net	Image only
VLSI'24 [16]	U-Net	Image only
ISCA'24 [14]	U-Net	Image only
DAC'24 [20]	U-Net	Image only
Ours	Transformer (DiT)	Image and video

model consistency. While the systolic array performs DNN operations for the DiT models, the CCU compresses the current input feature and compares it with the compressed input feature from the previous timestep. The comparison results are passed to the DTSM, which sets the threshold using Algorithm 2. If the similarity between two timesteps calculated by the CCU is smaller than the threshold determined by the DTSM, the redundancy detector in DTSM generates a *skip-and-reuse* signal, which is sent to the control unit. Based on this signal, the control unit skips the current operation and reuses the result from the previous computation. This mechanism enables RADiT to continuously identify redundancy between adjacent timesteps and skip unnecessary computations, thus improving operational efficiency.

IV. RESULTS

A. Experimental Setup

In this section, we present the evaluation results of the RADiT hardware architecture. The image generation capability of RADiT was assessed using the DiT [9] model, while its video generation capability was analyzed with the Latte [21] model. We implemented behavioral models of both the baseline and RADiT hardware accelerators in Verilog HDL. The design was synthesized in a 28nm targeting 500MHz. An LPDDR3 model was used for the DRAM performance evaluation. Previous diffusion hardware accelerators were dedicated for convolution-based DNNs only (Table I), so we compared our performance with vanilla DiT hardware baseline without timestep similarity.

B. Results

1) Area: Fig. 8 provides detailed area breakdown on the RADiT architecture. A large portion of the RADiT area is occupied by the systolic array [19], which is designed for DNN operations and consists of 64×64 processing element (PE) arrays. In the systolic array, the weight and psum(+input)

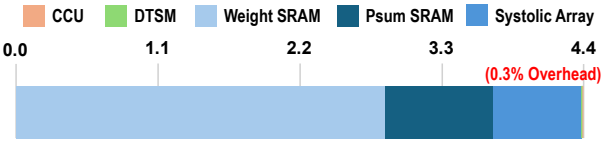


Fig. 8. The area breakdown of the proposed RADiT architecture. CCU and DTSM are overheads compared to the baseline.

SRAMs account for approximately 85% of the total area, while the logic components take up around 15%. The CCU and DTSM responsible for the redundancy aware algorithm require area overhead, but the additional area occupies only 0.3% of the total area, which is negligible.

2) Accuracy: To evaluate the performance of RADiT's redundancy-aware algorithm on the DiTs, we conducted experiments on the following three tasks: image generation, video generation, and text-to-video generation. For image generation, we used the DiT-XL/2 [9] model trained on the ImageNet [22] dataset to generate 50,000 images with a resolution of 256×256 for comparison. We evaluated image quality using widely used metrics such as FID [23] and Inception Score [24]. For video generation, we utilized the Latte [21] model trained on four public datasets-Faceforensics [25], SkyTimelapse [26], UCF101 [27], and Taichi-HD [28]-to generate 2,000 videos with a duration of 2 seconds, resolution of 256×256, and a frame rate of 16 fps. The performance was measured using the FVD [29]. For text-to-video generation, we used the Latte-1 model and evaluated the video quality with VBench [30], which measures the alignment between the generated video and the given prompt. 945 prompts were used to generate 5 videos per prompt, each with a duration of 2 seconds, resolution of 512×512, and a frame rate of 16 fps. The detailed results for each task are presented in Tables II, III, and IV. As a result, RADiT achieved approximately 1.7x Tflops reduction in image generation, 1.8x in video generation, and 1.6x in text-to-video generation, with minimal degradation in quality in most cases. In some cases, our RADiT approach shows better accuracy value than the baseline. We carefully analyze that non-critical redundant features have been eliminated via our redundancy-aware computing, thereby slightly improving the accuracy value in the cases.

3) Latency and Energy: Fig. 9 and Fig. 10 illustrates the normalized latency and energy results for three tasks: image generation, video generation, and text-to-video tasks, evaluated across three models. RADiT achieves significant latency

TABLE II
256x256 IMAGE GENERATION RESULTS.

Method	FID↓	sFID↓	IS↑	Precision↑	Recall↑	Gflops↓
DiT-XL/2 [9]	2.27	4.60	278.24	0.83	0.57	119
RADiT(linear)	2.57	5.38	264.17	0.81	0.59	70.2
RADiT(cosine)	2.68	5.49	261.92	0.81	0.59	71.4

TABLE III
FVD(↓) OF 256X256 VIDEO GENERATION ON DIFFERENT DATASETS.

Method	FaceForensics	SkyTimelapse	UCF101	Taichi-HD	Gflops↓
Latte [21]	30.69	46.43	150.74	108.14	5572
RADiT(linear)	30.72	47.98	146.75	116.64	3036.7
RADiT(cosine)	31.08	48.68	162.08	120.17	3176.0

TABLE IV
VBENCH SCORES OF 512X512 TEXT-TO-VIDEO GENERATION RESULTS.

Method	Quality score↑	Semantic score↑	Total score↑	Tflops↓
Latte-1 [21]	79.72	67.58	77.29	3439.47
RADiT(linear)	80.29	67.49	77.73	2235.65
RADiT(cosine)	80.47	68.02	77.98	2270.05

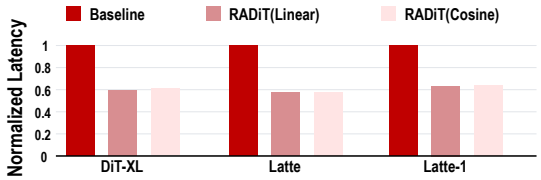


Fig. 9. The comparison of normalized latency between the baseline and RADiT for three DiT-based models.

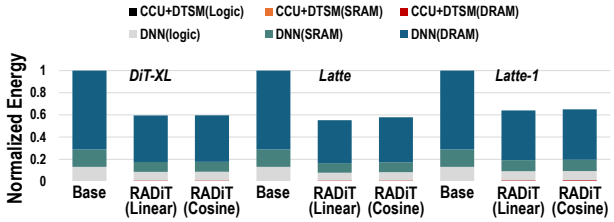


Fig. 10. The comparison of normalized energy consumption between the baseline and RADiT for three DiT-based models.

reductions of 40%, 43.1%, and 36.5% (Fig. 9) for the three tasks, respectively, due to its operation-skipping mechanism. The additional DRAM accesses required for comparisons with previous steps introduce an overhead of only 1.5%, which is negligible. So, our design still shows much smaller latency than the baseline thanks to the block-level computational skipping.

Energy consumption shows a similar trend (Fig. 10), with reductions of 41%, 45.5%, and 38.4% for the three tasks, respectively, and energy consumption due to the DRAM access is dominant part. These results demonstrate that RADiT effectively reduces inference time while maintaining low overhead, making it a highly efficient solution for latency and energy optimization.

V. CONCLUSION

We proposed RADiT, a redundancy-aware diffusion transformer hardware accelerator. RADiT introduces a software-hardware co-optimization approach that dynamically identifies and skips redundant computations, significantly reducing latency and energy consumption during inference. Our experimental results demonstrated that RADiT achieves substantial speedups of 1.6-1.8×, respectively, with negligible accuracy drop and hardware overhead.

ACKNOWLEDGMENT

This work was partly supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00266, Development of Ultra-Low Power Low-Bit Precision Mixed-mode SRAM PIM, 30%), National Research Foundation (NRF) funded by the Korean government (MSIT) (No. RS-2024-00439520, 30%), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (RS-2025-02263669, Development of server-level DRAM-stacked PIM solution for accelerating ultra-large AI models, 30%), and Korea Institute for Advancement of Technology(KIAT) grant funded by the Korea Government(MOTIE) (P0017011, HRD Program for Industrial Innovation, 10%). The EDA tool was supported by the IC Design Education Center(IDEC), Korea.

REFERENCES

[1] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[2] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*. pmlr, 2015, pp. 2256–2265.

[3] B. Kawar, M. Elad, S. Ermon, and J. Song, "Denoising diffusion restoration models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 23 593–23 606, 2022.

[4] P. Dhariwal and A. Nichol, "Diffusion models beat gans on image synthesis," *Advances in neural information processing systems*, vol. 34, pp. 8780–8794, 2021.

[5] A. Blattmann, T. Dockhorn, S. Kulal, D. Mendelevitch, M. Kilian, D. Lorenz, Y. Levi, Z. English, V. Voleti, A. Letts *et al.*, "Stable video diffusion: Scaling latent video diffusion models to large datasets," *arXiv preprint arXiv:2311.15127*, 2023.

[6] J. Ho, W. Chan, C. Saharia, J. Whang, R. Gao, A. A. Gritsenko, D. P. Kingma, B. Poole, M. Norouzi, D. J. Fleet, and T. Salimans, "Imagen video: High definition video generation with diffusion models," *ArXiv*, vol. abs/2210.02303, 2022.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[8] D. P. Kingma, M. Welling *et al.*, "Auto-encoding variational bayes," 2013.

[9] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 4195–4205.

[10] Y. Liu, K. Zhang, Y. Li, Z. Yan, C. Gao, R. Chen, Z. Yuan, Y. Huang, H. Sun, J. Gao, L. He, and L. Sun, "Sora: A review on background, technology, limitations, and opportunities of large vision models," 2024. [Online]. Available: https://arxiv.org/abs/2402.17177

[11] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[12] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2022, pp. 10 684–10 695.

[13] X. Ma, G. Fang, and X. Wang, "Deepcache: Accelerating diffusion models for free," *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15 762–15 772, 2023.

[14] W. Kong, Y. Hao, Q. Guo, Y. Zhao, X. Song, X. Li, M. Zou, Z. Du, R. Zhang, C. Liu, Y. Wen, P. Jin, X. Hu, W. Li, Z. Xu, and T. Chen, "Cambricon-d: Full-network differential acceleration for diffusion models," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 903–914.

[15] R. Guo, L. Wang, X. Chen, H. Sun, Z. Yue, Y. Qin, H. Han, Y. Wang, F. Tu, S. Wei, Y. Hu, and S. Yin, "20.2 a 28nm 74.34tflops/w bf16 heterogenous cim-based accelerator exploiting denoising-similarity for diffusion models," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 67, 2024, pp. 362–364.

[16] Y. Qin, Y. Wang, X. Yang, Z. Zhao, S. Wei, Y. Hu, and S. Yin, "A 52.01 tflops/w diffusion model processor with inter-time-step convolution-attention-redundancy elimination and bipolar floating-point multiplication," in *2024 IEEE Symposium on VLSI Technology and Circuits (VLSI Technology and Circuits)*, 2024, pp. 1–2.

[17] H. Lu, G. Yang, N. Fei, Y. Huo, Z. Lu, P. Luo, and M. Ding, "Vdt: General-purpose video diffusion transformers via mask modeling," 2023. [Online]. Available: https://arxiv.org/abs/2305.13311

[18] J. Xu, X. Sun, Z. Zhang, G. Zhao, and J. Lin, "Understanding and improving layer normalization," 2019. [Online]. Available: https://arxiv.org/abs/1911.07013

[19] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.

[20] Y. Jing, M. Wu, J. Zhou, Y. Sun, Y. Ma, R. Huang, T. Jia, and L. Ye, "Aig-cim: A scalable chiplet module with tri-gear heterogeneous compute-in-memory for diffusion acceleration," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. New York, NY, USA: Association for Computing Machinery, 2024. [Online]. Available: https://doi.org/10.1145/3649329.3657373

[21] X. Ma, Y. Wang, G. Jia, X. Chen, Z. Liu, Y.-F. Li, C. Chen, and Y. Qiao, "Latte: Latent diffusion transformer for video generation," *arXiv preprint arXiv:2401.03048*, 2024.

[22] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[23] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[24] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

[25] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "Faceforensics: A large-scale video dataset for forgery detection in human faces," *ArXiv*, vol. abs/1803.09179, 2018.

[26] W. Xiong, W. Luo, L. Ma, W. Liu, and J. Luo, "Learning to generate time-lapse videos using multi-stage dynamic generative adversarial networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2364–2373, 2017.

[27] K. Soomro, A. Zamir, and M. Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *ArXiv*, vol. abs/1212.0402, 2012.

[28] A. Siarohin, S. Lathuilière, S. Tulyakov, E. Ricci, and N. Sebe, "First order motion model for image animation," in *Neural Information Processing Systems*, 2020.

[29] T. Unterthiner, S. van Steenkiste, K. Kurach, R. Marinier, M. Michalski, and S. Gelly, "Fvd: A new metric for video generation," in *DGS@ICLR*, 2019.

[30] Z. Huang, Y. He, J. Yu, F. Zhang, C. Si, Y. Jiang, Y. Zhang, T. Wu, Q. Jin, N. Chanpaisit, Y. Wang, X. Chen, L. Wang, D. Lin, Y. Qiao, and Z. Liu, "Vbench: Comprehensive benchmark suite for video generative models," *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 21 807–21 818, 2023.

Hybrid Embedding Framework for Memory-Efficient Recommendation Systems

Seung Jin Yang
Seoul National University
Seoul, Republic of Korea
rkem1542@snu.ac.kr

Hyuk-Jae Lee
Seoul National University
Seoul, Republic of Korea
hyuk_jae_lee@capp.snu.ac.kr

Chae Eun Rhee
Hanyang University
Seoul, Republic of Korea
crhee@hanyang.ac.kr

***Abstract*—This study introduces a memory-efficient mixed representation for deep learning recommendation models (DLRM), addressing the embedding table memory bottleneck from growing data scale. By distinguishing between frequently accessed (hot) and infrequently accessed (cold) embeddings, we store hot embeddings in a compact table while representing cold embeddings using a deep hash embedding (DHE) network, significantly reducing memory usage. This hybrid approach performs table lookups for hot embeddings and parallelized computations for cold embeddings, minimizing training time while maintaining accuracy. Experimental results demonstrate that our method outperforms other embedding reduction techniques in memory efficiency, accuracy, and training speed in CPU-GPU hybrid environments.**

I. Introduction

Recommendation systems are widely used in various fields such as social networks, e-commerce platforms, online streaming services, and digital marketing [1], [2]. These systems analyze user behavior data, such as search history, purchasing patterns, and content preferences, to provide personalized recommendations and support decision-making. Meta's Deep Learning Recommendation Model (DLRM) [3] is one of the most widely used models, converting various categorical data into dense representations and storing them in an embedding table. As datasets continue to grow, more storage space is required for the embedding table to accommodate this increase [4], [5]. In industrial-scale implementations, the size of the embedding table can reach terabytes [6], [7], [8]. DLRM consists of four main operations: the bottom multi-layer perceptron (MLP), embedding lookup, feature interaction, and the top MLP. While the MLP is a compute-dominated task that can easily be accelerated using high-performance GPUs, the embedding lookup, which heavily depends on memory bandwidth, constitutes the largest portion of the total execution time due to memory bottlenecks [4], [9].

Vectors stored in the embedding table can be categorized into hot embeddings and cold embeddings based on their access frequency. While the threshold may vary depending on the context, hot embeddings are generally defined as the vectors that account for more than 80% of the total access frequency [10]. These hot and cold embeddings follow a long-tail distribution [11], [8], where hot embeddings, which account for most memory accesses, represent less than 10% of the entire dataset. On the other hand, the remaining 90% or more consists of cold embeddings [12]. In experiments with the Criteo Kaggle dataset, it was found that hot embeddings, which accounted for 80% of the total accesses, made up only 6.2% of the entire embedding [12]. Cold embeddings, while having a lower access frequency and minimal impact on model accuracy [13], represent a significant portion of the overall memory demand [11].

In this study, we propose a hybrid embedding framework that reflects the characteristics of hot and cold embedding vectors. Hot vectors, which have high access frequency, are stored directly in memory, while large-scale cold embedding vectors are replaced by neural representations. Instead of directly querying cold vectors from memory, they are computed through a compact deep hash embedding (DHE) network, effectively reducing memory requirements. The key benefits of replacing cold data with the DHE network are twofold: first, it allows for GPU acceleration, and second, the level of approximation can be flexibly adjusted through network design. However, the process of computing cold vectors using the DHE network takes more time than direct memory lookup, which introduces a trade-off between memory savings and increased computation time. To address this, the system handles hot vector retrieval and cold vector computation in parallel, minimizing execution time overhead. Compared to the traditional approach of processing all embedding data through table lookups, the proposed method reduces memory requirements by approximately 5%, while maintaining similar execution time and accuracy.

II. Background

A. Related Work

Research aimed at reducing the storage space of embedding vectors and accelerating lookup operations in DLRM has been ongoing, and these embedding compression techniques can be broadly classified into three categories: low-precision [14], [15], mixed-dimension [13], [16], [17], and hashing [18], [19] techniques. The low-precision method reduces memory usage and increases computational speed by representing embedding vectors, originally stored in 32-bit floating-point format, with fewer bits. However, it comes with the downside of accuracy degradation. The mixed-dimension method, unlike the traditional approach where all vectors are assigned the same dimension, adjusts dimensions based on data usage

frequency [13] or creates sparse embedding tables through pruning [16], [17]. This method can solve overfitting problems or provide high accuracy, but may incur additional costs in the training and inference process, making it inefficient. Hashing tricks [18] and compositional embedding [19] map large-dimensional token spaces to smaller dimensions using hash functions. While using a naive hash function can reduce memory requirements, it can lead to accuracy loss. To improve accuracy, the quotient-remainder trick can be used to divide the embedding table into two separate tables and combine the results, but this requires additional memory access and computation.

Conventional embedding compression techniques have had limitations in simultaneously reducing table size and maintaining model accuracy. To overcome this, a method was proposed where data is processed through computation instead of lookup. The DHE Network [20] is a model that combines hash embedding with an embedding network. It transforms the input index into a unique identifier through dense hash encoding, and then uses a deep embedding network to generate feature embeddings. DHE has been shown to achieve similar accuracy with approximately a quarter of the memory cost compared to models using traditional embedding tables, as demonstrated on the Movielens-20M [21] dataset.

Research that takes an architectural approach, rather than an algorithmic one, has proposed optimizing memory access performance by storing hot and cold embedding vectors in separate spaces. Frequently accessed hot vectors are placed in nearby high-speed memory, while infrequently used cold vectors are placed in remote memory, thereby improving overall memory performance [22], [23].

B. Problem Statements

In various studies aimed at reducing the storage space of embedding tables and shortening embedding lookup times, it has been found that deterministic methods cannot solve all the problems, and learning-based approaches introduce additional overhead. In particular, learning-based methods [19], [13], [16], [20] improve memory usage and model performance by utilizing adaptive hyperparameters that reflect the characteristics of embedding data. However, these methods are highly sensitive to the setting of hyperparameters, which can significantly affect performance. Additionally, when there is insufficient memory budget and the number of model parameters is inadequate, the accuracy can drop significantly compared to the original DLRM embedding approach.

The hybrid data embedding (HDE) approach proposed in this paper applies the DHE network only to cold data, thereby reducing memory requirements and minimizing accuracy loss due to approximation. While the proposed method of processing cold data with the DHE network involves a trade-off between reduced memory usage and increased execution time, it offers several key differentiators. The DHE network dedicated to cold data can be reduced in size, allowing for more aggressive approximations. Additionally, the DHE network can be executed in parallel with the table lookup

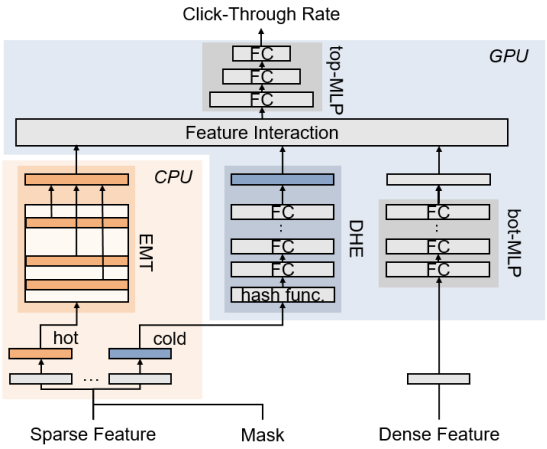
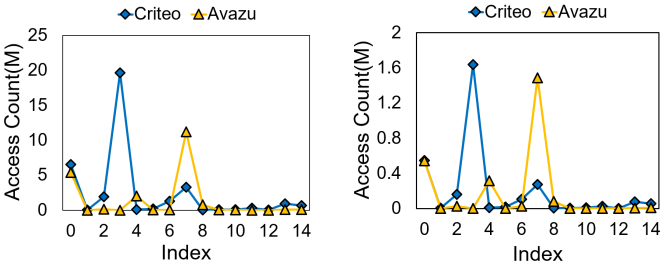


Fig. 1: Overall architecture of our proposed method, Hybrid Data Embedding(HDE).



(a) access count for train data (b) access count for test data

Fig. 2: Access count profiling on train/test data. The first 15 indices of the embedding table are used for profiling. The first and the third embedding table of Criteo and Avazu are used respectively.

of hot embedding data. When querying multiple embedding tables, hot and cold data are mixed, but by implementing cold embedding computations in parallel with the hot embedding table lookups, the computation time of the DHE network is hidden behind the table lookup time, maximizing efficiency. The proposed approach maintains model performance even with a small number of parameters, and it has the advantage of being less reliant on hyperparameters for performance.

III. Proposed Methods

A. Architecture Overview

Figure 1 shows the overall architecture of the proposed model, which follows a structure similar to the CPU-GPU hybrid architecture that utilizes both CPU and GPU [24], [12]. The model takes sparse features, dense features, and a mask as inputs. The mask contains information indicating whether the embedding vector referenced by the sparse feature is hot or cold. Based on the access count of the training data, the top 5% are classified as hot, while the remaining data is classified as cold. Figures 3(a) and (b) show the access count analysis of the Criteo [25] and Avazu [25] datasets, respectively, indicating that the access ratio of embedding data with the same index is similar in both the training and testing data. This suggests that data classified as hot in training can be assumed to be hot in testing as well. The sparse features and dense features

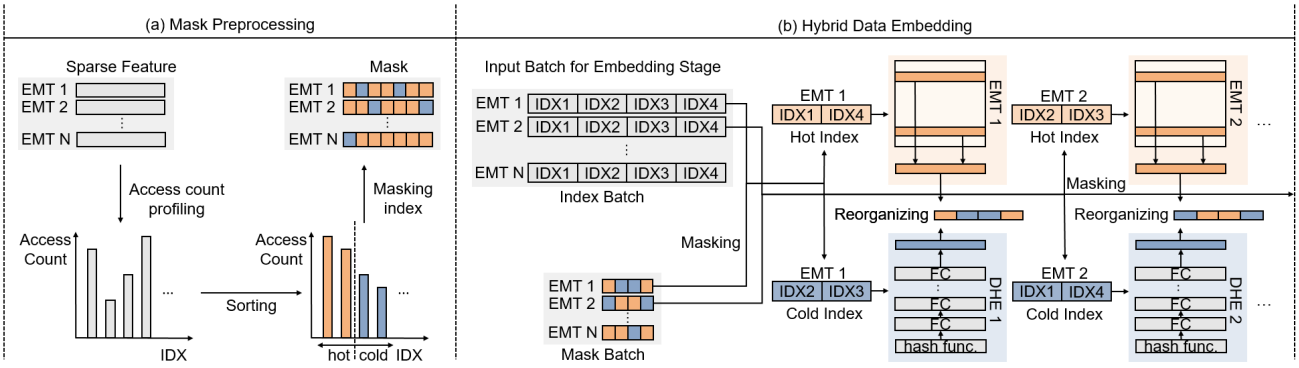


Fig. 3: Overall process of the hybrid data embedding.

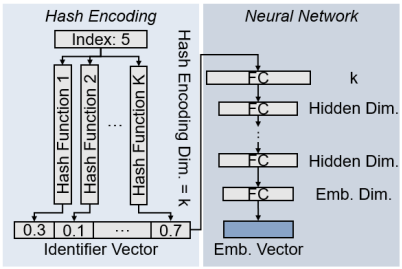


Fig. 4: DHE network computation. The number of the hash functions for hash encoding is equal to the hash encoding dimension k.

are processed through different paths. Dense features, like in conventional systems, are processed through the bottom MLP layer on the GPU. On the other hand, sparse features are divided into hot and cold features on the CPU via the mask. Hot features are processed through embedding table lookups on the CPU, while cold features are computed using the DHE network on the GPU. These two embedding vectors are concatenated, and their results, along with the bottom MLP layer's output, are passed through feature interaction before going through the top MLP layer to produce the final Click-Through Rate (CTR). Section III.C shows how embedding table and network aggregation techniques are employed to optimize the parallel processing of embedding lookups and DHE operations.

B. Hybrid Data Embedding: Embedding Tables and DHE for Efficient Data Handling

The overall process of the proposed HDE is depicted in Figure 3. Let's assume an example where the batch size is 4 and there are N embedding tables. Before applying HDE, a preprocessing step for the mask is required. In the mask preprocessing stage shown in Figure 3 (a), the access count for each embedding table is first analyzed based on the sparse features. The indices are then sorted according to these access counts, and the top 5% of indices are classified as hot, while the remaining indices are classified as cold, resulting in the creation of the mask. Figure 3 (b) shows the HDE stage. The input batch for HDE consists of an index batch and a pre-generated mask batch. First, the indices for the first embedding table in the index batch are divided into hot and cold indices using the mask batch. For example, the indices IDX1, IDX2, IDX3, and IDX4 for the first embedding table are classified into hot (IDX1 and IDX4) and cold (IDX2 and IDX3) based on the mask batch. Next, the hot indices are used to lookup the embedding table to retrieve the embedding vectors, while the cold indices are processed through the first DHE network to compute the embedding vectors. Finally, the output embedding vectors for each type of index (hot and cold) are rearranged in the original batch order, and the same procedure is repeated for the next embedding table.

Figure 4 illustrates the process of calculating cold embedding vectors using the proposed DHE network, which follows a structure similar to a typical DHE network [20]. The DHE network consists of two main components: hash encoding and a neural network. First, the cold index is input into K hash functions, which generate a K-dimensional identifier vector. Here, K represents the hash encoding dimension. The generated identifier vector is then used as the input to a neural network consisting of fully connected (FC) layers to compute the embedding vector. Compared to a typical embedding table lookup, the identifier vector corresponds to the row index of the embedding table, and the neural network takes over the role of the embedding table. The DHE network can progressively approximate the ground-truth embedding vector through training. In contrast to traditional embedding table lookup, which requires storing a row for each item in the embedding table and thus demands large memory, the DHE network implicitly represents data through the neural network, significantly reducing memory requirements.

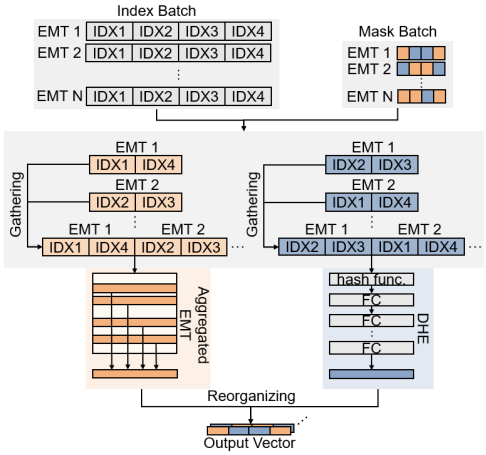
Since embedding vector computation and embedding table lookup are independent tasks, the table lookup for hot embedding vectors can be processed on the CPU, while the computation of cold embedding vectors through the DHE network can be handled on the GPU in parallel. Through this hybrid data embedding approach, cold embedding vectors, which account for about 90% or over of the traditional embedding table, can be replaced with much smaller DHE network parameters. This results in generating embedding vectors with only about 5% of the memory required by the traditional

embedding table, significantly reducing memory usage. Unlike the traditional DHE network, the proposed HDE approach does not replace the entire embedding table. Instead, only the cold embedding vectors are replaced with the DHE network, thus reducing memory usage more aggressively while minimizing performance degradation. This is because cold embedding vectors occupy only a very small portion of the total access count, and their impact on the overall model performance is minimal [13].

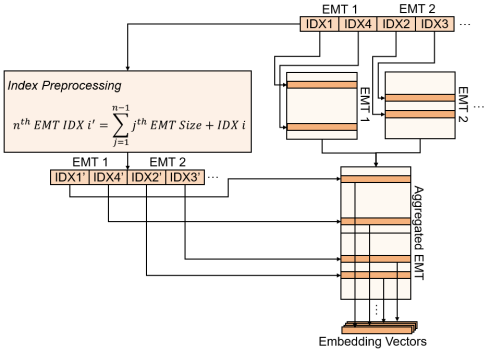
C. HDE Optimization Through Table and Network Aggregation

Through HDE, embedding lookup and computation can be performed in parallel, but several issues can arise with a naive implementation shown in Figure 3 (b). First, significant time overhead occurs in the process of distinguishing hot and cold indices and reorganizing the results. If there are N embedding tables, the task of distinguishing hot and cold indices using the mask and the task of reorganizing the embedding table lookup results for hot indices and the embedding vector computation results for cold indices will be repeated N times. Second, the DHE network replaces the embedding table, meaning that the number of DHE networks needed is equal to the number of embedding tables. As the number of embedding tables increases, the number of parameters for the DHE network also increases, leading to higher computational costs due to repeated embedding vector computations. Even though the embedding vector computation is performed in parallel with the embedding table lookup, if the computation time is too long, the lookup time may not be able to hide it. Lastly, the embedding table lookup for hot indices and the embedding vector computation for cold indices can cause bottlenecks. For example, if the lookup for hot indices in the first embedding table completes quickly, the process must wait for the cold index vector computation to finish, which could delay the lookup for the second embedding table.

Figure 5 shows the improvement of parallel processing for embedding lookup and computation from Figure 3 (b). Before proceeding with embedding lookup and computation, the indices for all embedding tables in the input batch are masked. As shown in Figure 4 (a), the first embedding table's mask identifies IDX1 and IDX4 as hot indices for EMT 1, and the second embedding table's mask identifies IDX2 and IDX3 as hot indices for EMT 2. Cold indices are masked in the same way. The hot and cold indices, once classified, are gathered separately. Hot indices are looked up in the aggregated embedding table all at once, and cold indices are computed all at once using a single DHE network. Figure 5 (b) shows the embedding table lookup using the aggregated embedding table in more detail. The aggregated embedding table is created by concatenating multiple embedding tables into one large table. As a result, preprocessing of the hot indices for the aggregated embedding table is necessary. The new index for a given IDX i in the original n-th embedding table can be calculated by adding the row sizes of the first embedding table through the (n-1)-th embedding table, and then adding IDX i. Depending



(a) Aggregated embedding table and DHE network



(b) Aggregated embedding table lookup

Fig. 5: Aggregated hybrid data embedding

on the dataset, the number of items in each embedding table can vary greatly. For smaller embedding tables, there may be only 3-4 items, while larger embedding tables can have more than 10 million items. Smaller embedding tables can be entirely stored in memory without significantly changing the overall memory requirements. Since the DHE network is intended to reduce the data size by investing in processing time, replacing already small embedding tables with the DHE network would be inefficient. Therefore, during preprocessing of the mask, only embedding tables with a large number of items are processed with the hot/cold mask for HDE, reducing unnecessary computation. The method described above solves the hot/cold overhead and computational cost issues that arise in a naive approach by performing the masking and embedding vector computation only once. Additionally, by masking and gathering all cold and hot indices in the batch before computation, it resolves the bottleneck problem that existed between embedding table lookup and embedding vector computation.

IV. Experiments

This section describes the experimental setup, including the datasets used, evaluation metrics, and implementation details. Subsequently, the effectiveness of the proposed HDE method is evaluated through comparisons with existing embedding reduction techniques.

TABLE I: Overall Results. The best and second-best results are **bolded** and underlined

Embedding Methods	Criteo					Avazu				
	Memory Reduction	AUC	Accuracy(%)	Logloss	Exe. Time(s)	Memory Reduction	AUC	Accuracy(%)	Logloss	Exe. Time(s)
FDE	1	**0.8022**	78.609	**0.4477**	323.74	1	0.7493	84.551	**0.3927**	**141.92**
DHE	0.268	0.7662	77.156	0.4765	4305.18	0.257	0.7280	84.367	0.4048	2927.91
CE	0.250	0.7931	78.359	0.4556	632.31	0.250	0.7460	84.531	0.3968	230.72
HDE	**0.051**	0.8009	**78.612**	0.4491	**316.50**	**0.050**	**0.7516**	**84.577**	0.3934	146.25

A. Experimental Setup

Datasets We conducted experiments using two widely used CTR prediction datasets, Criteo [25] and Avazu [25]. The Criteo dataset consists of advertisement click records from 45 million users over a 7-day period and includes 13 dense features and 26 categorical features. The data from the first 6 days was used as the training set, and the data from the last 7th day was split into validation and test sets in a 1:1 ratio. The Avazu dataset consists of mobile ad click records from an 11-day period and includes 8 dense features and 13 categorical features. 80% of the user-item interactions were used as the training set, with the remaining 20% split equally into validation and test sets.

Comparisons The conventional techniques used for comparison include full-dimension embedding (FDE) [3], compositional embedding (CE) [19], and DHE [20]. FDE is the baseline method used in the original DLRM, where the same embedding dimension is applied across all feature fields. CE utilizes the quotient-remainder trick to replace the embedding table with smaller embedding tables. In the experiments, the QR collision value was set to 4. DHE combines hash encoding and a neural network to replace the embedding table, generating embedding vectors through computation rather than lookup. For the experiments, we set the size of the DHE network to approximately one-quarter of the size of the FDE.

Evaluation metrics For performance comparison, memory reduction, execution time, area under the receiver operating characteristic curve (AUC), and Logloss were used as evaluation metrics. Memory reduction is a metric representing the size of the experimental method relative to the baseline FDE's embedding table size, normalized based on the FDE's embedding table size. Execution time measures the effective latency during actual parallel execution of the task. For the baseline, since embedding lookup and the bottom MLP are performed simultaneously, the execution time is measured as the time until both tasks are completed. AUC represents the probability that a positive instance is ranked higher than a randomly selected negative instance, with a higher value indicating better performance. Logloss is the commonly used binary cross-entropy loss, with a lower value indicating better performance. In DLRM, the embedding table size constitutes the majority of the model parameters [5], and differences of 0.001 in AUC or Logloss are considered significant in CTR prediction tasks [26], [27].

Implement Details In the proposed HDE, hot/cold masking is performed by classifying the top 5% of vectors based on access count as hot, and the remaining vectors as cold. The hash bucket size for HDE is set to 10^6, and the hash encoding dimension, hidden layer dimension, and FC layer count are set to 1024, 256, and 9 for the Criteo dataset, respectively, and 128, 128, and 5 for the Avazu dataset. The embedding dimension for the DLRM model used in the experiments is 16, and the top MLP layer's dimension is configured as "512-64-1". The bottom MLP layer is designed considering the dataset size and the number of dense features, with "13-512-256-64-16" for the Criteo dataset and "8-256-64-16" for the Avazu dataset. DLRM training is conducted for 1 epoch, with a batch size set to 128. The learning rate is 0.1 for FDE, DHE, and HDE using stochastic gradient descent (SGD), and 0.01 for CE using adaptive gradient (AdaGrad). The experiments are performed on a single AMD EPYC 7742 64-core processor and a single Nvidia Tesla A100 GPU, and the execution time for each task is profiled using Python's line profiler.

B. Overall Results

In the CPU-GPU hybrid environment, for the baseline FDE, the embedding lookup is performed on the CPU while the bottom MLP is executed on the GPU. For DHE and CE, the embedding lookup is performed on the GPU, while the bottom MLP runs on the CPU. The results for each embedding method are summarized based on the fastest execution time. Table I shows the overall results.

Comparison with baseline When comparing FDE with the proposed HDE, we can see that HDE significantly reduces memory usage, utilizing only 5.1% and 5.0% of FDE's parameters for the Criteo dataset and Avazu dataset, respectively. Specifically, for the Avazu dataset, memory usage is reduced to 5.0% of FDE's memory, while AUC and accuracy improve by about 0.0023 and 0.026%, respectively, and training time remains similar to the baseline.

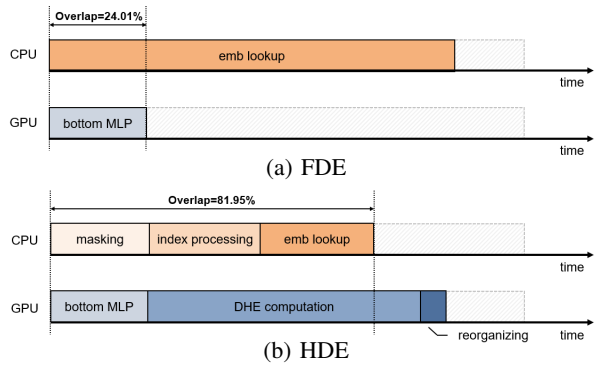
Comparison with other methods When comparing HDE with other embedding methods, it outperforms in all metrics such as memory reduction, accuracy, and execution time. Compared to DHE, for the Criteo dataset, HDE uses 5.25 times fewer parameters, has 13.60 times faster training time, and shows an accuracy improvement of approximately 1.46%. Compared to CE, for the Criteo dataset, HDE uses about 5 times fewer parameters, has approximately twice as fast training time, and achieves 0.253% higher accuracy.

C. Latency Breakdown

Table II shows the latency breakdown between the baseline FDE and the proposed HDE. While there is data transfer

TABLE II: Latency breakdown on Criteo Kaggle dataset.

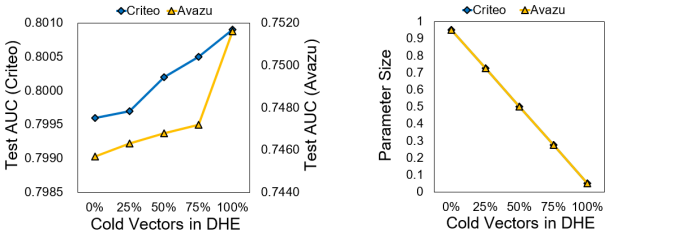
Embedding Methods	CPU time (s)			GPU time (s)			Overall time (s)
	Masking	Index Preprocessing	Embedding Lookup	Bot MLP	DHE Computation	Reorganizing	(CPU/GPU)
FDE	-	-	323.74	77.73	-	-	323.74/77.73
HDE	79.14	89.28	90.96	77.73	218.39	20.38	259.38/316.50



(a) FDE

(b) HDE

Fig. 6: Timing diagrams of the (a) full dimension embedding(FDE) and (b) the proposed hybrid data embedding(HDE). In the diagram, the length of each task is proportional to the actual latency.



(a) Test AUC

(b) Number of Parameters

Fig. 7: Test AUC and number of parameters for varying portion of cold vectors in DHE.

from the CPU to the GPU in HDE, the data transfer latency is negligible when considering the GPU bandwidth and the size of the transferred data. As a result, it was not reported separately. According to the experimental results, FDE takes 323.74s for embedding lookup on the CPU and 77.73s for the bottom MLP on the GPU, with about 24.01% overlap between the two. In contrast, HDE takes 259.38s for embedding lookup on the CPU and 316.50s for the bottom MLP on the GPU, with about 81.95% overlap. The high overlap percentage indicates that resources are being utilized more efficiently during execution [28].

Figure 6 shows the timing diagram when embedding lookup and bottom MLP are performed in parallel in the CPU-GPU hybrid environment. In Figure 6(a), for FDE, it is observed that there is significant idle time where the GPU waits for the CPU's embedding lookup to finish after completing the bottom MLP. On the other hand, in Figure 6(b) for HDE, we see that the work between the CPU and GPU is effectively distributed. While the GPU performs the bottom MLP, the CPU handles masking and index preprocessing, and while the CPU performs the embedding table lookup, the GPU simultaneously performs the DHE computation, increasing the overlap between CPU and GPU operations and thereby efficiently utilizing resources.

D. Analysis of Hybrid Data Embedding

One of the key contributions of HDE is the division of embedding vectors into hot and cold categories based on their access frequency, with only the cold embedding vectors being computed through the DHE network. Figure 7 shows the relationship between the proportion of cold vectors computed by the DHE network, the test AUC, and the number of parameters. In Figure 7(a), as the proportion of cold vectors computed by the DHE network increases from 0% to 100%, we observe a gradual increase in test AUC. Additionally, Figure 7(b) shows that as the proportion of cold vectors increases, the total number of parameters decreases. This indicates that processing only the cold embedding vectors through the DHE network based on data *hotness* is an efficient method that can maintain high performance with fewer parameters.

V. Conclusion

This study proposed the HDE framework to address the memory bottleneck in DLRMs by storing frequently accessed hot data in embedding tables and computing less frequently accessed cold data using a DHE network. This approach reduced memory usage by approximately 20 times compared to baseline methods while maintaining accuracy and training time, with efficient resource utilization achieved through parallel execution in a CPU-GPU hybrid environment. Future work includes incorporating dynamic hot/cold classification, optimizing DHE network architectures, scaling to multi-GPU systems, and generalizing the framework to other applications, enhancing its scalability and versatility for large-scale recommendation systems.

VI. Acknowledgment

This work was partly supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2020-II201373, Artificial Intelligence Graduate School Program at Hanyang University), the Technology Innovation Program (or Industrial Strategic Technology Development Program) (RS-2023-00235718, 23040-15FC) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) (1415187505), and the Information Technology Research Center (ITRC) support program (IITP-2025-2020-0-01461) supervised by IITP and funded by the MSIT.

REFERENCES

[1] Hang Xue and Dongmei Zhang. A recommendation model based on content and social network. In *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pages 477–481, 2019.

[2] Doae Mensouri and Abdellah Azmani. A new marketing recommendation system using a hybrid approach to generate smart offers. *Applied Computer Systems*, 27(2):149–158, 2022.

[3] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. Deep learning recommendation model for personalization and recommendation systems, 2019.

[4] Bilge Acun, Matthew Murphy, Xiaodong Wang, Jade Nie, Carole-Jean Wu, and Kim Hazelwood. Understanding training efficiency of deep learning recommendation models at scale. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 802–814, 2021.

[5] Ehsan K. Ardestani, Changkyu Kim, Seung Jae Lee, Luoshang Pan, Jens Axboe, Valmiki Rampersad, Banit Agrawal, Fuxun Yu, Ansha Yu, Trung Le, Hector Yuen, Dheevatsa Mudigere, Shishir Juluri, Akshat Nanda, Manoj Wodekar, Krishnakumar Nair, Maxim Naumov, Chris Petersen, Mikhail Smelyanskiy, and Vijay Rao. Supporting massive dlrm inference through software defined memory. In *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, pages 302–312, 2022.

[6] Michael Lui, Yavuz Yetim, Özgür Özkan, Zhuoran Zhao, Shin-Yeh Tsai, Carole-Jean Wu, and Mark Hempstead. Understanding capacity-driven scale-out neural recommendation inference. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 162–171, 2021.

[7] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. Software-hardware co-design for fast and scalable training of deep learning recommendation models. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, pages 993–1011, 2022.

[8] Aditya Desai and Anshumali Shrivastava. The trade-offs of model size in large recommendation models: 100gb to 10mb criteo-tb dlrm model. *Advances in Neural Information Processing Systems*, 35:33961–33972, 2022.

[9] Daochen Zha, Louis Feng, Qiaoyu Tan, Zirui Liu, Kwei-Herng Lai, Bhargav Bhushanam, Yuandong Tian, Arun Kejariwal, and Xia Hu. Dreamshard: generalizable embedding table placement for recommender systems. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2024. Curran Associates Inc.

[10] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. Heterogeneous acceleration pipeline for recommendation system training. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 1063–1079, 2024.

[11] Saurabh Agarwal, Chengpo Yan, Ziyi Zhang, and Shivaram Venkataraman. Bagpipe: Accelerating deep recommendation model training. In *Proceedings of the 29th Symposium on Operating Systems Principles*, SOSP '23, page 348–363, New York, NY, USA, 2023. Association for Computing Machinery.

[12] Muhammad Adnan, Yassaman Ebrahimzadeh Maboud, Divya Mahajan, and Prashant J. Nair. Accelerating recommendation system training by leveraging popular choices. *Proceedings of the VLDB Endowment*, 15(1):127–140, September 2021.

[13] A.A. Ginart, Maxim Naumov, Dheevatsa Mudigere, Jiyan Yang, and James Zou. Mixed dimension embeddings with application to memory-efficient recommendation systems. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pages 2786–2791, July 2021.

[14] Jian Zhang, Jiyan Yang, and Hector Yuen. Training with low-precision embedding tables. In *Systems for Machine Learning Workshop at NeurIPS*, volume 2018, 2018.

[15] Hui Guan, Andrey Malevich, Jiyan Yang, Jongsoo Park, and Hector Yuen. Post-training 4-bit quantization on embedding tables. *arXiv preprint arXiv:1911.02079*, 2019.

[16] Zirui Liu, Qingquan Song, Li Li, Soo-Hyun Choi, Rui Chen, and Xia Hu. Pme: pruning-based multi-size embedding for recommender systems. *Frontiers in big Data*, 6:1195742, 2023.

[17] Wei Deng, Junwei Pan, Tian Zhou, Deguang Kong, Aaron Flores, and Guang Lin. Deeplight: Deep lightweight feature interactions for accelerating ctr predictions in ad serving. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 922–930, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 1113–1120, New York, NY, USA, 2009. Association for Computing Machinery.

[19] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 165–175, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Wang-Cheng Kang, Derek Zhiyuan Cheng, Tiansheng Yao, Xinyang Yi, Ting Chen, Lichan Hong, and Ed H. Chi. Learning to embed categorical features without embedding tables for recommendation. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 840–850, New York, NY, USA, 2021. Association for Computing Machinery.

[21] F. Maxwell Harper and Joseph A. Konstan. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015.

[22] Cheng-Yu Chen, Jui-Nan Yen, You-Ru Lai, Yun-Ping Lin, and Chia-Lin Yang. Rects: A temporal-aware memory system optimization for training deep learning recommendation models. In *Proceedings of the 17th ACM International Systems and Storage Conference*, SYSTOR '24, page 104–117, New York, NY, USA, 2024. Association for Computing Machinery.

[23] Hiwot Tadese Kassa, Paul Johnson, Jason Akers, Mrinmoy Ghosh, Andrew Tulloch, Dheevatsa Mudigere, Jongsoo Park, Xing Liu, Ronald Dreslinski, and Ehsan K Ardestani. Mtrains: Improving dlrm training efficiency using heterogeneous memories. *arXiv preprint arXiv:2305.01515*, 2023.

[24] Keshav Balasubramanian, Abdulla Alshabanah, Joshua D Choe, and Murali Annavaram. cdlrm: Look ahead caching for scalable training of recommendation models. In *Proceedings of the 15th ACM Conference on Recommender Systems*, RecSys '21, page 263–272, New York, NY, USA, 2021. Association for Computing Machinery.

[25] Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. Open benchmarking for click-through rate prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 2759–2769, New York, NY, USA, 2021. Association for Computing Machinery.

[26] Gopi Krishna Jha, Anthony Thomas, Nilesh Jain, Sameh Gobriel, Tajana Rosing, and Ravi Iyer. Mem-Rec: Memory efficient recommendation system using alternative representation. In Berrin Yanıkoğlu and Wray Buntine, editors, *Proceedings of the 15th Asian Conference on Machine Learning*, volume 222 of *Proceedings of Machine Learning Research*, pages 518–533. PMLR, 11–14 Nov 2024.

[27] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

[28] Joel Hestness, Stephen W. Keckler, and David A. Wood. Gpu computing pipeline inefficiencies and optimization opportunities in heterogeneous cpu-gpu processors. In *2015 IEEE International Symposium on Workload Characterization*, pages 87–97, 2015.

Mitigating Routability Problems in Complementary-FET-based VLSI Designs

Junghyun Yoon* and Heechun Park†
Department of Electrical Engineering, Ulsan National Institute of Science and Technology, Ulsan, Korea
{*yoon_jh, †h.park}@unist.ac.kr

***Abstract*—As semiconductor technology scales beyond 5nm, complementary FET (CFET) that stacks P-FET and N-FET enables extreme cell area scaling. However, reduced standard cell height with CFET presents routability challenges due to limited back-end-of-line (BEOL) routing resources. In this paper, we address two key routability problems, i.e., pin accessibility and routing congestion, by employing various pin-extended cells on demand. Moreover, we introduce an end-to-end VLSI design framework that further alleviates routing congestion using partial placement blockages. Experimental results demonstrate that our framework eliminates most design rule violations (DRVs) related to routability while maintaining the area advantages of CFET technology.**

I. Introduction

As semiconductor process technology advances beyond 5nm, conventional transistor structures encounter scaling limitations. For instance, when FinFET technology [1] has attempted to reduce cell height through fin depopulation [2], the 3nm process confines transistors to a single fin, making further scaling challenging. Nanosheet FET (NSFET) [3] was then proposed to overcome this limitation by vertically stacking thin sheet-shaped channels, enabling an increased channel count. While nanosheets eliminated the need for lateral spacing between channels, the spacing requirement between NMOS and PMOS regions remained. Forksheet FET (FSFET) [4] was introduced to minimize the gap between NMOS and PMOS regions, which further downscales the cell height.

Beyond the aforementioned approaches, the complementary FET (CFET) [5] structure introduces an innovative design by vertically stacking PMOS on top of NMOS, or vice versa, that significantly reduces the cell footprint. By eliminating the need for lateral NMOS-PMOS spacing, CFET achieves a substantial reduction in standard cell height from the conventional 5-6 tracks in the 5nm process to 2-4 tracks [6], resulting in over 40% improvement in the cell area. However, the CFET structure is facing new challenges in physical design, mainly due to the routing resource limitation caused by significantly reduced cell height.

The routing resource limitation incurs two significant problems: *pin accessibility* and *routing congestion*. Firstly, pin accessibility refers to the difficulty of accessing a standard cell's input/output pins. While conventional cells with NMOS and PMOS on the same plane (Fig. 1(a)) provide sufficient cell height to extend pins vertically, CFET cells (Fig. 1(b)) have limited height for pin extension, and thus the number of pin access points is severely restricted. Secondly, the dramatic size reduction of CFET cells results in significantly higher cell density within the same area, leading to increased routing demand followed by severe routing congestion issues.

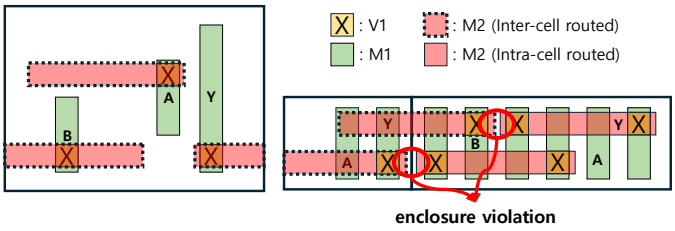


Fig. 1: An illustration of the usage of (a) a conventional FET cell and (b) a complementary FET (CFET) cell.

In this paper, We propose a novel methodology to solve routability issues while maintaining single-row structures for simple logic gates. Our approach is based on two key ideas: First, we introduce a systematic approach to improving pin accessibility by strategically *extending pins to the M2 layer*. While this might initially appear to increase metal resource usage, our optimized pin shape design minimizes M2 track utilization while maximizing accessibility. Furthermore, we leverage available white space during placement to selectively *extend pins in the M1 layer*, creating additional access points without compromising routing resources. Second, we propose a *congestion-aware local density allocation* to address routing congestion by analyzing congestion from early global routing (eGR) reports and strategically allocating local density.

Our methodology makes the following three key contributions:

1) We introduce a comprehensive methodology for **improving pin accessibility** in CFET cells through pin extension techniques, demonstrated through detailed analysis and practical implementation strategies.
2) We propose a proactive approach to **resolve routing congestion problems** with congestion-aware local density allocation to demonstrate that effective routing is possible even with extreme scaling.
3) We develop an **end-to-end design framework** for implementing the proposed methodology that addresses routability issues while preserving the advantages of the CFET technology node.

The remainder of this paper is organized as follows. Section II explains the routability problems in CFET cells and related works that address these issues. Section III describes our

design framework implementation, and Section IV presents experimental results. Finally, Section V concludes the paper.

II. Background and Related Works

A. Routability Problems in CFET-based Designs

Due to the vertical stacking of NMOS and PMOS, CFET reduces cell height by approximately half compared to conventional cells, and some complex cells like full adder, flip-flop, and AOI/OAI require more routing tracks for intra-cell routing resources. Since the CFET structure allows easier connections between the same layers in the same vertical line within standard cells when signals are identical [7], there is an increasing trend toward implementing multi-row CFET-based standard cells to improve intra-cell routability [7], [8].

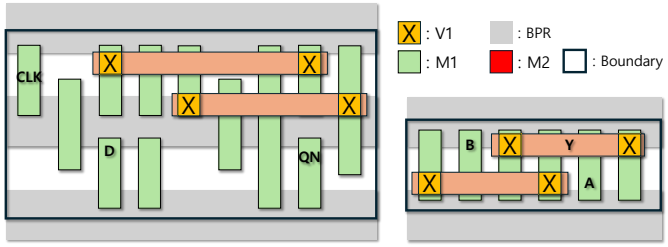


Fig. 2: CFET_fp [9] multi-row 2T library layout. (a) DFFHQNx1 (b) NAND2x1

Nevertheless, the reduced cell size still faces routing resource shortages, resulting in less accessible space for I/O pins within cells. As shown in Fig. 2, while the complex cells are made as double-row cells (Fig. 2(a)), simple cells like NAND2, NOR2, INV can only be designed as a single row (Fig. 2(b)). Here, with only two accessible M2 tracks per cell, and if one M2 track is already used for intra-cell routing, pin access through M1 becomes even more challenging.

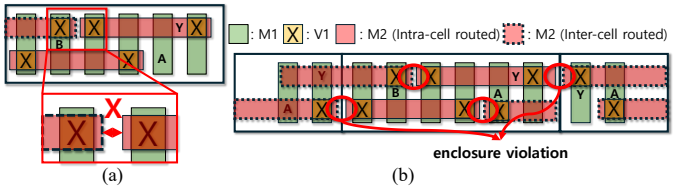


Fig. 3: (a), (b): A tip-to-tip violation occurred due to the via enclosure during inter-cell routing.

In single-row cells, as shown in Fig. 3(a), due to intra-cell routing, both input pins A and B have only one pin access point for each. When these pins undergo inter-cell routing at the chip level, tip-to-tip violation may happen between the via enclosure for the chip-level M2 segment and the existing intra-cell M2 segment. When two such cells are adjacent, the tip-to-tip violation may also happen between the two cells as shown in Fig. 3(b).

Moreover, CFET-based designs are also facing routing congestion problems. Fig. 4 compares the design layouts between using a conventional ASAP7 PDK [10] and using a CFET PDK with negligible pin accessibility issues (CFET_fp single-row 4T [9]). These layouts demonstrate that, aside from pin accessibility issues, the reduced area by CFET incurs

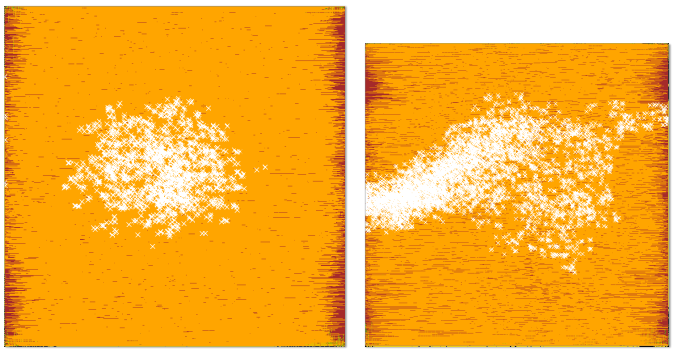


Fig. 4: The layout of LDPC benchmark using different PDK. (a) ASAP7 (100.76 um×100.44 um, 3479 DRVs), (b) CFET_fp [9] (89.57 um×89.46 um, 8012 DRVs). White X marks represent DRVs.

design rule violations (DRVs) that remain unresolved by the conventional P&R tool, which are highly related to the limited routing resources.

B. Related Works

Recent research has made significant progress in implementing CFET technology, particularly in standard cell design and chip-level evaluation. [7] achieved a comprehensive implementation spanning from device simulation to chip-level assessment for CFET, demonstrating its feasibility in real-world applications.

Multiple research efforts have successfully demonstrated automated CFET standard cell synthesis [9], [11], showcasing the possibility of generating manufacturable CFET-based standard cell libraries. These studies have established fundamental CFET standard cell design methodologies and achieved substantial area reduction compared to conventional architectures. [11] made notable advances in CFET design by improving pin accessibility through the SMT(satisfiability modulo theories)-based solution. Then, [9] provided an automated CFET multi-row standard cell design flow with an insight into routing congestion at the chip level.

While the previous studies identified and characterized routing congestion issues, they did not present specific solutions to mitigate these challenges. In this work, we focus on addressing the critical routability issues of pin accessibility and routing congestion in CFET-based designs, and propose novel solutions to enhance pin accessibility while simultaneously mitigating routing congestion, which are not covered by the previous works.

III. Routability-Aware CFET-based VLSI Design Framework

A. Overall Design Flow

Our novel design flow, illustrated in Fig. 5, addresses the critical challenges of pin accessibility and routing congestion in CFET-based designs. The flow begins with library preparation, where we create various *pin-extended cells* on both M1 (whitespace-pin cells) and M2 (forced-M2-pin cells) layers for the single-row cells in the original standard cell

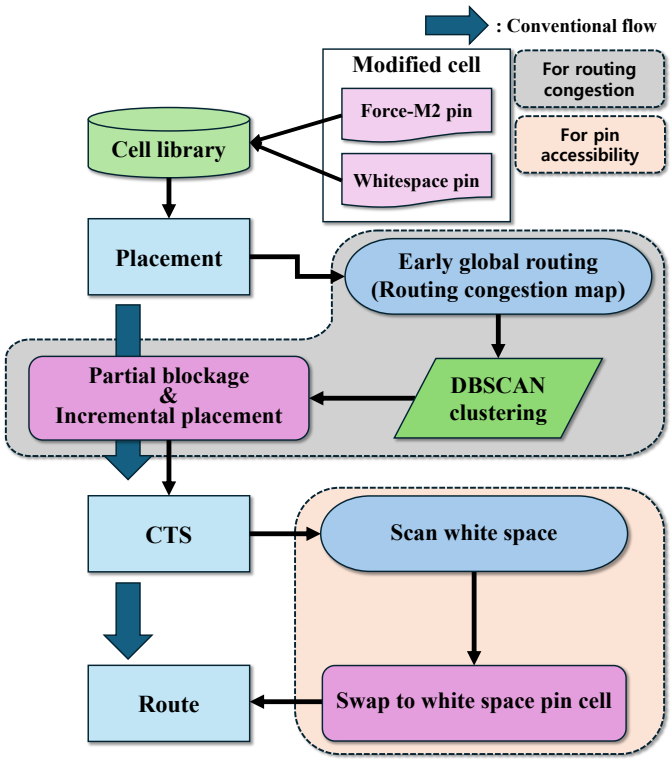


Fig. 5: Overall design flow of our routability-aware design framework.

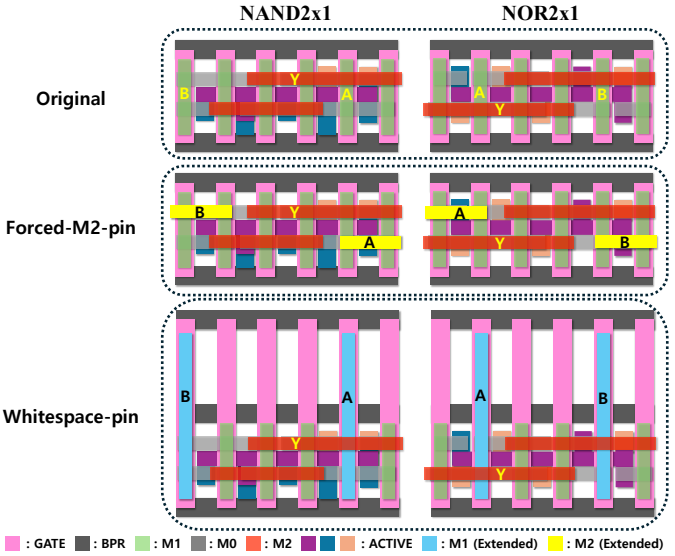


Fig. 6: Our pin-extended cell layouts.

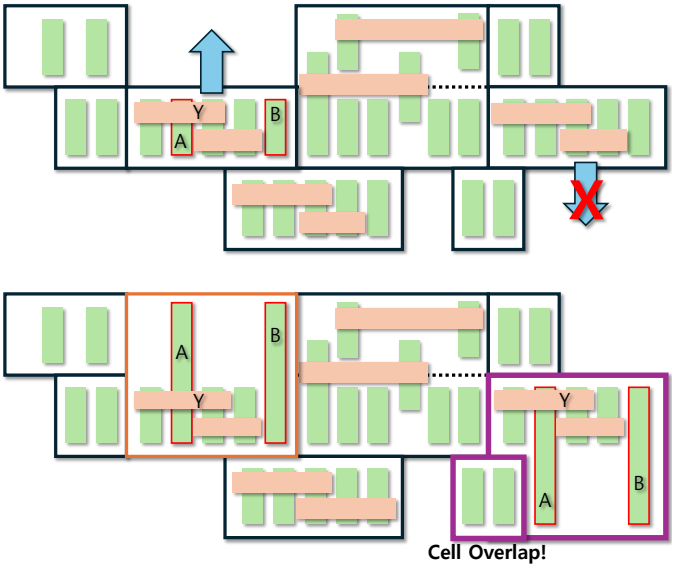


Fig. 7: If a single-row cell has enough whitespace in the row above or below it, the cell can be swapped with a whitespace-pin cell.

library. Fig. 6 lists the representative single-row cells in the original CFET standard cell library, and their corresponding whitespace-pin cells and forced-M2-pin cells. A whitespace-pin cell is made by extending the M1 pins and the cell's placement boundary vertically to the upper (or lower) row, and a forced-M2-pin cell is created by extending the pin to an unoccupied M2 metal track within the cell boundary. Since we do not modify any intra-cell routing when generating these pin-extended cells, they resolve pin accessibility challenges effectively with negligible impact on chip-level design quality.

In the beginning, we conduct a conventional placement stage after substituting all single-row cells to the forced-M2-pin cells that have no area overhead compared to the original cell. Then, we generate the routing congestion map from the eGR result, and insert partial placement blockages followed by an incremental placement step to proactively address the congested region problems before routing. After another conventional clock tree synthesis (CTS) stage, we examine the placement of single-row cells. If there is a whitespace above or below a single-row cell, the cell type is swapped with a whitespace-pin cell that incurs no chip area penalty, improves pin accessibility, and reduces potential routing congestion over the M2 layer at the same time. The conventional routing stage then finalizes the full-chip design, with routability issues being resolved.

B. Pin Accessibility Enhancement

In this subsection, we provide a detailed explanation of the methods we propose to address the pin accessibility issue, which corresponds to the bottom right part of Fig. 5. By the end of the CTS stage, all single-row cells are in the state of being replaced with forced-M2-pin cells. Here, we use the scenario as illustrated in Fig. 7 to describe our strategy of replacing forced-M2-pin cells with whitespace-pin cells.

Starting from the placement status corresponding to the upper figure of Fig. 7, we examine all single-row cells to identify those with sufficient whitespace in the row above or below to allow vertical extension. If we find such a cell (i.e., the middle left cell in Fig. 7), we replace its cell type with the pre-defined whitespace-pin cell, resulting in the lower figure of Fig. 7. Such a replacement remains the core area, allows more pin access points, and does not use the M2 layer that achieves less routing congestion compared to the forced-M2-pin cell counterpart.

Note that another candidate cell, the rightmost single-row

cell in Fig. 7, is not allowed for vertical extensions since its extension to any direction (i.e., upper or lower) incurs cell overlaps and the following placement refinement may harm the entire design quality. Single-row cells that cannot be vertically extended remain in the form of forced-M2-pin cells, preserving the benefits of improved pin accessibility. The routing congestion issue on the M2 layer is handled by subsequent solutions.

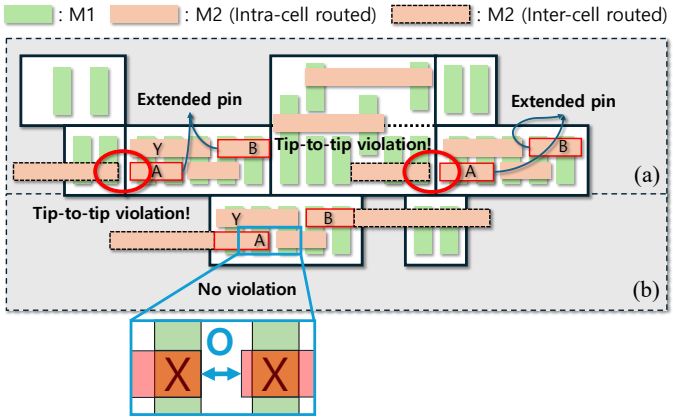


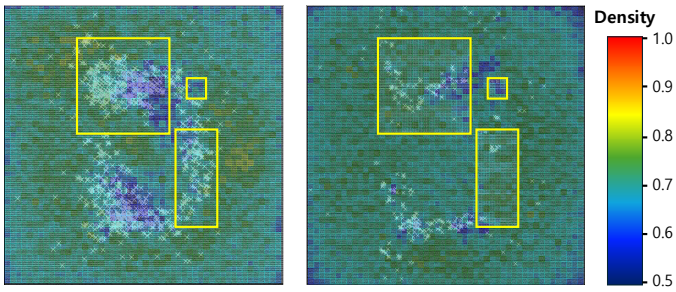
Fig. 8: (a) Tip-to-top violations due to the via enclosures between extended M2 pins and chip-level routing segments. (b) No enclosure violation by the asymmetric via enclosure in the forced-M2-pin cell.

Apart from the algorithmic approach, when we utilize forced-M2-pin cells for higher pin accessibility, those extended pins have a risk of incurring tip-to-tip violations with the nearby intra-cell or inter-cell routing segment, as illustrated in Fig. 8(a). This mainly occurs due to the *via enclosure* for the intra-cell V1, which is used to connect the original M1 pin and the extended M2 pin, being added to M2 and not properly accounted for during the chip-level routing stage. This problem can be solved by generating an asymmetric via enclosure in the M2 layer to reserve more spacing with the potential M2 routing segment, as in Fig. 8(b).

C. Alleviating Routing Congestion

This subsection describes our technique to address routing congestion problems, corresponding to the middle part of Fig. 5. After the placement stage, we can obtain the congestion report from the placement density and the eGR result. Then, we employ the conventional DBSCAN (density-based spatial clustering of applications with noise) [12] algorithm to find clusters for the identified congestion hotspots, where the congestion levels are up to two steps below the peak value. We use the empirically obtained parameter values ($eps = 4$, $minPts = 2$) for the DBSCAN algorithm.

The partial placement blockages are then sized and placed based on the bounding boxes of identified congestion clusters. We set the utilization target within these blockages as same as global design utilization, which prevents excessive gathering of cell instances while maintaining overall design efficiency. After placing the partial placement blockages, we utilize the commercial tool's placement engine to conduct *incremental placement* that preserves the quality of the current placement



(a) Before (2126 DRVs) (b) After (**619** DRVs)

Fig. 9: Effect of applying partial placement blockages and incremental placement in LDPC benchmark.

and simultaneously refines cell positions according to our partial blockages.

Fig. 9 shows the reduction of DRVs using our method. We significantly reduce DRVs, especially in areas where we apply partial blockages. This improvement stems from our strategic density management approach.

IV. Experimental Results

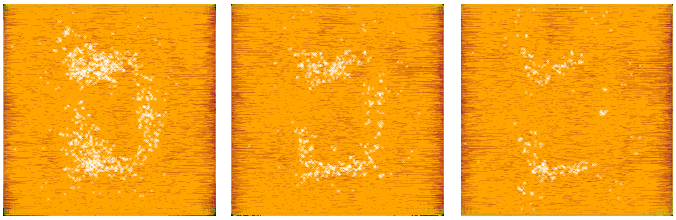
A. Experimental Setup

Our CFET-based routability optimization framework leverages commercial EDA tools for the main flow, i.e., Synopsys Design Compiler for logic synthesis and Cadence Innovus for P&R. We use the open-source CFET_fp [13] standard cell library as our reference CFET standard cells, where we utilize a multi-row 2T standard cell library that contains 11 complex cells in double-row format and 17 simple cells (e.g., NAND, NOR, INV, etc.) implemented in a single row. Since the library in [13] only contains the standard cell layouts in GDS format, we incorporate a scaled version of open-source ASAP 7nm [10] as BEOL rules, and a predictive 3nm timing/power library from [14] to construct the full technology files for our experiment. Such a mixture of technology files represents a reasonable approximation of future technology nodes, allowing us to evaluate the effectiveness of our approach for upcoming challenges in advanced node designs, and ensure that our routability improvements maintain timing closure in realistic scenarios. Our benchmark circuits are obtained from Opencores [15].

B. Routability Analysis

We first validate the efficiency of our routing congestion mitigation method with partial placement blockage and incremental placement. Fig. 10 shows different post-route results of LDPC benchmark according to various design flows: Fig. 10(a) is the result from the conventional P&R flow; Fig. 10(b) comes from the design flow that an incremental placement step is conducted after the common placement stage *without inserting any partial placement blockages*; Fig. 10(c) is the result of our proposed framework, which inserts partial placement blockages followed by an incremental placement step. In this experiment, unlike the following subsection, we use the 4T single-row standard cell library in [13]. This is because 4T single-row standard cells have larger cell sizes

compared to 2T cells and thus have no pin accessibility issues, which enables the experiment to focus solely on the number of DRVs related to routing congestion. We select the LDPC benchmark due to its routing-dominant characteristics, which makes routing congestion effects more pronounced.

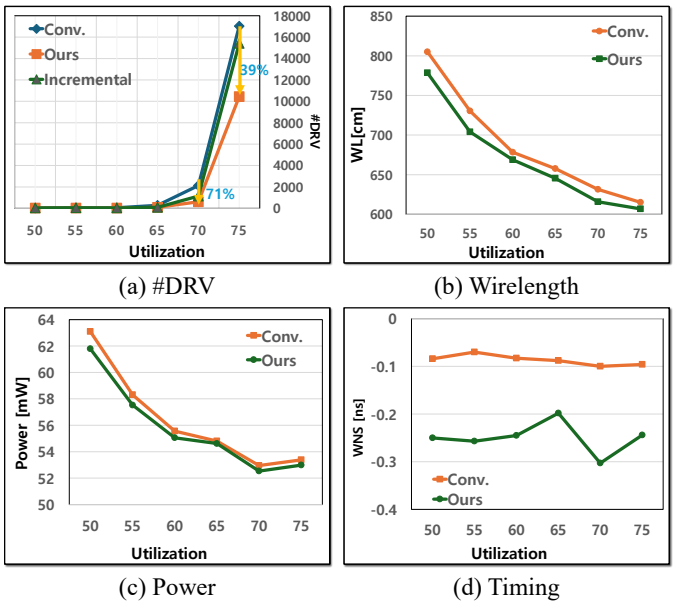


(a) Original (2162 DRVs) (b) Incremental (1143 DRVs) (c) Partial blockage (619 DRVs)

Fig. 10: The layout of the LDPC benchmark designed from: (a) the conventional P&R flow; (b) additionally performed incremental placement *without* partial placement blockages; (c) our proposed framework.

The result shows that our proposed method achieved a 71.3% reduction in DRVs (2161 $\rightarrow$ 619) compared to the conventional P&R result. As an ablation study of our partial placement blockages, we implement another design result of performing incremental placement in the same stage (i.e., after placement / before CTS) without adding partial placement blockages. The design result is shown in Fig.10(b), which also reduces DRVs from the conventional result to some extent (2161 $\rightarrow$ 1143, 47.1% reduction), but contains higher DRVs compared to our proposed method, validating the significance of our approach. We also empirically find that the incremental placement step accounts for approximately 10$\sim$12% of the total runtime, with or without partial placement blockage, and the runtime of both eGR report generation and our clustering-based partial placement blockage generation are within seconds. This relatively low overhead demonstrates the practical applicability of our approach in real design flow.

Fig. 11 presents a comprehensive analysis of our experimental results. Fig. 11(a) shows the DRV count variation, which claims that higher utilization leads to increased cell density, resulting in higher routing congestion followed by increased DRVs. Although the increased utilization reduces the total wirelength since the distance between cell instances becomes smaller (Fig. 11(b)) and also reduces wire-related power consumption (Fig. 11(c)), the DRV count increases dramatically when utilization exceeds 70% and grows exponentially when it goes beyond 80%. Our proposed framework reduces 71.3% of DRVs for 70% utilization (as in Fig. 10), and the reduction effect becomes lower to 39% if the utilization goes up to 75% for the same benchmark. This implies that there is a threshold of utilization that the routing resource becomes absolutely insufficient when using CFET technology, and BEOL scaling in the manufacturing process needs to advance sufficiently before further FEOL scaling. We also observe that our incremental placement step results in slightly degraded timing metrics, due to the unwanted cell displacements for DRV reduction. Nevertheless, our proposed methodology effectively suppresses DRV increases with rising utilization while maintaining comparable design quality.



(a) #DRV (b) Wirelength (c) Power (d) Timing

Fig. 11: Design quality exploration for LDPC benchmark according to various utilizations: #DRV, wirelength, power, and timing.

C. Pin Accessibility Analysis

In this subsection, we analyze the effect of our pin accessibility enhancement method, which utilizes various pin-extended cells. Table I presents the design results for various benchmarks and design options: conventional flow with the baseline CFET cells (CFET_fp); all single-row cells converted to forced-M2-pin cells from the beginning (FM2); all single-row cells converted to whitespace-pin cells from the beginning (WSP); our framework that utilizes both pin-extended cells (Ours).

The FM2 column indicates that DRVs can only be reduced to some extent when all single-row cells are implemented in the forced-M2-pin fashion, with comparable design quality to the baseline results. While two out of three benchmarks achieve up to an 80% reduction in DRVs, the relatively wire-dominated WB_CONMAX benchmark exhibits a much lower reduction ratio. This is because the additional M2 usage by forced-M2-pin cells introduces more DRVs related to routing congestion issues on the M2 layer.

The following WSP column, i.e., design results with changing all single-row cells to whitespace-pin cells, shows the best DRV reduction rate, which goes beyond 90% for all three benchmarks. A whitespace-pin cell nearly doubles the pin access points in a vertical direction without causing any additional routing congestion, while the area is also doubled. As we should design from scratch with all single-row cells in the library substituted to whitespace-pin cells, the design results of all three benchmarks show increased core and standard cell areas for the same utilization, leading to increased wire length compared to the baseline results.

The results from our framework, which starts with force-M2-pin cells and is replaced with whitespace-pin cells when available whitespace exists on the upper or lower standard

TABLE I: Experimental results of each method (FM2: forced-M2-pin cells only, WSP: whitespace-pin cells only, Ours: proposed method.)

Design			CFET_fp [9]	FM2		WSP		Ours	
WB_CONMAX at 1Ghz	*#Total cells* 20603	*Util.*	0.76	0.76	-	0.76	-	0.78	0.02↑
		Core area[μm^2]	1499	1499	-	1714	14.3%↑	1499	-
		Std. area[μm^2]	1139	1138	-	1305	14.3%↑	1176	3.3%↑
	#Single-row cells 8607	*WL*[μm]	123764	123345	0.3%↓	130344	5.3%↑	124281	0.4%↑
		#DRV	10170	6131	**39.7%**↓	457	**95.5%**↓	5575	**45.2%**↓
		WNS[ns]	0.326	0.346	0.02↑	0.422	0.09↑	0.329	-
		Power[mW]	1.91	1.91	-	1.90	-	1.91	-
DES3 at 1Ghz	*#Total cells* 51981	*Util.*	0.70	0.70	-	0.70	-	0.72	0.02↑
		Core area[μm^2]	4173	4173	-	4508	8.0%↑	4173	-
		Std. area[μm^2]	2933	2933	-	3168	8.0%↑	2998	2.2%↑
	#Single-row cells 21461	*WL*[μm]	108350	108928	0.5%↓	111221	2.6%↑	107959	0.4%↓
		#DRV	11253	1279	**88.6%**↓	88	**99.2%**↓	1458	**87.0%**↓
		WNS[ns]	0.753	0.76	-	0.753	-	0.75	-
		Power[mW]	11.29	11.32	0.03↑	11.33	0.04↑	11.31	0.02↑
VGA_LCD at 1.25Ghz	*#Total cells* 37413	*Util.*	0.80	0.80	-	0.80	-	0.81	0.01↑
		Core area[μm^2]	3617	3617	-	3667	1.4%↑	3617	-
		Std. area[μm^2]	2898	2897	-	2940	1.5%↑	2916	0.6%↑
	#Single-row cells 5137	*WL*[μm]	209251	208519	0.3%↓	209157	-	208160	0.5%↓
		#DRV	1857	364	**80.4%**↓	59	**96.8%**↓	193	**89.6%**↓
		WNS[ns]	-1.087	-1.06	0.03↑	-0.769	0.32↑	-1.277	0.19↓
		Power[mW]	16.80	16.85	0.04↑	16.94	0.135↑	16.85	0.043↑

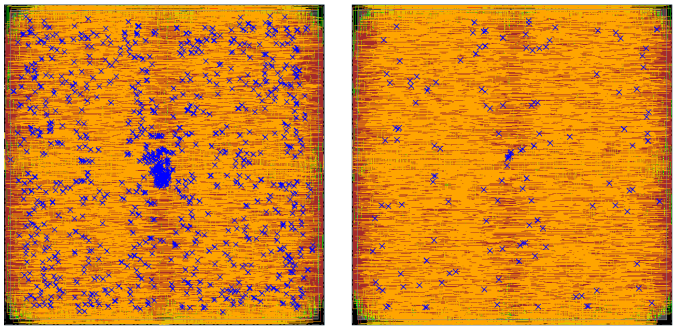
cell row, are shown in the last column. Our framework successfully reduces more DRVs than the forced-M2-pins-only results (FM2) by utilizing whitespace-pin cells that pose no harm to routing congestion. Although the whitespace-pin-cell-only results (WSP) achieve more DRV reduction than our framework, our results offer advantages in terms of design quality, as the use of smaller forced-M2-pin cells does not require additional area and allows for reduced wirelength. Fig. 12 shows the layouts and corresponding DRV markers for each methodology applied to the VGA_LCD benchmark.

D. Discussion

As CFET technology pushes toward extreme scaling, pin access points become severely limited. Here, multi-row cells maximize pin size within available space, which indirectly contributes to overall routability. From this characteristic, we develop a solution for improving pin accessibility in CFET-based designs and suggest the use of pin-extended cells. Together with the alleviation of routing congestion within selective potential congested regions, our proposed framework effectively addresses both routability challenges in CFET-based designs and simultaneously maintains the area benefits of CFET technology.

V. Conclusion

In this paper, we explored the origin of routability problems in CFET-based designs and presented a comprehensive framework for mitigating routability challenges while preserving the inherent area advantages of CFET technology. Our solution addresses both pin accessibility and routing congestion through a novel approach combining various pin-extended cells and strategic partial placement blockage with incremental placement. Experimental results demonstrate significant improvements with up to 71% reduction in routing congestion and an average of 73.9% reduction in pin accessibility violations, while maintaining power consumption and timing performance. Our framework is proven as a practical solution for implementing CFET technology in real-world designs, effectively balancing area efficiency and routability. Our work establishes a foundation for future research in advanced node physical design optimization, particularly for technologies beyond 5nm where routability challenges are expected to become increasingly critical.



(a) Baseline (1857 DRVs)

(b) Forced-M2-pin cells only (364 DRVs)

(c) Whitespace-pin cells only (59 DRVs)

(d) Ours (193 DRVs)

Fig. 12: The layout of the VGA_LCD benchmark designed from four design options in Table I.

ACKNOWLEDGMENT

This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea Government (MSIT) RS-2024-00359696, and RS-2024-00405495. The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

REFERENCES

[1] H.-S. P. Wong, "Beyond the Conventional Transistor," *IBM Journal of Research and Development*, vol. 46, no. 2.3, pp. 133–168, 2002.

[2] T.-H. Wang *et al.*, "Six-track Standard Cell Libraries with Fin Depopulation, Contact over Active Gate, and Narrower Diffusion Break in 7nm Technology," in *International Symposium on Quality Electronic Design (ISQED)*, 2021.

[3] T. Kim *et al.*, "NS3K: A 3-nm Nanosheet FET Standard Cell Library Development and its Impact," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[4] Y. Shin *et al.*, "FS2K: A Forksheet FET Technology Library and a Study of VLSI Prediction for 2nm and Beyond," in *International Symposium on Circuits and Systems (ISCAS)*, 2024.

[5] J. Ryckaert *et al.*, "The Complementary FET (CFET) for CMOS scaling beyond N3," in *Symposium on VLSI Technology (VLSI)*, 2018.

[6] S. M. Y. Sherazi *et al.*, "CFET standard-cell design down to 3Track height for node 3nm and below," in *Advanced Lithography*, 2019.

[7] E. Park and T. Song, "Complementary FET (CFET) Standard Cell Design for Low Parasitics and Its Impact on VLSI Prediction at 3-nm Process," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[8] C.-K. Cheng *et al.*, "Multirow complementary-fet (cfet) standard cell synthesis framework using satisfiability modulo theories (smts)," *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits*, 2021.

[9] S. Kim and T. Kim, "Optimal Transistor Folding and Placement for Synthesizing Standard Cells of Complementary FET Technology," in *Design Automation Conference (DAC)*, 2024.

[10] L. T. Clark *et al.*, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, 2016.

[11] C.-K. Cheng *et al.*, "Complementary-FET (CFET) Standard Cell Synthesis Framework for Design and System Technology Co-Optimization Using SMT," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2021.

[12] O. Kulkarni and A. Burhanpurwala, "A Survey of Advancements in DBSCAN Clustering Algorithms for Big Data," in *International Conference on Power Electronics and IoT Applications in Renewable Energy and its Control (PARC)*, 2024, pp. 106–111.

[13] S. Kim, "CFET_fp," https://github.com/SNU-suwan/CFET_fp, gitHub repository.

[14] S. Choi *et al.*, "PROBE3.0: A Systematic Framework for Design-Technology Pathfinding With Improved Design Enablement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[15] Oliscience, "Opencores." [Online]. Available: https://opencores.org

Optimizing Recovery Logic in Speculative High-Level Synthesis

Dylan Leothaud*, Jean-Michel Gorius*, Simon Rokicki*, Steven Derrien†
*Univ Rennes, Inria, CNRS, IRISA
†UBO, Lab-STICC

***Abstract*—High-Level Synthesis (HLS) excels at handling compute-intensive loops with straightforward control but struggles to identify parallelism in kernels with complex and irregular control-flow. To address this, novel scheduling techniques based on speculation have been introduced. While these methods outperform traditional static scheduling, they also introduce significant area overhead, particularly in the rollback control logic. Optimizing the cost of this rollback control logic remains an open challenge. In this work, we show how it is possible to simplify and/or eliminate rollback logic using a combination of static analysis and linear programming. Our results show improvements in both execution throughput and area cost.**

I. Introduction

High-Level Synthesis (HLS) has emerged as a powerful technique for designing hardware accelerators, with the potential to rival manual designs. However, current HLS tools still struggle with complex data-dependent control flow and memory accesses, largely due to the use of static scheduling techniques that cannot utilize runtime information.

There has been growing research interest in extending HLS tools to support dynamic [1]–[8] and speculative [5], [8]–[11] execution mechanisms to address the shortcomings of static scheduling. While dynamic scheduling has been studied extensively, speculative scheduling has received comparatively less attention, partly because of the significant area and clock speed overheads introduced by the speculative circuits generated by corresponding toolchains, which often outweigh the potential performance benefits. Reducing overhead in speculative circuits is a fundamental issue that needs to be addressed to make speculative execution techniques practical for HLS. The recovery logic, which tracks program states and restores the last valid state in the event of a misspeculation, is the primary source of this overhead.

This work shows that current approaches result in unnecessarily complex recovery logic. We address this issue by proposing a set of techniques to optimize the rollback logic and its placement within the datapath. Specifically, our contributions are as follows:

- We introduce two transformations to simplify or eliminate unecessary checkpoint/recovery logic.
- We formulate and solve the problem of minimum cost checkpointing.
- We validate our approach experimentally, showing an average throughput improvement of 23%, and an average resource utilization reduction of 10%.

This paper is organized as follows. Section II provides background information on speculative scheduling. Section III presents our approach, Section IV addresses its experimental validation, and discusses related work. Section VI concludes the paper.

II. Background

This section recalls key concepts relevant to HLS scheduling, focusing on speculative pipelining techniques and their associated recovery mechanisms.

A. Scheduling in High-Level Synthesis

Custom hardware relies on parallelism and customization to achieve high performance. From a compiler's perspective, *identifying* and *exploiting* parallelism is challenging. Notably, *loop pipelining* [12] has proven to be a critical transformation in HLS due to its ability to automatically generate deep and wide pipelined datapaths, even in the presence of loop-carried dependencies. However, some kernels cannot benefit from loop pipelining, often because of unpredictable control flow or memory access patterns. Speculative (and dynamic) loop pipelining (SLP) techniques enable more efficient implementations by anticipating the outcome of conditionals and memory accesses.

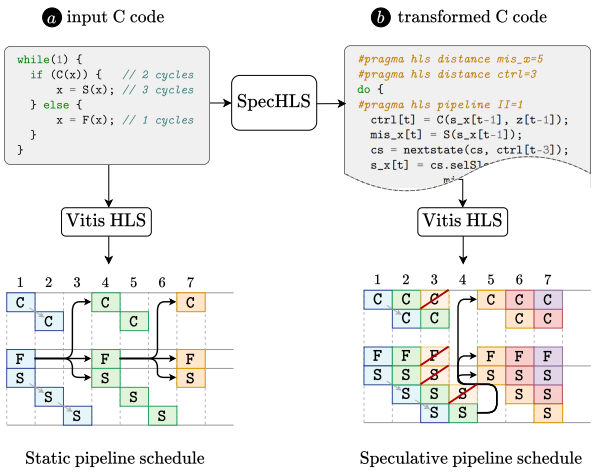


Fig. 1. Speculative pipelining as a source-level transformation in SpecHLS.

The SpecHLS flow enables SLP by extending existing HLS toolchains through source-level transformations to support dynamic and speculation execution [9], [11]. Figure 1 illustrates this approach on a simple loop where the execution path of an

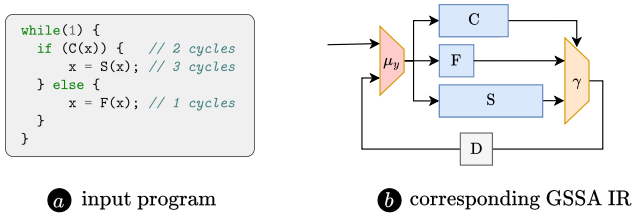
iteration depends on the outcome of a data-dependent decision. In such cases, SOTA HLS tools will assume the worst-case scenario and derive a schedule with an Initiation Interval (II) of $\mathrm{II} = 3$, as illustrated in part ⓐ. In contrast, SpecHLS produces optimized C++ code that takes advantage of speculation by considering that the `F` path is always taken, leading to a design with $\mathrm{II} = 1$. When a mispeculation occurs, the loop cancels its pending operations and restarts from a valid state.

B. The SpecHLS flow and its compiler IR

The SpecHLS flow uses a variant of the Gated-SSA representation (GSSA) [13], [14] as its core program Intermediate Representation (IR). In GSSA, the φ-nodes of traditional SSA form [15] are replaced with *gating nodes*. The type of gating node that replaces a given φ-node depends on the context in which the φ-node appears. Gated-SSA enables us to work with fully-predicated φ-nodes, represented as μ- and γ-nodes. The following gating nodes are particularly relevant to our discussion:

- μ-nodes are placed in loop headers. They take three arguments, $\mu(p, i_x, l_x)$, where p is the loop exit condition, i_x is the initial value of variable x, and l_x is the value of x after a loop iteration.
- γ-nodes are placed at join points in the control-flow graph, such as the end of conditional structures. They encode a predicate, determining which value to select when execution reaches them. We denote them by $\gamma(c, x_0, x_1, \ldots, x_n)$, where c is the predicate, and $x_0, x_1, \ldots, x_n$ are the possible values for variable x.
- delay nodes (δ) capture inter-iteration producer/consumer relations. For example, data reuse spanning two iterations will be represented as a 2δ node.

Figure 2 illustrates the GSSA operators utilized in this work. For clarity, our examples omit the loop exit condition from μ-nodes.



ⓐ input program ⓑ corresponding GSSA IR

Fig. 2. Extended Gated-SSA operators. μ-nodes represent loop headers, and γ-nodes encode control-flow decisions. Inter-iteration dataflow is modeled using δ-nodes.

C. Speculation as an IR transformation

Because they capture conditionals, γ-nodes expose speculation opportunities. For instance, if we target $T_{clk} = 10\,\mathrm{ns}$ and assume the existence of pipelined implementations for the operators F, C, and S, with latencies of 1, 2, and 3 cycles respectively, speculating that the γ-node in Figure 2 ⓑ selects the result from the F operation allows pipelining with an $\mathrm{II} = 1$, leading to the speculative pipelined schedule depicted in part ⓑ of Figure 1.

To be considered for speculation, a given γ-node must (i) expose a long delay over at least one of its data or control inputs, (ii) expose a short delay over one of its input data ports, and (iii) the probability for the latter path to be taken must be higher than a given threshold. Determining which γ-nodes to speculate is a complex problem, which has been addressed in prior work [8].

The key idea behind the SpecHLS transformation flow is to expose the pipeline depth directly in the IR by extending the reuse distance through explicit delays (δ nodes), weave the appropriate hazard recovery logic, and then let the HLS back-end tool map each loop kernel operations to a specific pipeline stage. More precisely, the transformation includes two key steps:

1) Delays (δ-nodes) are inserted on non-speculated paths (e.g., path involving `S` and `C`) as depicted in Figure 3 ⓑ. Those delays increase the reuse distance for loop-carried dependencies and allow the back-end HLS toolchain scheduler to obtain an initiation interval of $\mathrm{II} = 1$.
2) Recovery logic is also inserted to handle mispeculations. This logic consists of (i) rollback nodes (R nodes), with an internal buffer of a given *depth*, for restoring past inputs, and a Finite State Machine (FSM) depicted in Figure 3 ⓑ.

The transformation is illustrated in part ⓑ of Figure 3, where two extra delays have been added along the slow path, one along the conditional path, and a rollback node of depth three is used to store the last three values for variable `x`. The ways the HLS back-end uses these additional nodes is illustrated in part ⓒ of Figure 3, where the rollback node has been lowered into a combination of a shift register and a multiplexer, and where the static pipeline scheduler has moved delays to enable full pipelining.

The SpecHLS flow supports multiple concurrent speculations [11]. The flow also handles speculative memory accesses, which do not require rollback nodes (they use some form of LSQ instead [16]). Detailing the whole flow is out of the scope of the present work, and the reader is invited to refer to previous work on the topic [9], [11]. Since this work is focused on recovery optimizations, we describe where and how rollback nodes are currently inserted in the SpecHLS flow:

- Rollback nodes are inserted at the output of all μ nodes in the Strongly Connected Component (SCC) of the GSSA graph containing the speculated gammas.
- In the case of multiple speculations (*i.e.*, multiple γ-nodes within the same SCC), rollback nodes are inserted at the output of all speculated γ-nodes.
- The recovery depth of the rollback node is obtained by determining the longest path (in terms of clock cycles) in the datapath for a mispeculation to be detected (2 cycles in our example)

D. Problem statement

A significant fraction of the area overhead caused by SLP is linked to the mispeculation recovery process, which needs

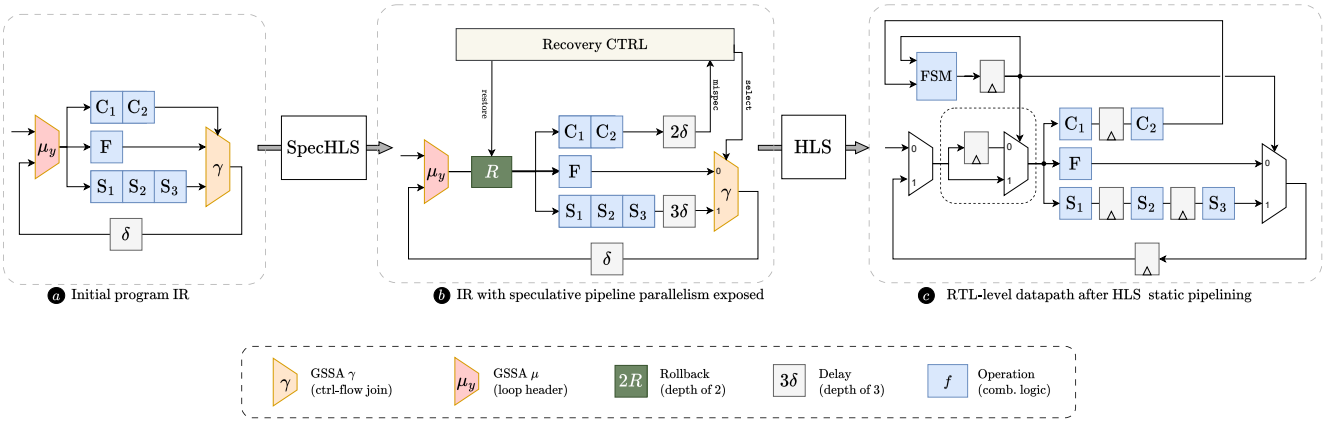


Fig. 3. Overview of the program IR transformations involved in the SpecHLS compilation flow.

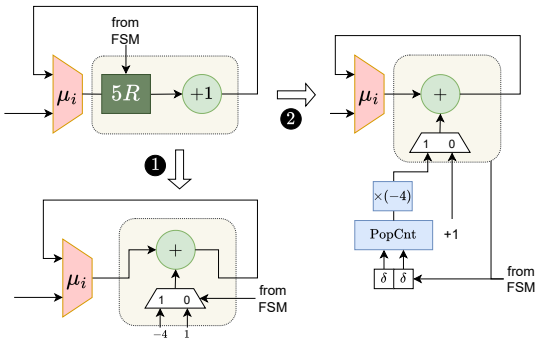


Fig. 4. Rollback simplification for a loop increment operation.

to keep track of previous loop iteration states (through multiplexers and shift registers) and manage the recovery sequence using dedicated control signals from the FSM. In addition, rollback nodes also negatively impact clock speed as they end up being part of the new IR critical path. The problem we address in this work is minimizing this overhead through a set of transformations, which we describe in the following section.

III. Proposed Approach

In practice, rollback nodes can be placed at different locations without impacting correctness, and in many cases, this logic can be optimized. In the following, we describe these optimizations and show how we can formulate a global optimization problem using a simple Linear Programming model.

A. Rollback simplification

Sometimes, a variable can be reverted to a previous state without saving its prior values. This optimization applies to *reversible* operations, *i.e.*, computations where the original input can be reconstructed from the output. In other words, given the result of the operation, it is possible to reverse the process and precisely recover the input. Increment operations follow this pattern: their rollback can be performed using a simple subtraction, where the value to be subtracted is either a constant or can be computed with minimal resource overhead.

Consider, for example, a kernel snippet that increments a variable `i` by 1 during each iteration, along with its GSSA intermediate representation, as shown in Figure 4. Two scenarios need to be considered depending on whether multiple speculations are involved.

- In the case of a single speculation ❶, the datapath will either increment `i` by `1` or recover from a speculation through a rollback. A rollback with a depth d simply requires decrementing the variable by $1 \times d$. This subtraction operator can then be merged with the subsequent adder to produce the simplified circuit shown as the result transformation ❶.
- The situation becomes more complex with multiple speculations ❷, since several speculative execution states may overlap. In such cases, tracking the number of pending rollbacks within the speculation window is necessary to determine the value to subtract. This tracking can be effectively implemented using a single-bit-wide shift register whose depth corresponds to the rollback maximum depth (to keep pending rollback requests). This shift register is paired with a pop-count structure, as depicted in transformation ❷, whose result is used to determine the actual iterator value. This structure can be easily generalized to support rollbacks with multiple depths at a slight increase in area cost.

B. Rollback elimination

Rollbacks revert execution to previous iterations through checkpointing, ensuring the datapath can restart from a valid state after recovering from misspeculation. In the SpecHLS flow, rollbacks are inserted at the output of every μ-nodes and, in cases of multiple speculations, at the output of certain γ-nodes. While this method guarantees correctness, it can be overly conservative, as the output of the rollback node may not always be necessary.

This scenario is demonstrated in part ⓐ of Figure 5, where a rollback is inserted after the μ-nodes. This rollback is associated with the misspeculation recovery process from the γ-node. In this example, the reader will notice that the output of the rollback does *never* contribute to the actual result because it will always be overshadowed by the output of the `F` function (if no misspeculation occurred) or the `S` functions (when recovering

from a mispeculation). This rollback node can, therefore, safely be removed without affecting the program's correctness.

Such a simplification is, however, not always possible, as demonstrated in part (b) of Figure 5, where the output of the μ-node is forwarded beyond the γ-node through the `H` operation. This rollback node is mandatory in such a situation since it will contribute to the next value of x during the recovery.

More formally, the condition for removing a rollback node is based on postdominance properties in the graph. Postdominance can be defined as follows: a node d in a graph is said to *postdominate* another node n if every path to the exit node that starts at n also has to go through d. In our context, a rollback node can be eliminated when it is post-dominated by a speculative γ-node.

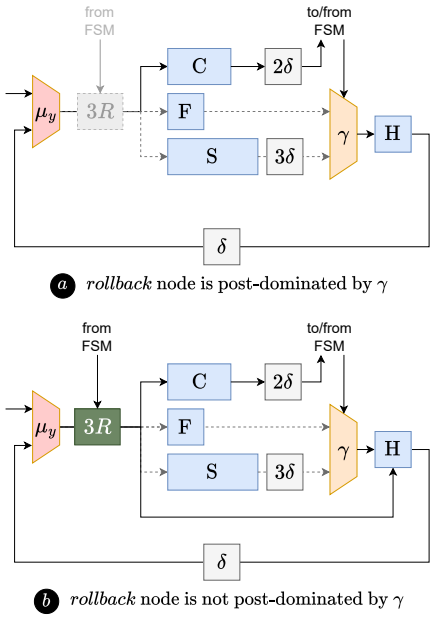


Fig. 5. Rollback elimination based on post-domination.

The issue becomes more complex when dealing with multiple speculated γ-nodes, as a rollback node, can only be eliminated if it is post-dominated by all of the speculated γ-nodes, which is rarely the case. However, our implementation addresses several specific scenarios to enhance the effectiveness of the analysis. For instance, when multiple speculated γ-nodes share the same speculated control signal, they are treated as a single node for the post-domination analysis.

C. Rollback placement

As discussed in Section II, the SpecHLS flow places rollback nodes in the IR graph at specific points (after μ-and γ-nodes). However, this placement is somewhat arbitrary, as alternative placement options can ensure correctness while potentially reducing area overheads. For example, rollback nodes can be moved forward in the IR graph, as illustrated in Figure 6. To be valid, the move must enforce certain conditions:

- A rollback node r may only be moved past another node s in the graph if all predecessors of r are also rollback nodes. Figure 6 depicts such a transformation. This rule ensures consistency in the information flow within the IR graph. This transformation strongly resembles the well-known register retiming optimization [17].
- Rollback moves must not cross inter-iteration boundaries to avoid de-synchronizing the control signal from the FSM, which could otherwise arrive one or more cycles too late. Consequently, we also prohibit rollback moves across delay and μ-nodes, which mark iteration boundaries.

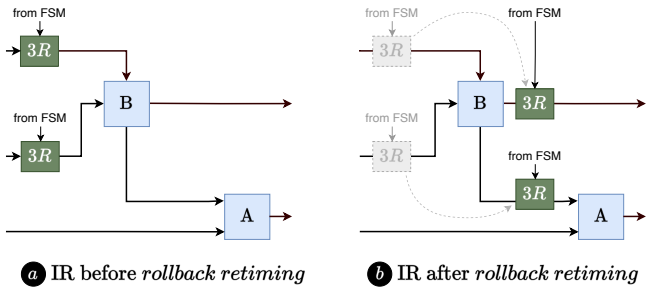


Fig. 6. Example of *forward* retiming of a *rollback* node.

As discussed previously, simplifying or eliminating rollback nodes depends on their position within the graph. Since these nodes can be moved using the transformations described above, we aim to derive a set of elementary moves that achieve the optimal placement of all rollback nodes in the graph.

This optimization is illustrated in Figure 7. In this example, two variables (`x` and `y`) are updated at every iteration (as shown in (a)). Assuming that a speculation decision (of depth 3) is applied to the γ-node, two rollback nodes (with a cost of 3 registers each) must be inserted at the μ nodes (as shown in (b)), resulting in a cost of $2 \times 3 = 6$ registers. It is, however, possible to move those rollback nodes past `B` and `A` nodes to obtain an optimized IR with a cost of 3 registers, since two of them can be eliminated (they become post-dominated by the γ-node), as illustrated in (c). In the following, we show how it is possible to derive such an optimal set of moves automatically.

D. Rollback minimization as a retiming problem

As mentioned previously, the rollback move operation can be seen as a form of retiming [17], a technique used in digital circuit optimization where the positions of registers within a circuit are adjusted to optimize both the number of registers and the clock period. Our rollback placement problem is very similar to the registers cost minimization problem, which we recall below.

Consider a synchronous digital circuit represented by a directed graph $G = (V, E)$, where:

- V is the set of vertices representing the computational nodes in our program representation.
- E is the set of directed connection edges between these blocks, where each edge $(u, v) \in E$ has a weight $w(u, v)$ indicating the number of delays on that edge, and a cost $c(u, v)$ which accounts for the edge data's bit-width.

Retiming involves shifting registers from one edge to another without changing the logical behavior of the circuit. Let

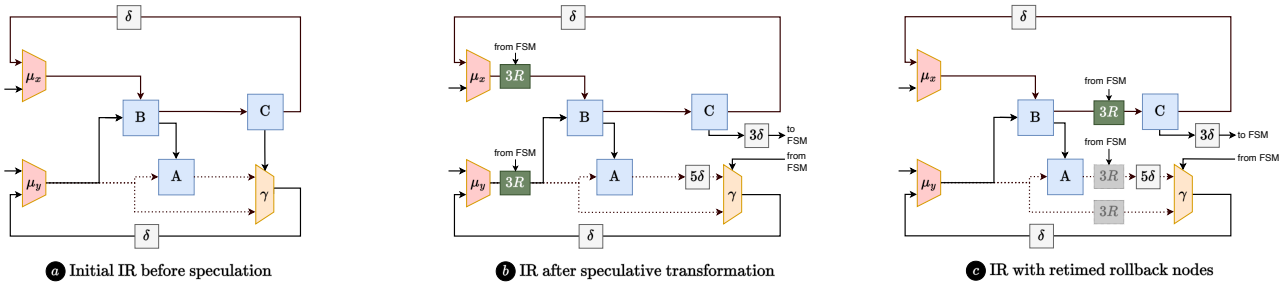


Fig. 7. Example of rollback optimization.

$r : V \to \mathbb{Z}$ be a retiming function, where $r(v)$ represents the retiming value (i.e., the number of registers added to or removed from the edges incident to vertex v).

The register minimization problem can be modeled using a Linear Programming formulation, which can be solved in polynomial time [17]. The objective is to minimize the cost of registers in the circuit after retiming. The LP problem is as follows:

$$\text{Minimize} \quad \sum_{(u,v)\in E} c(u,v) \times (w(u,v) + r(v) - r(u))$$

After retiming, we must also ensure that the number of registers on any edge (u, v) must be non-negative:

$$r(v) - r(u) \leq w(u,v) \quad \forall (u,v) \in E$$

We can reuse this formulation as is for our rollback optimization problem, but with the following modifications.

- we model a rollback node of depth d as a d-deep retiming register. We also prevent retiming over delay nodes and force $r(v) = 0$ for such nodes.
- for edges whose successors are post-dominated by a speculated gamma node (and for which rollback nodes can be eliminated), we set $c(u, v) = 0$.

IV. Experimental validation

This Section evaluates the relevance of our approach from a qualitative and quantitative perspective. We aim to demonstrate that our transformations reduce area costs and improve clock speed by reducing the designs' critical path length. We do not, however, explore the trade-offs implied by the use of speculation in HLS designs, since those orthogonal issues have been discussed in prior works [9]–[11].

A. Experimental Setup

Our technique is implemented within the SpecHLS source-to-source compiler [11], which is based on the GeCoS framework [18], coupled to the `lpSolve` solver [19]. We use Vitis HLS 2022.2 with an XC7A25 FPGA as the target device, with a target clock period of 10 ns. Existing HLS benchmarks [20], [21] primarily focus on kernels with regular control-flow and memory access patterns, favoring applications that work well with state-of-the-art HLS tools. These benchmarks are not suitable for evaluating our approach. Instead, we selected a set of benchmarks that exhibit data-dependent control flow and memory accesses, where speculation can be applied. These benchmarks were collected from previously published work [1], [11], [22]–[24].

We compare resource usage and performance for each benchmark across four implementations: a baseline implementation (based solely on Vitis HLS), shown in gray, and three SpecHLS versions. The first SpecHLS version (in green) does not incorporate our proposed rollback optimization strategies, the second one (in blue) includes our rollback optimization strategy, and the third (in orange) is a speculative version where we assume no misspeculation occurs, allowing the removal of all recovery logic. We use this last version to represent the hypothetical maximum reduction in recovery logic area cost.[1]

B. Experimental results

Resource usage after RTL synthesis, and for a target clock of $T_{\text{clk}} = 10$ ns is shown in Figure 8 and is reported in terms of LUTs, FFs, SRLs, BRAMs, and DSPs for each benchmark. Performance, shown in Figure 8, is measured in terms of absolute running time. The baseline's running time is computed as $T_{\text{exec}} = \text{II} \times T_{\text{clk}}$. In contrast, for the speculative versions, it is calculated as $T_{\text{exec}} = \text{CPI} \times T_{\text{clk}}$, where CPI (Cycles Per Iteration) accounts for misspeculation penalties. For benchmarks where the misspeculation rate depends on the dataset, results may vary depending on the misspeculation rate. We illustrate this phenomenon on the `hist` benchmark at the bottom of Figure 8, where we vary the misspeculation rate (2%, 10%, 30%) following the approach suggested by Derrien *et al.* [9] to reflect the impact of these datasets on performance.

Results show that our approach improves area and performance for all but one benchmark, with an average throughput improvement of 13% and a reduction in FF+LUT+SRL area of 10%. There is, however, one outlier: for the `nr` benchmark, our analysis failed to identify any rollback optimization opportunities. Upon conducting a manual analysis, we found a rollback node that could have been eliminated, but detecting this would require more complex analyses (such as model checking or SMT solvers), which are beyond the scope of this work.

We note that speculative loop pipelining is a *last mile* optimization, which implies that is intended to extract parallelism from applications that are inherently hard to optimize for HLS

[1]The area overhead, when compared to the golden version, is due to the fact that designs with a unit initiation interval ($\text{II} = 1$) use more resources than those with $\text{II} > 1$, as full pipelining prevents resource reuse.

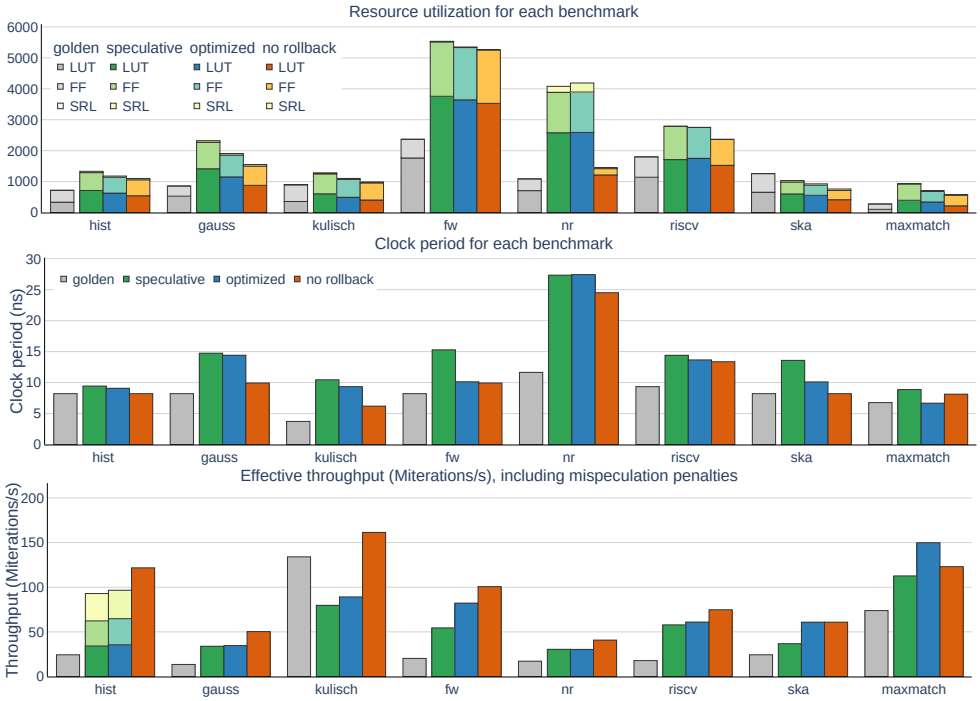


Fig. 8. Resource utilization, clock period, and effective throughput for our set of benchmarks on an XC7A25 FPGA device. The throughput graph for the speculative versions of `hist` shows the effective throughput for a mispeculation rate of 2%, 10%, and 30%, using overlaid bars, from highest to lowest. The version without rollbacks assumes a mispeculation rate of 0%.

toolchains. Consequently, the benefits one can obtain from using speculative HLS techniques often outweigh the potential scalability concerns that other more systematic compiler optimizations (such as loop transformations) may face in terms of usability. Evaluation of our approach on different boards and larger designs is left as future work.

V. Related work and discussion

This section covers three key points: (i) how our approach differs from existing work on speculative and dynamic HLS, (ii) the distinctions between our rollback optimization technique and compiler optimizations for checkpointing and recovery in Thread-Level Speculation for multi-core systems, and (iii) how our objectives differ from checkpointing mechanisms in large-scale HPC systems.

Several works have tackled the challenge of leveraging speculation in HLS tools, with speculative dataflow circuits [10] standing out as one of the most advanced approaches. The latter introduce speculative execution capabilities into elastic hardware designs. Speculative data tokens, generated by *speculators*, propagate through a network of elastic components synchronized via handshake signals. While this token-based design is effective, it is less general than SpecHLS's approach, since it cannot handle multiple intertwined speculations [11]. Furthermore, the distributed nature of speculative dataflow circuits limits rollback optimization opportunities by restricting recovery to computational replay.

There has also been extensive research on speculative parallelization, mainly focusing on recovery mechanisms and enhancing the efficiency of speculative execution in multi-threaded environments for multi-core processors [25]. For example Tian *et al.* [26] propose a speculative parallelization framework incorporating incremental recovery, which reduces the overhead from speculative failures by rolling back only the minimal necessary computations. Yagüez *et al.* [27] explore different squashing alternatives in software-based speculative parallelization, evaluating strategies for handling mispeculations and assessing their impact on system performance. However, these approaches operate at a much coarser granularity than the iteration level, making them unsuitable for our context.

Checkpointing and recovery mechanisms are also commonly used to address hardware failures in HPC or intermittent systems, with several works proposing compiler support for such mechanisms [28]–[30]. However, these approaches typically operate at the software level (with some exceptions [31]), work at a very coarse granularity, or assume that such failures are rare events, making them significantly different from the fine-grained, frequent speculation context we address.

VI. Conclusion

Speculative HLS opens up interesting new design perspectives, as it allows for the acceleration of complex, control-dominated kernels. It does, however, come at a significant area cost. In this paper, we show how to reduce the area overhead of the rollback logic in speculatively scheduled hardware designs. Through a combination of static analysis and linear programming, we are able to synthesize smaller designs while simultaneously improving the throughput of the synthesized hardware by 13% on average, with some benchmarks exhibiting more than 40% performance improvement.

ACKNOWLEDGMENT

This work is partly funded by the French National Research Agency (ANR), as part of the LOTR project[2] (ANR-23-CE25-0016)

REFERENCES

[1] M. Alle, A. Morvan, and S. Derrien, "Runtime Dependency Analysis for Loop Pipelining in High-Level Synthesis," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2463209.2488796

[2] J. Cong, J. Lau, G. Liu, S. Neuendorffer, P. Pan, K. Vissers, and Z. Zhang, "FPGA HLS Today: Successes, Challenges, and Opportunities," *ACM Transactions on Reconfigurable Technology and Systems*, vol. 15, no. 4, pp. 51:1–51:42, Aug. 2022. [Online]. Available: https://dl.acm.org/doi/10.1145/3530775

[3] S. Dai, R. Zhao, G. Liu, S. Srinath, U. Gupta, C. Batten, and Z. Zhang, "Dynamic hazard resolution for pipelining irregular loops in high-level synthesis," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 189–194. [Online]. Available: https://doi.org/10.1145/3020078.3021754

[4] L. Josipović, R. Ghosal, and P. Ienne, "Dynamically Scheduled High-level Synthesis," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 127–136. [Online]. Available: https://dl.acm.org/doi/10.1145/3174243.3174264

[5] V. Lapotre, P. Coussy, C. Chavet, H. Wouafo, and R. Danilo, "Dynamic branch prediction for high-level synthesis," in *2013 23rd International Conference on Field programmable Logic and Applications*, 2013, pp. 1–6.

[6] J. Cheng, J. Wickerson, and G. A. Constantinides, "Probabilistic Scheduling in High-Level Synthesis," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 195–203.

[7] R. Szafarczyk, S. W. Nabi, and W. Vanderbauwhede, "Compiler Discovered Dynamic Scheduling of Irregular Code in High-Level Synthesis," in *2023 33rd International Conference on Field-Programmable Logic and Applications (FPL)*. Los Alamitos, CA, USA: IEEE Computer Society, sep 2023, pp. 1–9. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/FPL60245.2023.00009

[8] D. Leothaud, J.-M. Gorius, S. Rokicki, and S. Derrien, "Efficient design space exploration for dynamic & speculative high-level synthesis," in *34th International Conference on Field-Programmable Logic and Applications*, 2024.

[9] S. Derrien, T. Marty, S. Rokicki, and T. Yuki, "Toward Speculative Loop Pipelining for High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 4229–4239, Nov. 2020.

[10] L. Josipović, A. Guerrieri, and P. Ienne, "Speculative Dataflow Circuits," in *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 162–171.

[11] J.-M. Gorius, S. Rokicki, and S. Derrien, "SpecHLS: Speculative Accelerator Design Using High-Level Synthesis," *IEEE Micro*, vol. 42, no. 5, pp. 99–107, 2022.

[12] M. Lam, "Software Pipelining: An Effective Scheduling Technique for VLIW Machines," in *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation*, ser. PLDI '88. New York, NY, USA: Association for Computing Machinery, 1988, p. 318–328. [Online]. Available: https://doi.org/10.1145/53990.54022

[13] P. Tu and D. Padua, "Efficient building and placing of gating functions," in *Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*, ser. PLDI '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 47–55.

[14] ——, "Gated SSA-Based Demand-Driven Symbolic Analysis for Parallelizing Compilers," in *Proceedings of the 9th International Conference on Supercomputing*, ser. ICS '95. New York, NY, USA: Association for Computing Machinery, 1995, p. 414–423. [Online]. Available: https://doi.org/10.1145/224538.224648

[15] R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently computing static single assignment form and the control dependence graph," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 13, no. 4, pp. 451–490, 1991.

[16] J.-M. Gorius, S. Rokicki, and S. Derrien, "A Unified Memory Dependency Framework for Speculative High-Level Synthesis," in *Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction*, ser. CC 2024. New York, NY, USA: Association for Computing Machinery, 2024, p. 13–25. [Online]. Available: https://doi.org/10.1145/3640537.3641581

[17] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.

[18] A. Floc'h, T. Yuki, A. El-Moussawi, A. Morvan, K. Martin, M. Naullet, M. Alle, L. L'Hours, N. Simon, S. Derrien, F. Charot, C. Wolinski, and O. Sentieys, "GeCoS: A framework for prototyping custom hardware design flows," in *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2013, pp. 100–105.

[19] M. Berkelaar *et al.*, "Package 'lpsolve'," 2015.

[20] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "CHStone: A benchmark program suite for practical C-based high-level synthesis," in *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2008, pp. 1192–1195.

[21] Y. Zhou, U. Gupta, S. Dai, R. Zhao, N. Srivastava, H. Jin, J. Featherston, Y.-H. Lai, G. Liu, G. A. Velasquez, W. Wang, and Z. Zhang, "Rosetta: A Realistic High-Level Synthesis Benchmark Suite for Software Programmable FPGAs," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 269–278. [Online]. Available: https://doi.org/10.1145/3174243.3174255

[22] J. Cheng, L. Josipović, G. A. Constantinides, P. Ienne, and J. Wickerson, "DASS: Combining Dynamic & Static Scheduling in High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 3, pp. 628–641, 2022.

[23] J. Liu, J. Wickerson, S. Bayliss, and G. A. Constantinides, "Polyhedral-Based Dynamic Loop Pipelining for High-Level Synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1802–1815, Sep. 2018.

[24] J.-M. Gorius, S. Rokicki, and S. Derrien, "Design Exploration of RISC-V Soft-Cores through Speculative High-Level Synthesis," in *2022 International Conference on Field-Programmable Technology (ICFPT)*, 2022, pp. 1–6.

[25] N. Vachharajani, R. Rangan, E. Raman, M. J. Bridges, G. Ottoni, and D. I. August, "Speculative Decoupled Software Pipelining," in *16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007)*, 2007, pp. 49–59.

[26] C. Tian, C. Lin, M. Feng, and R. Gupta, "Enhanced speculative parallelization via incremental recovery," *ACM SIGPLAN Notices*, vol. 46, no. 8, pp. 189–200, 2011.

[27] Á. G. Yágüez, D. R. Llanos, and A. Gonzalez-Escribano, "Squashing alternatives for software-based speculative parallelization," *IEEE Transactions on Computers*, vol. 63, no. 7, pp. 1826–1839, 2013.

[28] B. Yarahmadi and E. Rohou, "Compiler optimizations for safe insertion of checkpoints in intermittently powered systems," in *Embedded Computer Systems: Architectures, Modeling, and Simulation: 20th International Conference, SAMOS 2020, Samos, Greece, July 5–9, 2020, Proceedings 20*. Springer, 2020, pp. 169–185.

[29] S.-E. Choi and S. J. Deitz, "Compiler support for automatic checkpointing," in *Proceedings 16th Annual International Symposium on High Performance Computing Systems and Applications*. IEEE, 2002, pp. 213–220.

[30] X. Yang, P. Wang, H. Fu, Y. Du, Z. Wang, and J. Jia, "Compiler-assisted application-level checkpointing for mpi programs," in *2008 The 28th International Conference on Distributed Computing Systems*. IEEE, 2008, pp. 251–259.

[31] A. Bourge, O. Muller, and F. Rousseau, "Automatic high-level hardware checkpoint selection for reconfigurable systems," in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2015, pp. 155–158.

[2] https://lotr.irisa.fr/

Segmented Angular Pre-Processing for Accurate and Efficient In-Memory Vector Similarity Search

Chi-Tse Huang[1], Jen-Chieh Wang[1], Hsiang-Yun Cheng[2], and An-Yeu (Andy) Wu[1]
[1] Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan
[2]Research Center for Information Technology Innovation, Academia Sinica, Taipei, Taiwan
rickhuang@access.ee.ntu.edu.tw; jack@access.ee.ntu.edu.tw; hycheng@citi.sinica.edu.tw; andywu@ntu.edu.tw

***Abstract*—Vector similarity search (VSS) is a fundamental operation in modern AI applications, including few-shot learning (FSL) and approximate nearest neighbor search (ANNS). Cosine similarity is widely regarded as the optimal metric for VSS. However, VSS incurs substantial energy and computational overhead, primarily due to frequent vector transfers and the complexity of cosine similarity calculations in high-dimensional spaces. Prior research has explored the use of ternary content addressable memories (TCAMs) for parallel in-memory VSS to reduce vector movement. Exact-Match TCAM (EX-TCAM) enables exact bit-matching, and Best-Match TCAM (Best-TCAM) supports Hamming distance calculations, both of which are spatial metrics and computationally efficient. As a result, existing TCAM-based VSS approaches have focused on developing frameworks to efficiently support more complex spatial metrics such as the L_∞ and L_1 norms. However, these spatial metrics exhibit notable discrepancies compared to angular metrics like cosine similarity. To overcome this limitation, we propose Seg-Cos, a TCAM-based framework that directly approximates cosine similarity within TCAM for angular VSS. Seg-Cos introduces a dedicated pre-processing technique and encoding scheme that segments vectors and encodes them as circular ranges based on their angles and magnitudes. Seg-Cos is the first angular VSS framework compatible with both EX-TCAM and Best-TCAM, enabling accurate and energy-efficient VSS in the angular domain. Simulation results demonstrate that Seg-Cos improves energy efficiency by 1.41× and achieves up to 2.2% higher accuracy over prior EX-TCAM-based methods in FSL. In ANNS, Seg-Cos enhances recall rate by 10% to 52% and improves energy efficiency by 2× compared to previous Best-TCAM approaches with L_1 norm.**

I. Introduction

Vector similarity search (VSS) serves as a foundational operation in modern artificial intelligence (AI) applications, including few-shot learning (FSL) [1], [2] and approximate nearest neighbor search (ANNS) [3]. These applications leverage VSS to address challenges such as catastrophic forgetting in FSL [1], [2] and hallucinations in generative AI [4]. By comparing vectors in high-dimensional spaces, VSS enables critical tasks, including classifying novel instances and retrieving relevant information from large databases. A key metric for these tasks is cosine similarity, which effectively measures angular relationships between vectors.

Despite its efficacy, the deployment of VSS is hindered by the substantial energy consumption [5], [6] associated with the frequent transfer of vectors between off-chip memories and processing units in traditional von Neumann systems. In pursuit of energy-efficient solutions in VSS, researchers have been exploring in-memory search (IMS) [7]–[14]. IMS aims to minimize vector movement and accelerate the VSS process by performing parallel searches directly within memory. Recently, two primary IMS-based VSS approaches have been developed using ternary content-addressable memories (TCAMs): 1) Exact-Match TCAM (EX-TCAM) [7]–[10] and 2) Best-Match TCAM (Best-TCAM) [11]–[14]. EX-TCAM locates stored vectors that exactly match a given query vector, whereas Best-TCAM extends EX-TCAM by enhancing its sensing circuitry to identify the closest stored vector. As a result, EX-TCAM achieves significantly lower power consumption per search request compared to Best-TCAM. EX-TCAM-based VSS operates in an iterative manner to support the L_∞ norm [7]–[10] using range encoding [7], as illustrated in the left part of Fig. 1(a). On the other hand, Best-TCAM-based VSS [11]–[14] enables single-pass identification of the nearest vector. Best-TCAM originally supports the Hamming distance [11], [12] and extends to the L_1 norm with thermometer encoding [13], [14], as shown in the right part of Fig. 1(a).

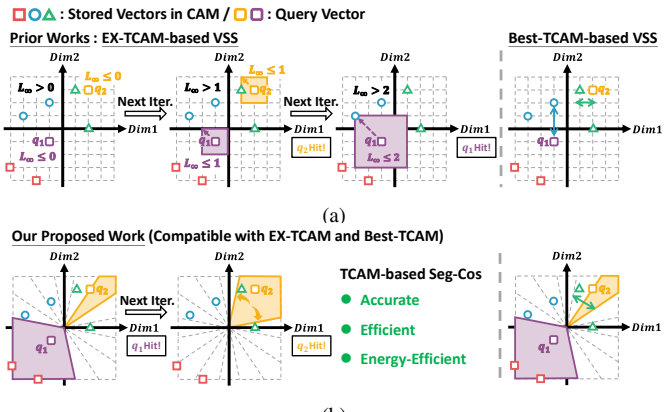


Fig. 1. Comparison between (a) TCAM-based prior works with spatial metrics and (b) the proposed work for angular similarity metrics

While TCAM-based VSS offers advantages, existing frameworks are primarily designed for spatial metrics. In contrast, cosine similarity, an angular metric widely recognized as optimal for software-based VSS [1], [2], achieves superior accuracy compared to TCAM-based VSS [8]–[11], which relies on spatial metrics. To bridge this gap, prior research [10] has explored fine-tuning embedding models for spatial metrics. However, this approach incurs significant training overhead, making them impractical for many real-world applications. Thus, enabling the execution of angular similarity directly within TCAM has become a critical issue. Integrating

angular measurements into TCAM-based VSS presents three key design challenges. First, the underlying metric of TCAM-based VSS is Hamming distance, an additive metric where each dimension contributes independently. In contrast, cosine similarity evaluates the angular relationship through a multiplicative interaction. This difference makes it challenging to support angular similarity with TCAM. Second, cosine similarity is more sensitive to dimensions with larger absolute values, whereas TCAM-based VSS treats all dimensions equally, failing to account for non-uniform effects. Third, existing encoding schemes are designed for spatial metrics, which cannot accommodate the circular distance calculations necessary for angular measurements.

To overcome the above challenges, we propose Seg-Cos, a training-free framework for TCAM-based angular VSS without requiring modifications to existing hardware. Seg-Cos introduces a novel segmented cosine similarity metric that integrates seamlessly into existing TCAM-based VSS. Our approach optimizes quantization, vector representation, and encoding schemes, thereby improving accuracy by 2.2% in FSL, recall by 10% to 52% in ANNS, and energy efficiency by 1.41× in FSL and 2× in ANNS compared with prior TCAM-based VSS supporting L_∞ norm and L_1 norm. The key contributions of this work are as follows:

1) **Developing Segmented Cosine Similarity based on the Re-interpretation of Hamming Distance:** Seg-Cos incorporates two-dimensional cosine similarity into the flow of Hamming distance, making it the first TCAM-based angular VSS framework compatible with both EX-TCAM and Best-TCAM.
2) **Incorporating Characteristics of Cosine Similarity into Range Generation:** Seg-Cos develops magnitude-aware range generation to account for the non-uniform effects in cosine similarity to enhance the functionality and accuracy of TCAM-based Seg-Cos.
3) **Circular Encoding with Range Representation for Angular Distance Measurement:** Seg-Cos enables circular range representation and distance calculation for TCAM-based VSS and avoids the extra codeword overhead induced by the circular structure.

II. Background

A. Vector Similarity Search (VSS)

Vector similarity search (VSS) is a foundational operation in modern AI applications. These applications represent entities—such as words, sentences, images, or videos—as embeddings, i.e., vectors in a learned feature space. For instance, convolutional neural networks (CNNs) transform images into vector embeddings, while sentence transformers encode textual data as vectors. VSS enables the comparison of high-dimensional vectors to identify vectors that are closest to a query, supporting tasks such as few-shot learning (FSL) and approximate nearest neighbor search (ANNS). In FSL, VSS classifies query data by identifying its closest class. In ANNS, VSS efficiently retrieves the approximately nearest

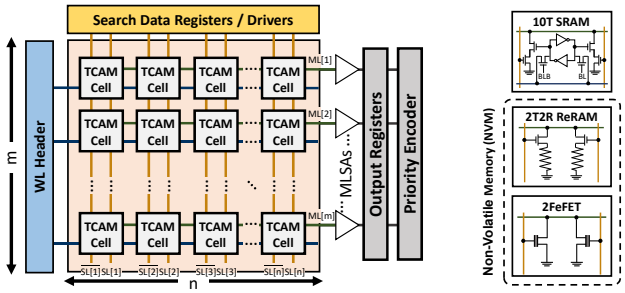


Fig. 2. (a) The TCAM architecture. (b) The configuration of bit cells.

vectors from large-scale databases, which is essential for applications like retrieval-augmented generation (RAG) [4] and recommendation systems. Both cases heavily rely on metrics like cosine similarity $\mathrm{COS}(\vec{q}, \vec{s})$, which measures the cosine of the angle between vectors $\vec{q}$ and $\vec{s}$.

$$\mathrm{COS}(\vec{q}, \vec{s}) = \frac{\vec{q} \cdot \vec{s}}{|\vec{q}|\,|\vec{s}|}. \tag{1}$$

While cosine similarity is a standard metric in VSS, quantifying the angular difference imposes a significant computational burden due to complex floating-point operations. Additionally, the frequent transfer of vectors in von Neumann architectures further degrades energy efficiency. As alternatives, computationally efficient spatial metrics, such as L_∞ and L_1 norms, which measure the maximum or summation of difference across dimensions as described in Eq. (2) and Eq. (3), have been proposed to simplify calculations of metrics.

$$\|\vec{q}, \vec{s}\|_\infty = \max_i |q_{[i]} - s_{[i]}|. \tag{2}$$

$$\|\vec{q}, \vec{s}\|_1 = \sum_i |q_{[i]} - s_{[i]}|. \tag{3}$$

To reduce data transfer overhead, researchers have focused on incorporating these computationally efficient spatial metrics into ternary content-addressable memory (TCAM) combined with different encoding schemes.

B. Ternary Content-Addressable Memory (TCAM)

Ternary content-addressable memory (TCAM) is a specialized memory architecture designed to execute parallel search operations rapidly. Unlike traditional random access memory (RAM), which retrieves data based on specific memory addresses, TCAM compares query data against all stored entries simultaneously, identifying matched memory words in a single clock cycle. Every TCAM cell can represent one of three states: '1', '0', or 'X' ("don't care"), with the "don't care" state capable of matching any input voltage, thereby supporting more flexible search functionalities. The architecture of a standard $n \times m$ TCAM array with n columns and m rows is showcased in Fig. 2(a). Match-lines (MLs) are interconnected among bit cells of n-bit words and linked to match-line sensing amplifiers (MLSAs). Search-lines (SLs) are shared across rows of the TCAM array. The search procedure involves the pre-charging phase for MLs and the assertion of the query data onto SLs. TCAM cells that do not align

with the query data (mismatched cells) will discharge more significantly, causing the voltage levels on the MLs to drop. MLSAs then determine the state of the MLs by evaluating the discharge rate and extent of the voltage drops. Exact-Match TCAM (EX-TCAM) utilizes a pair of inverters [15] as sensing circuits for identifying exact match and mismatch. On the other hand, Best-Match TCAM employs more sophisticated sensing mechanisms, such as analog-to-digital converters (ADC) [11], [13], [14] or winner-take-all (WTA) circuits [12]. These components can read out the distances or detect the columns that exhibit the smallest discharge currents.

The cell structure of TCAM (shown in Fig. 2(b)) can be built upon a variety of devices, including CMOS-based SRAM, as well as emerging non-volatile memories (NVMs) (e.g., resistive RAM (ReRAM) [17], ferroelectric field effect transistor (FeFET) [18] and spin-transfer torque magnetic RAM (STT-MRAM) [19]). Emerging NVM-based TCAMs offer superior density and improved search energy efficiency over the CMOS-based SRAM [20], presenting a promising choice for implementing TCAM to enhance energy efficiency.

C. In-Memory Search (IMS)-based VSS

Several studies [7]–[14] have explored VSS using TCAM, focusing on EX-TCAM-based and Best-TCAM-based VSS. IMS-based VSS generally consists of three main components: pre-processing, encoding scheme, and search process. While the pre-processing flow is typically consistent across IMS-based VSS supporting spatial metrics, the key difference between EX-TCAM-based and Best-TCAM-based VSS lies in their search processes and encoding schemes.

The pre-processing flow in IMS-based VSS consists of two stages: quantization and range Generation. In the quantization stage, vectors are converted from floating-point to fixed-point values, typically using uniform quantization. The range generation stage, introduced by BORE [10], processes these fixed-point values to generate ranges, which enhances the pre-processing step with additional functionality.

For EX-TCAM-based VSS [8], [9], upon receiving a query q, the system expands a hyper-cube centered on q until it encompasses a stored vector, as illustrated in Fig. 1(a). This process identifies the vector with the smallest L_∞ norm distance. It utilizes range encoding (e.g., RENÉ [7]) to encode ranges as ternary codewords for threshold matching. Range encoding preserves the correspondence between overlaps of ranges and match outcomes of codewords. The match outcomes between codewords determine if the stored vectors are inside (exact match) or outside (mismatch) the hyper-cube. A "hit" occurs when any stored vector is within the hyper-cube; otherwise, the search continues with expanded ranges.

On the other hand, Best-TCAM-based VSS preforms a one-pass search process, directly comparing the query with stored vectors without iterative range expansion. Best-TCAM-based VSS supports the Hamming distance [11] without encoding. Furthermore, SAPIENS [13], [14] extend the functionality of Best-TCAM by adopting thermometer encoding to calculate the L_1 norm distance based on the Hamming distance.

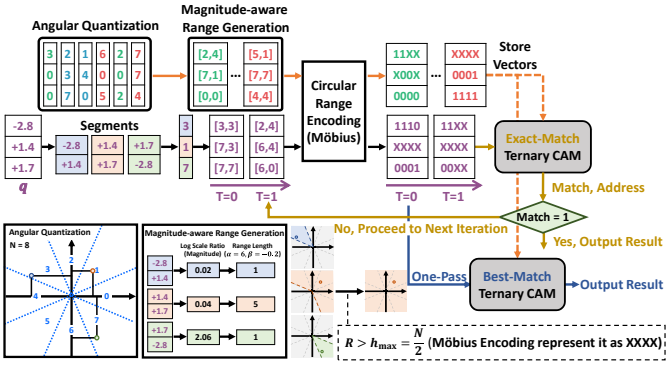
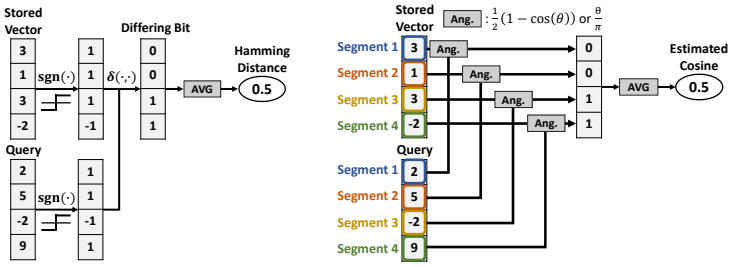


Fig. 3. Processing flow of TCAM-based Seg-Cos



(a) (b)

Fig. 4. (a) Conventional and (b) our perspective to interpret Hamming distance

III. Seg-Cos: Segmented Angular Vector Similarity Search for In-Memory Search

In this paper, we propose segmented cosine similarity, a TCAM-compatible metric based on Hamming distance, to approximate cosine similarity. We also design Seg-Cos, a framework that supports segmented cosine similarity for angular VSS within TCAM-based systems. The processing flow and technical enablers of Seg-Cos are illustrated in Fig. 3. In Section III-A, we define segmented cosine similarity for integrating the angular similarity measurements into TCAM-based VSS. In Section III-B, we develop pre-processing techniques, including angular quantization and magnitude-aware range generation, by leveraging the characteristic of cosine similarity. In Section III-C, we propose a circular encoding that enables circular distance calculations without inducing extra overhead compared to non-circular encoding.

A. Proposed Segmented Cosine Similarity Measurement

To enable efficient angular VSS in TCAM, we design Seg-Cos by reinterpreting the Hamming distance as an angular metric. This insight allow us to simplify the calculation of cosine similarity, make it compatible with TCAM. The Hamming distance is paired with a bipolar sign function $\mathrm{sgn}(\cdot)$ to quantize vectors into 1-bit precision for VSS, as illustrated in Eq. (4a). It measures bit-wise differences and is commonly regarded as a spatial metric, as shown in Fig. 4(a).

$$\mathrm{HAM}(\vec{q},\vec{s}) = \frac{1}{D}\sum_{i=1}^{D}\frac{1}{2}|\mathrm{sgn}(q_i) - \mathrm{sgn}(s_i)| \tag{4a}$$

$$= \frac{1}{D}\sum_{i=1}^{D}\frac{1}{2}(1-\mathrm{COS}(q_i,s_i)) = \frac{1}{D}\sum_{i=1}^{D}\frac{\theta_{q_i,s_i}}{\pi} \tag{4b}$$

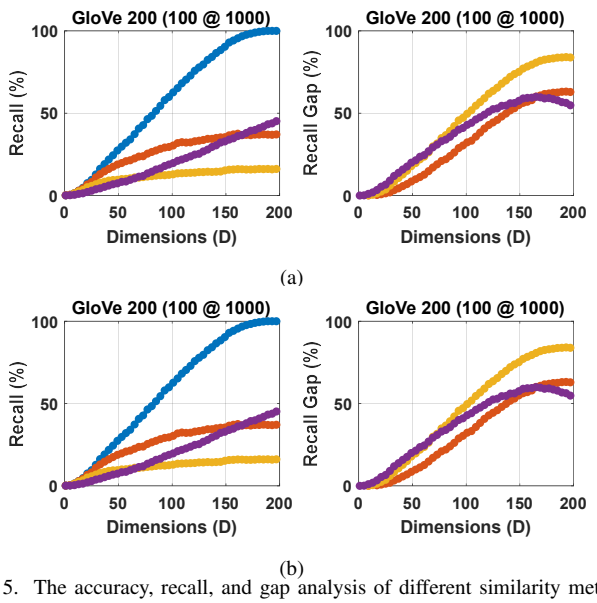


Fig. 5. The accuracy, recall, and gap analysis of different similarity metrics compared with cosine similarity in (a) miniImageNet and (b) GloVe 200

Previous works have explored the encoding and search process for IMS-based VSS to conduct more complex spatial metrics (e.g., L_∞ norm and L_1 norm) based on the Hamming distance measurement. Our motivational experiments, shown in Fig. 5, compare the accuracy and recall of cosine similarity with these spatial metrics across varying dimensions D. As D increases, the gaps between cosine similarity and L_∞ norm, as well as L_1 norm, widen and eventually saturate. In contrast, the gap between cosine similarity and Hamming distance initially grows but later diminishes. This trend indicates that Hamming distance behaves distinctly from other spatial metrics.

To explain this phenomenon, we introduce a novel perspective on interpreting Hamming distance, as depicted in Fig. 4(b). We treat the Hamming distance as a process that divides the D-dimensional vectors $\vec{s} = (s_1, s_2, ..., s_D)$ and $\vec{q} = (q_1, q_2, ..., q_D)$ into D one-dimensional segments, calculate an angular metric, such as cosine distance or angular difference, for each segment pair (s_i, q_i), and then average these values to estimate the angular metrics between the entire vectors, as illustrated in Eq. (4b). Notably, in one-dimensional space, all angular metrics yield the same result and are equivalent up to an affine transformation, as the comparison reduces to a binary distinction: whether the directions of s_i and q_i are the same or opposite. When $D = 1$, the Hamming distance is proportional to the cosine similarity. As D increases, discrepancies between cosine similarity and segment-wise angular metrics grow because each segment remains one-dimensional while D expands. At the same time, the number of segments also increases. As a result, initially, the accuracy gap widens as the growing discrepancy outweighs the benefits of additional segments. However, with further increases in D, the larger number of segments improves the estimation, reducing the gap between Hamming distance and cosine similarity.

Building on this reinterpretation of the Hamming distance, we consider a general case where each vector $\vec{v}$ is divided into D overlapping segments, each with D_{seg} dimensions, denoted as $\vec{v}_{\text{seg},i} = (v_i, v_{(i+1) \bmod D}, ..., v_{(i+D_{\text{seg}}-1) \bmod D})$. We analyze the cosine similarity in Eq. (1) and express it as a weighted synthesis of segment-wise similarities in Eq. (5). This formulation reveals that segment pairs with larger magnitude products $|\vec{q}_{\text{seg},i}||\vec{s}_{\text{seg},i}|$ contribute more significantly to cosine similarity and are more representative of it.

$$\text{COS}(\vec{q}, \vec{s}) = \sum_{i=1}^{D} \frac{|\vec{q}_{\text{seg},i}||\vec{s}_{\text{seg},i}|}{D_{\text{seg}}|\vec{q}||\vec{s}|} \text{COS}(\vec{q}_{\text{seg},i}, \vec{s}_{\text{seg},i}) \tag{5}$$

However, due to the limited operations supported by TCAM, the weighted aggregation involving coupled coefficients from both the query and stored vectors becomes impractical. To address this, we apply two inequalities (Cauchy–Schwarz and Jensen's inequality) to separate the coupling through a logarithmic transformation, as shown in Eq. (6). This decoupling allows for independent pre-processing while preserving the effectiveness of the metric.

$$\begin{aligned} &\ln\left(1 - \text{COS}(\vec{q}, \vec{s})\right) \\ \geq &\ln\left(\sum_{i=1}^{D} \frac{|\vec{q}_{\text{seg},i}||\vec{s}_{\text{seg},i}|}{D_{\text{seg}}|\vec{q}||\vec{s}|}\left(1 - \text{COS}(\vec{q}_{\text{seg},i}, \vec{s}_{\text{seg},i})\right)\right) \\ \geq &\frac{1}{D}\sum_{i=1}^{D} \ln\left(1 - \text{COS}(\vec{q}_{\text{seg},i}, \vec{s}_{\text{seg},i})\right) - \ln\frac{|\vec{q}||\vec{s}|}{|\vec{q}_{\text{seg},i}||\vec{s}_{\text{seg},i}|} \end{aligned} \tag{6}$$

To simplify the angular computation, we apply the first-order Taylor expansion of $\ln(1-\cos\theta)$ around $\theta = \phi$, as shown in Eq. (7). Since classification results are invariant to scaling and shifting of similarity and distance, we replace $\frac{\sin\phi}{1-\cos\phi}$ with $\frac{1}{\alpha}$ and multiply the entire expression with α, as shown in Eq. (8). To further optimize this metric, we introduce β as a parameter that controls clamping effects in Eq. (9), mitigating the error induced by Jensen's inequality. For the EX-TCAM-based Seg-Cos, we select the MAX aggregation function, as shown in Eq. (10a). For the Best-TCAM-based Seg-Cos, we choose the AVG aggregation function, as depicted in Eq. (10b).

$$\ln\left(1 - \cos\theta\right) \approx \frac{\sin\phi}{1 - \cos\phi}(\theta - \phi) + \ln\left(1 - \cos\phi\right) \tag{7}$$

$$\theta_i(\vec{q}, \vec{s}, \alpha, \beta) = \theta_{\text{seg},i} - \alpha \times (\ln\frac{|\vec{q}|}{|\vec{q}_{\text{seg},i}|} + \ln\frac{|\vec{s}|}{|\vec{s}_{\text{seg},i}|} + \beta) \tag{8}$$

$$\max(\theta_i(\vec{q}, \vec{s}, \alpha, 0), \alpha\beta) = \max(\theta_i(\vec{q}, \vec{s}, \alpha, \beta), 0) + \alpha\beta \tag{9}$$

$$\text{Seg-Cos}_{\text{MAX}}(\vec{q}, \vec{s}) = \max_{i:1\leq i\leq D} \max(\theta_i(\vec{q}, \vec{s}, \alpha, \beta), 0) \tag{10a}$$

$$\text{Seg-Cos}_{\text{AVG}}(\vec{q}, \vec{s}) = \frac{1}{D}\sum_{i=1}^{D} \max(\theta_i(\vec{q}, \vec{s}, \alpha, \beta), 0) \tag{10b}$$

The choice of D_{seg} is crucial for the Seg-Cos metric. When $D_{\text{seg}} = 1$ and $\alpha = 0$, the metric reduces to the Hamming distance. When $D_{\text{seg}} = D$ and $\alpha\beta \leq 0$, it corresponds to the angular difference between vectors. Although increasing

D_{seg} allows the metric to capture higher dimensional characteristics and improve accuracy, setting D_{seg} beyond two introduces multiple degrees of freedom (DoF) in angular orientation, significantly increasing the complexity of angular representation. Meanwhile, a two-dimensional space is the smallest space capable of representing the full range of angular differences, offering substantial improvements over a one-dimensional space. Therefore, we select $D_{\text{seg}} = 2$ to preserve a single DoF, ensuring both computational simplicity in pre-processing and significant accuracy gains.

B. Pre-processing of IMS-based Segmented Cosine Similarity

The pre-processing flow of vectors in IMS-based VSS consists of two stages: **Quantization** and **Range Generation**. BORE [10] utilized ranges to skip redundant search iterations in EX-TCAM-based VSS. Seg-Cos extends the role of the **Range Generation** stage by embedding logarithmic magnitude ratios in the ranges. These two stages enable Seg-Cos to integrate segmented cosine similarity into TCAM-based VSS, as defined in Eq. (10a) and Eq. (10b), where angular orientations are subtracted and logarithmic magnitude ratios are summed to produce the final similarity measure.

Quantization. To enable the calculations of two-dimensional angular difference within TCAM, we design a quantization method for representing two-dimensional angular orientation. With only a single DoF in two-dimensional space, we use uniform angular divisions for quantization. Specifically, the two-dimensional plane is divided into N equal sections, each spanning an angle of $\frac{2\pi}{N}$ radians. As illustrated in the quantization stage of Fig. 3, the vector is first segmented, and each segment is projected onto the two-dimensional plane. The orientation of each segment is then mapped to one of these angular sections, indexed from 0 to $N-1$. Consequently, this allows us to calculate the two-dimensional angular difference using the circular distance between quantized values.

Range Generation. The length of range for each segment is generated based on the logarithmic ratio of the full vector magnitude to the segment magnitude, as shown in Eq. (11) and Eq. (12). Since the minimum range length is 1, the half-span is clamped to ensure it remains non-negative. In EX-TCAM-based VSS, the query vector must expand for distance measurement, while the stored vectors remain fixed. If the range encoding cannot accommodate this expansion, we introduce γ as a knob to shift part of the range from the query vector to stored vectors in advance. In Best-TCAM-based VSS, where the roles of the query and stored vectors are functionally identical, γ is set to zero. With Eq. (11) and Eq. (12), segment pairs with larger magnitude products have narrower ranges to maintain precision, while pairs with smaller products have wider ranges to reduce their influence on the final similarity measure.

$$R_{\vec{q}}(\vec{v}_{\text{seq},i}) = 2 \times \max(\lfloor \alpha \times (\ln \frac{|\vec{v}|}{|\vec{v}_{\text{seg},i}|} + \frac{\beta}{2}) \rceil - \gamma, 0) + 1 \quad (11)$$

$$R_{\vec{s}}(\vec{v}_{\text{seq},i}) = 2 \times \max(\lfloor \alpha \times (\ln \frac{|\vec{v}|}{|\vec{v}_{\text{seg},i}|} + \frac{\beta}{2}) \rceil + \gamma, 0) + 1 \quad (12)$$

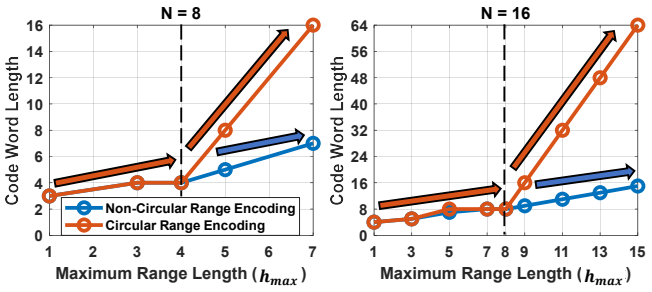


Fig. 6. Codeword length of range encoding solved by CP

C. Encoding for TCAM-based Segmented Cosine Similarity

Previous works have introduced encoding schemes for non-circular distance calculation and range representation, such as RENÉ for value-to-range searches and BORE for range-to-range searches. The codeword length (CL) of them grows linearly with the maximum range length ($h_{\max}$) [7]:

$$CL = \log_2 N + h_{\max} - \log_2 h_{\max} - 1. \quad (13)$$

To develop encoding supporting circular distance calculations for the Seg-Cos framework, we first utilized OR-Tools with constrained programming, extending the approach of BORE to circular range encoding. As shown in Fig. 6, our experiment reveals a key trend: while the codeword length of non-circular range encoding increases linearly with $h_{\max}$, circular range encoding exhibits a two-piece linear growth pattern. The first piece spans $h_{\max}$ from 1 to $\frac{N}{2}$, with CL increasing linearly from $\log_2 N$ (binary encoding) to $\frac{N}{2}$. The second piece extends $h_{\max}$ from $\frac{N}{2}$ to $N-1$, where CL grows linearly from $\frac{N}{2}$ to $\frac{N^2}{4}$ with a steeper slope. Notably, in this second piece, every one-unit increase in $h_{\max}$ results in an extra $\frac{N}{2}$ codewords, which is equivalent to the total codeword length required for the entire first piece. To avoid excessive codeword length overhead for supporting circular range representation, we choose $h_{\max}$ as $N/2$ for the design of circular encoding.

Based on the above analysis, we propose a circular encoding scheme called "Möbius Encoding," designed for $h_{\max} = \frac{N}{2}$. Möbius encoding converts range $[s, e]$ into a ternary codewords $(c_0, c_1, ..., c_{\frac{N}{2}-1})$ as defined in Eq. (14) with $CL = \frac{N}{2}$, where $[i, j]^{\complement}$ denotes the complement range of the range $[i, j]$, expressed as $[(j+1) \bmod N, (i-1) \bmod N]$. The codewords are determined by evaluating the relationship between the range $[s, e]$ and reference half-circle ranges. Specifically, the codeword c_i is determined as follows: $c_i = 0$ if $[s, e]$ lies entirely outside the reference half-circle range, $c_i = 1$ if $[s, e]$ is fully contained within the reference half-circle range, and $c_i = \text{X}$ if range $[s, e]$ overlaps with the reference half-circle range but is not completely contained.

$$c_i = \begin{cases} 0, & \text{if } [s,e] \subset [i+1, (i+\frac{N}{2}) \bmod N]^{\complement} \\ 1, & \text{if } [s,e] \subset [i+1, (i+\frac{N}{2}) \bmod N] \\ \text{X}, & \text{otherwise} \end{cases} \quad (14)$$

TABLE I
SETTING AND STATISTICS OF TCAM SIMULATION.

Platform	Eva-CAM [16]	NeuroSim [27]
TCAM Type	EX-TCAM	Best-TCAM
Device	2FeFET [23]	
Technology	22nm	
On/off Ratio	1.6×10^5	
Area (μm^2)	1698.575	6090.125
Search Latency (ns)	1.069	13.8432
Search Energy (pJ)	1.934	56.715

IV. EXPERIMENTAL RESULTS

A. Settings of Dataset and Simulation

We conducted experiments on FSL tasks to assess the effectiveness of EX-TCAM-based Seg-Cos. Our small-scale experiments utilize Conv4 [2] with 32 dimensions on the Omniglot dataset [17]. For the large-scale experiments, we employed ResNet18 [21] with 128 dimensions on the mini-ImageNet dataset [2]. Our experiments followed the standard 32-way 1-shot Omniglot setting [7], [8]. For the mini-ImageNet, we ventured into a more challenging 8-way 4-shot configuration [22]. For our experiments on ANNS, we aimed to evaluate the effectiveness of the Best-TCAM-based Seg-Cos with GloVe [24] and NYTimes [25]. The GloVe dataset consists of millions of word embeddings, each with dimensions of either 100 or 200. The NYTimes dataset contains 300,000 embeddings, each with dimensions of 256. The key performance metric for these ANNS experiments is the recall rate, specifically "recall 100@1000," which follows the evaluation setting used in [26], and measures the proportion of the top 100 true neighbors in retrieved 1000 candidates returned by VSS.

To estimate the search energy in our experiments, we utilized the Eva-CAM [16] for EX-TCAM and NeuroSim [27] for Best-TCAM with 22nm 2FeFET [23] TCAM models. The TCAM arrays are configured to 128 $\times$ 32 for all scenarios. All simulated statistics are listed in Table I.

B. Energy-Accuracy Trade-off for EX-TCAM VSS in FSL

We compare the effectiveness of EX-TCAM-based Seg-Cos against prior EX-TCAM-based VSS methods [7]–[10] in FSL scenarios, including a low-dimension case (32-way 1-shot omniglot, $D = 32$) and a high-dimension case (8-way 4-shot miniImageNet, $D = 128$). Fig. 7(a) depicts the Pareto fronts, showcasing the energy-accuracy trade-offs with pre-trained controllers for cosine similarity. Each point on the curves corresponds to a different quantization bit-width ranging from 4 bits ($N = 16$) to 6 bits ($N = 64$). Seg-Cos pushes the Pareto front to a better trade-off and maximizes accuracy and energy efficiency in the high-dimension scenario. Among the prior approaches, RENÉ [7]–[9] lacks support for range-to-range searches, making BORE [10] more energy-efficient by reducing iterations and codeword length. While EX-TCAM-based Seg-Cos exhibits similar trends to BORE [10] in the low-dimensional scenario, it excels by 2.2% accuracy gain in high-dimensional settings, where the increasing number of segments provides better estimations of cosine similarity.

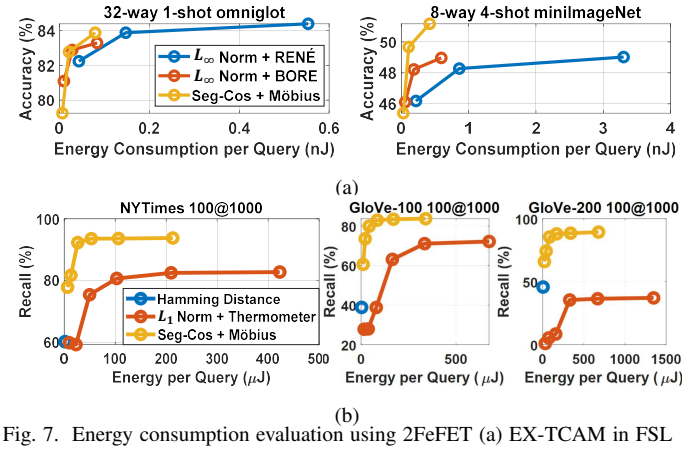


Fig. 7. Energy consumption evaluation using 2FeFET (a) EX-TCAM in FSL and (b) Best-TCAM in ANNS with our proposed Seg-Cos and prior works

C. Energy-Recall Trade-off for Best-TCAM VSS in ANNS

We compare Best-TCAM-based Seg-Cos against prior Best-TCAM-based works with Hamming distance [11] and L_1 norm [13], [14] with thermometer encoding in ANNS scenarios. Fig. 7(b) depicts the Pareto fronts illustrating the energy-recall trade-offs, where cosine similarity serves as the ground truth with a recall rate of 100%. The points on the curves represent the different quantization bits of vectors ranging from 2 bits ($N = 4$) to 7 bits ($N = 512$). Seg-Cos pushes the Pareto front to a better trade-off and enhances the recall rate by 10% to 52% across different datasets compared to Best-TCAM-based L_1 norm implementations [13], [14]. While Seg-Cos extends Hamming distance and consequently consumes more energy, it improves energy efficiency by 2$\times$ over Best-TCAM-based VSS with L_1 norm deu to shorter codeword length in Möbius encoding compared to thermometer encoding.

V. CONCLUSION

In this paper, we propose Seg-Cos, a framework that extends TCAM-based VSS to the angular domain, overcoming the limitations of traditional TCAM-based VSS with spatial metrics. Through the use of angular quantization, magnitude-aware range generation, and Möbius encoding, Seg-Cos bridges the gap between TCAM-based VSS and cosine similarity. Our approach integrates seamlessly with existing TCAM-based systems, enabling improvements in accuracy and energy efficiency without hardware modifications. Experimental results demonstrate that Seg-Cos improves energy efficiency by up to 1.41$\times$ and 2.2% higher accuracy over prior EX-TCAM works in FSL. In ANNS scenarios, Seg-Cos enhances recall by 10% to 52% and energy efficiency by 2$\times$ over prior Best-TCAM works supporting L_1 norm. By addressing the challenges of cosine similarity implementation within TCAM architectures, Seg-Cos represents an advancement in in-memory search.

ACKNOWLEDGMENT

This work was supported by the National Science and Technology Council of Taiwan under Grants of MOST 111-2218-E-002-018-MBK and NSTC 112-2218-E-002-025-MBK. The first author is sponsored by the Novatek Ph.D. Fellowship program.

REFERENCES

[1] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," *Int. Conf. Machine Learning (ICML)*, pp. 1842–1850, 2016.

[2] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, D, Wierstra, "Matching networks for one shot learning," *Advances in Neural Information Processing Syst.*, 2016.

[3] W. Li, *et al*., "Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement," *IEEE Trans. Knowledge and Data Engineering*, vol. 32, no. 8, pp. 1475-1488, 2019.

[4] P. Lewis, *et al*., "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in Neural Information Processing Syst. (NeurIPS)*, pp. 9459-9474, 2020.

[5] A. Ranjan, *et al*., "X-mann: A crossbar based architecture for memory augmented neural networks," *IEEE/ACM Design Autom. Conf. (DAC)*, pp. 1–6, 2019.

[6] H.-W.Hu, *et al*., "ICE: An Intelligent Cognition Engine with 3D NAND-based In-Memory Computing for Vector Similarity Search Acceleration," *IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, pp. 763–783, 2022.

[7] Bremler-Barr, Anat, *et al*., "Encoding Short Ranges in TCAM Without Expansion: Efficient Algorithm and Applications," *IEEE/ACM Trans. Netw. (TN)*, vol. 26, no. 2, pp. 835–850, 2018.

[8] Laguna, Ann Franchesca, Michael Niemier, and X. Sharon Hu, "Design of hardware-friendly memory enhanced neural networks," *IEEE Design, Autom. & Test in Europe Conf. & Exhibition (DATE)*, pp. 1583–1586, 2019.

[9] Laguna, Ann Franchesca, *et al*., "Ferroelectric FET based in-memory computing for few-shot learning," *Great Lakes Symp. VLSI (GLSVLSI)*, pp. 373–378, 2019.

[10] Huang, Chi-Tse, *et al*., "BORE: Energy-Efficient Banded Vector Similarity Search with Optimized Range Encoding for Memory-Augmented Neural Network," *IEEE Design, Autom. & Test in Europe Conf. & Exhibition (DATE)*, 2024.

[11] Karunaratne, Geethan, *et al*., "Robust high-dimensional memory-augmented neural networks.," *Nature communications*, vol. 12, no. 1, pp. 2468, 2021.

[12] Kazemi, Arman, *et al*., "Achieving software-equivalent accuracy for hyperdimensional computing with ferroelectric-based in-memory computing," *Scientific reports*, vol. 12, no. 1, pp. 19201, 2022.

[13] Li, Haitong, *et al*., "One-shot learning with memory-augmented neural networks using a 64-kbit, 118 GOPS/W RRAM-based non-volatile associative memory," *IEEE Symp. VLSI Technology*, pp. 1–2, 2021.

[14] Li, Haitong, et al., "SAPIENS: A 64-kb RRAM-based non-volatile associative memory for one-shot learning and inference at the edge," *IEEE Trans. Electron Devices*, vol. 68, no. 12, pp. 6637–6643, 2021.

[15] Yin, Xunzhao, *et al*., "Ferroelectric ternary content addressable memories for energy-efficient associative search," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 4, pp. 1099–1112, 2022.

[16] Liu, Liu, *et al*., "Eva-cam: a circuit/architecture-level evaluation tool for general content addressable memories," *IEEE Design, Autom. & Test in Europe Conf. & Exhibition (DATE)*, pp. 1173–1176, 2022.

[17] D. Fujiki, *et al*., "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[18] X. Yin, *et al*., "Design and benchmarking of ferroelectric fet based tcam," *IEEE Design, Autom. & Test in Europe Conf. & Exhibition (DATE)*, pp. 1448–1453, 2017 .

[19] W. Kang, *et al*., "In-memory processing paradigm for bitwise logic operations in STT–MRAM," *IEEE Trans. Magnetics (TMAG)*, vol. 53, no. 11, pp. 1–4, 2017.

[20] Li, Shuangchen, *et al*., "Nvsim-cam: a circuit-level simulator for emerging nonvolatile memory based content-addressable memory," *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, vol. 13, no. 1, pp. 1-7, 2016.

[21] Kaiming, He, *et al*., "Deep residual learning for image recognition," *Proceedings of the IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 1685–1694, 2016.

[22] Kazemi, Arman, *et al*., "A flash-based multi-bit content-addressable memory with euclidean squared distance," *IEEE/ACM Int. Symp. Low Power Electronics and Design (ISLPED)*, pp. 1–6,12021.

[23] X. Yin, *et al*., "An Ultra-Dense 2FeFET TCAM Design Based on a Multi-Domain FeFET Model," *IEEE Trans. Circuits and Syst. II (TCAS-II)*, vol. 66, no. 9, pp. 1577–1581, 2019.

[24] Pennington, Jeffrey, Richard Socher, and Christopher D. Manning., "Glove: Global vectors for word representation," *Conf. empirical methods in natural language processing (EMNLP)*, pp. 1532-1543, 2014.

[25] Bache, K. & Lichman, M. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science, 2013

[26] Lee, Yejin, *et al*., "Anna: Specialized architecture for approximate nearest neighbor search." *IEEE Inter. Symp. High-Performance Computer Architecture (HPCA)*, pp. 169-183, 2022.

[27] Peng, Xiaochen, *et al*., "DNN+NeuroSim: An end-to-end benchmarking framework for compute-in-memory accelerators with versatile device technologies," *IEEE Inter. Electron Devices Meeting (IEDM)*, pp. 1532-1543, 2019.

ReSMiPS: A ReRAM-based Sparse Mixed-precision Solver with Fast Matrix Reordering Algorithm

Yuyang Fu[1,†], Jiancong Li[1,†], Jia Chen[2,*], Zhiwei Zhou[1], Houji Zhou[1], Wenlong Peng[1], Yi Li[1,*] *Member, IEEE*, and Xiangshui Miao[1] *Senior Member, IEEE*
[1]School of Integrated Circuits, Huazhong University of Science and Technology
[2]AI Chip Center for Emerging Smart Systems (ACCESS), Hong Kong University of Science and Technology

***Abstract*—The solution of sparse matrix equations is essential in scientific computing. However, traditional solvers on digital computing platforms are limited by memory bottlenecks in large-scale sparse matrix storage and computation. Resistive Random Access Memory (ReRAM)-based computing-in-memory (CIM) offers a promising solution to this challenge but faces constraints in achieving high solution precision and energy efficiency simultaneously in sparse matrix computations. In this work, we propose ReSMiPS, a ReRAM-accelerated sparse mixed-precision solver. ReSMiPS incorporates a novel Fast Sparse Matrix Reordering (FSMR) algorithm and introduces an In-memory float64 (IF64) data format, enabling efficient floating-point sparse matrix computation directly within the analog ReRAM array. By combining our floating-point CIM macro design with a hybrid-domain solution framework, ReSMiPS achieves precision comparable to CPU and GPU-based BiCGSTAB solvers (with errors below 10^{-15}) on real-world workloads, while demonstrating over two orders of magnitude improvement in both computational speed and energy efficiency.**

***Index Terms*—Sparse matrix equations, Mixed-precision Solver, ReRAM, Computing-in-memory**

I. Introduction

Solution of large-scale sparse matrix equations is a primary task in modern scientific modeling technology across various domains such as technology computer-aided design, and electronic design automation [1]. Traditionally, these problems are processed using von Neumann digital computers. These computing platforms now face significant challenges, including the scaling limitations of Moore's law and the power-intensive data movement between separate memory and processing units. Consequently, building high-performance sparse solvers requires substantial hardware resources and energy consumption.

Computing-in-memory (CIM) technology using resistive random access memory (ReRAM) offers promising unconventional computing solutions to address these challenges. By enabling parallelized multiplication and accumulation (MAC) operations directly within the analog domain [2], this technology has demonstrated significant improvements in computational speed and energy efficiency for deep learning and image/signal processing tasks [3], [4].

Despite ReRAM-based CIM has also shown promise in accelerating matrix equation solutions [5], [6], constructing a sparse solver encounters critical challenges due to the following reasons: (i) **Efficient and fast computation of sparse matrix-vector multiplication (SpMV).** The nature of sparse matrices makes it inefficient to directly map an entire sparse matrix onto the ReRAM array (mainly undesirable hardware overhead to store the massive zero elements). While recent SpMV accelerators that integrate content-addressable memory and CIM arrays with sparse storage formats suffer from a significant reduction in computational parallelism [7], [8]. (ii) **Low-Cost Floating-Point (FP) Precision Matrix Storage.** Sparse coefficient matrices for equations are typically stored in 64-bit floating-point (FP64) precision to ensure high-precision solutions. However, directly storing FP64 matrices in ReRAM array using the IEEE-754 binary format incurs substantial hardware overhead [9]. Additionally, implementing complex exponent alignment and fixed-point mantissa MAC operations tightly within the ReRAM array compromises hardware efficiency, especially for FP64 computations [10]. (iii) **Achieving software-comparable solution precision.** Analog computing errors caused by ReRAM fabrication variations and array parasitic effects remain unmitigated, leading to a precision bottleneck in ReRAM-based CIM. Consequently, even though the FP computing scheme was introduced, existing ReRAM-based sparse matrix equation solvers struggle to achieve the precision required for real-world workloads, particularly when compared to digital FP64 solvers [11], [12].

Based on these challenges, we proposed ReSMiPS, a ReRAM-based Sparse Mixed-precision Solver in this work. The main contributions are summarized as follows:

- We introduce a novel Fast Sparse Matrix Reordering (FSMR) algorithm that effectively partitions sparse matrices by clustering non-zero elements into sub-blocks near the diagonal without additional hardware resources. This algorithm enables parallel and efficient SpMV computation using the ReRAM array.
- We propose an In-memory Float64 (IF64) data format for storing FP64 sparse matrices within ReRAM arrays. The IF64 format offers a broad dynamic range at a significantly reduced hardware cost and represents, to our

This article has received funding from the STI 2030—Major Projects (grant no. 2021ZD0201201), National Key Research and Development Plan of China (grant no. 2022YFB4500101), Fundamental Research Funds for Central Universities (grant HUST: 5003190012), and the Hubei Province National Science Foundation (grant no. 2024AFA043).

† These authors contributed equally to this work.

*Correspondence: Yi Li (liyi@hust.edu.cn), Jia Chen (jjiachen@ust.hk)

knowledge, the first data format optimized specifically for CIM-based high-precision sparse solvers.

- We design a ReRAM-accelerated segmented pre-alignment FP-computation framework. This framework incorporates a tiered macro-tile-processing element (PE) IF64-CIM architecture, allowing low-error, high-efficiency FP-SpMV alongside the FSMR algorithm.
- We combine the ReRAM-based lower-precision FP-SpMV unit with a FP64 digital processing unit in ReSMiPS . Enabling solution approximations in ReRAM-array and refines solutions in the digital domain. Along with the proposed optimization approaches. This hybrid system is expected to achieve high solution efficiency, speed, and precision.
- Performance benchmark indicates that ReSMiPS shows significant improvements over CPU and GPU-based BiCGSTAB solver baseline in real-world workloads, achieving residual errors below 10^{-15} within 500 iterations. Additionally, ReSMiPS provides up to 6×10^3 time speedup and over 6×10^2 times energy savings.

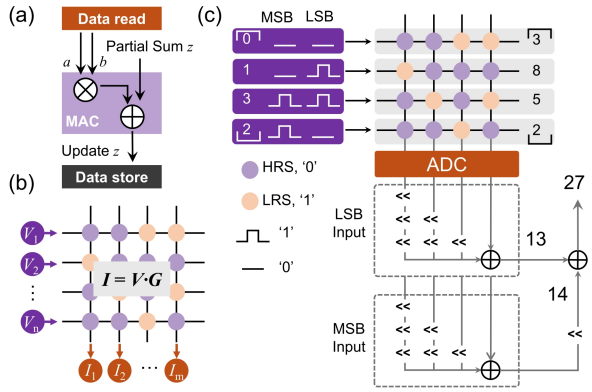


Fig. 1: (a) MAC operation in digiatl computers; (b) ReRAM-based CIM for MAC acceleration; (c) Fixed-point MAC computation in ReRAM accelerator

II. Background

A. ReRAM-based Analog CIM

Fig. 1(a) illustrates the traditional digital MAC workflow, which requires three memory reads and one memory write for each MAC operation. Consequently, the massive MAC operations involved in deep learning and scientific modeling tasks lead to significant memory access bottlenecks. In contrast, ReRAM-based CIM technology, as shown in Fig. 1(b), can accelerate MAC operations within the analog domain. In this computing paradigm, numerical vector or matrix values are directly mapped to ReRAM cells within the array as analog conductance values G. When a forward input vector is applied to the array rows as voltage pulses V, the resulting current I, is collected at the array columns according to Ohm's and Kirchhoff's laws $I = \sum V \cdot G$. Thus, the ReRAM array enables one-step parallel MAC operations in the analog domain, achieving efficient parallel MAC acceleration with $O(1)$ time complexity.

B. Fixed-Point Processing in ReRAM Accelerator

The processing of SpMV on a ReRAM array with FP precision builds upon fixed-point MAC operations within the ReRAM. Fig. 1(c) illustrates an example of ReRAM-accelerated fixed-point MAC. In this example, a 4-bit vector is converted into binary bits and mapped to multiple single-level cells (SLCs). The 2-bit decimal input is similarly bit-sliced and sequentially encoded as binary voltage pulses. Consequently, the MAC operation is performed in a pipelined manner, with the least significant bit (LSB) of the vector applied to the ReRAM array first, followed by the most significant bit (MSB). Bit-wise MAC operations are computed within the ReRAM array, with the final high-precision results obtained through a peripheral shift-and-add unit [13]. Notably, this method can be adapted to arbitrary fixed-point precision by extending the hardware configuration. Additionally, more efficient computation can be achieved by mapping binary bits to both SLCs and multi-level cells (MLCs) [14].

III. ReSMiPS Architecture

Fig. 2 illustrates the overall architecture of ReSMiPS, which consists of a digital processor and a ReRAM-based FP-CIM processor. The CIM processor includes multiple processing tiles, on-chip memory, and an embedded processor to support large-scale FP-SpMV computations and approximate solution generation. The digital processor comprises a system controller, system memory, and several digital functional units, which handle sparse matrix reordering and mapping through the FSMR block and provide high-precision solutions within the iterative refinement block. Detailed descriptions of the FSMR algorithm, the internal design of the CIM processor, and the solution framework are provided below.

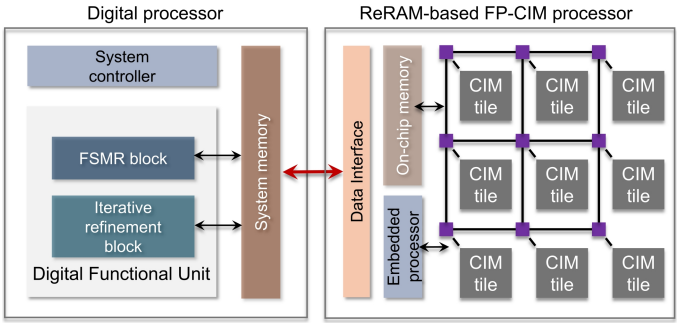


Fig. 2: Architecture overview of the ReSMiPS

A. FSMR Algorithm

The FSMR algorithm is an extension of the Cuthill-McKee (CM) algorithm, which is traditionally used to reduce matrix bandwidth by clustering non-zero elements along the diagonal [15]. While the reordered matrices produced by the CM algorithm exhibit optimal symmetry and bandwidth properties, which are advantageous for CIM architectures, the row-column permutations required result in asymmetric matrices. This asymmetry introduces complications during matrix-vector MAC operations, requiring complex transformations for input

and output vectors. It also limits the flexibility of left/right matrix multiplications, increasing energy consumption and area costs. To address these limitations, we propose a symmetric transformation matrix T_{sym} as outlined in Algorithm 1.

Herein, we apply the CM algorithm to a matrix A that produces a reordered matrix B alongside asymmetric row and column permutation matrices T_{row} and T_{col}. The FSMR algorithm refines this result by locally permuting B to derive symmetric permutation matrices T_{sym}, as shown in Fig.3. Specifically, the segmentation process begins with a traversal of T_{row}. During each iteration, δ is calculated as the root-2 times the differential between the current traversal flag position and the terminal element in chunk flag array α. If δ exceeds or equals the bandwidth value at coordinates (row, row) in B, a validation procedure is triggered. If the non-zero element counts in the new chunks match their symmetric counterparts, the row is appended to α.

ALGORITH 1. FAST SPARSE MATRIX REORDERING

Input: rectangular, symmetrical matrix ***A***

Output: matrix ***A'*** (***A*** after reordering), symmetrical transformation matrix $\boldsymbol{T_{sym}}$

Construct matrix ***B*** and row transformation matrix $\boldsymbol{T_{row}}$ from ***A*** as in Cuthill-McKee

Set list $\boldsymbol{\alpha}$ as the block boundaries of the block matrixes

while iteration flag *row* < length of $\boldsymbol{T_{row}}$

  set *bandwidth* be equal to the bandwidth at (*row*, *row*) in ***B***

  if (*row* – $\boldsymbol{\alpha}$[-1]) × $\sqrt{2} \geq$ *bandwidth* and the number of non-zero elements in the symmetrical new divided blocks are equal: append *row* to the end of $\boldsymbol{\alpha}$

endwhile

while iteration flag *block* < number of divided block matrixes

  if block is on the diagonal: rearrange all non-zero elements in the block to the diagonal

  else if block is not on the diagonal: Set the row indices of the non-zero elements to the column indices of the corresponding non-zero elements in its symmetric block

endwhile

Combine all reordered blocks together to obtain $\boldsymbol{T_{sym}}$

Obtain ***A'*** by the equation $\boldsymbol{A' = T_{sym} \cdot B \cdot T_{sym}}$

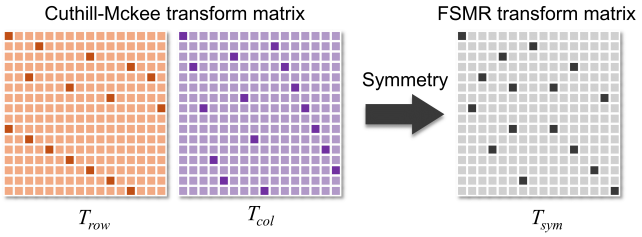


Fig. 3: Obtaining the FSMR transformation matrix

After fully traversing T_{row}, α defines the defines the partitioning scheme for B. Iterative traversal of the resulting submatrices redistributes non-zero elements to enforce symmetry. For diagonal blocks, non-zero elements are aligned along the principal diagonal. For off-diagonal blocks, row indices of non-zero elements are mapped to align with the column indices of their symmetric counterparts. Finally, concatenating these reordered sub-blocks produces the symmetric transformation matrix T_{sym}. Using this transformation, B is converted into A' through the transformation $A' = T_{sym} \cdot B \cdot T_{sym}$.

Fig. 4 illustrates the application of the FSMR algorithm to a large-scale sparse matrix. Compared to the previously reported General matrix reordering algorithm for ReRAM-CIM, the FSMR algorithm demonstrates superior efficiency in reorganizing sparse matrices into dense non-zero sub-blocks alongside the matrix diagonal [15]. This enhanced reordering optimally leverages the parallel processing capabilities of the ReRAM array, significantly accelerating SpMV computations.

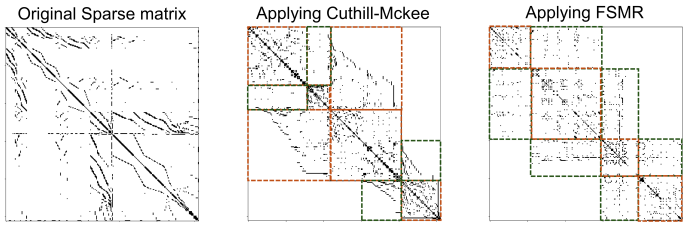


Fig. 4: Comparison of FSMR with Cuthill-Mckee algorithm

B. IF64 Data Format and ReRAM-Accelerated Computation

In conventional digital sparse solvers, the IEEE-754 FP64 data format is widely used to represent sparse matrices due to its extensive dynamic range and high numerical precision, enabling high-accuracy solutions. However, directly mapping FP64 matrix data onto a ReRAM array through pre-alignment is impractical [16]. The large exponent range and wide mantissa bit width of FP64 data introduce substantial hardware overhead for pre-alignment mapping. Furthermore, physical noise in the analog ReRAM array restricts the effective precision of FP64 data representation, resulting in a precision bottleneck that does not meet hardware computational requirements, even with increased bit width. Conversely, representing matrix data in FP32 format fails to support matrices with large dynamic ranges, limiting the ReSMiPS system's ability to achieve the desired solution precision.

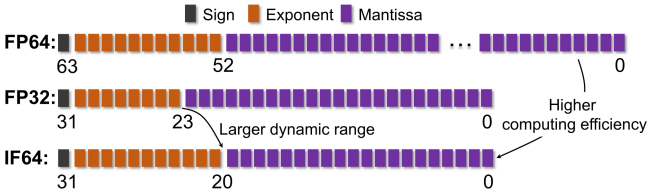


Fig. 5: Comparison of the IF64 data with FP64 and FP32 data

In view of this limitation, we defined IF64, a new FP data format specifically designed for the mixed-precision CIM solver. An IF64 data element consists of a sign bit, an 11-bit exponent, and a 20-bit mantissa. Compared to standard FP64 and FP32 formats, the IF64 format maintains a large exponent dynamic range similar to FP64 while supporting more efficient data mapping due to its reduced mantissa length, as shown in Fig.5.The value of data in the IF64 format is calculated in the same manner as standard IEEE-754 floating-point data, using

$$V_{IF64} = (-1)^S \times 2^{E-Bias} \times 1 \cdot M \tag{1}$$

Where the exponent bias is consistent with the FP64 data format, thereby, the MAC with IF64 format is operated using the same arithmetic as the IEEE-754 standard, i.e., involving mantissa computation and exponent alignment.

Accordingly, Fig. 6 illustrates the proposed IF64 MAC computation within the ReRAM array, which involves four

main processes: (i) Pre-alignment and mantissa truncation. This process decouples exponent alignment from mantissa computation by identifying the maximum exponent (E_{max}) across a set of IF64 matrix or vector elements and shifting or padding each element's mantissa (including the hidden leading 1). Truncation is then applied as an approximation technique to maintain accuracy while reducing storage and computation overhead. (ii) Complementary code conversion. Here, the pre-aligned and truncated mantissa are converted to complementary code, which includes the number's sign bit. Using complementary representation eliminates the need for differential pairs in mapping data sign information to the ReRAM array, thereby reducing hardware overhead. (iii) SLC/MLC hybrid mapping. The binary complementary matrix is split, with the sign and MSBs mapped to SLC arrays for processing efficiency, while the least LSBs are mapped to MLC arrays to enhance processing capacity. (iv) Fixed-point CIM. This final process executes fixed-point MAC operations on the pre-processed vector and matrix data within the ReRAM accelerator. The resulting IF64 MAC values are then obtained by scaling the fixed-point MAC results with the exponent value.

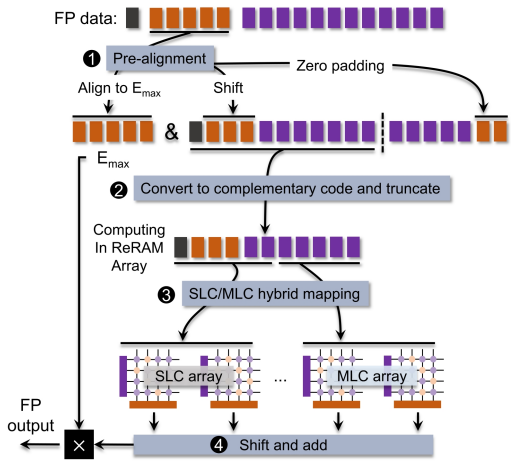


Fig. 6: FP-MAC computation within ReRAM-accelerator

It is important to note that the FSMR algorithm is specifically designed to partition large-scale sparse matrices efficiently into dense non-zero sub-blocks. Consequently, we perform locality pre-alignment on the sliced non-zero sub-blocks rather than applying global alignment to the entire reordered sparse matrix (Fig. 7). Locality pre-alignment effectively mitigates the issue of small data flooding caused by excessively large data dynamic ranges, thereby improving computational accuracy, particularly for large-scale SpMV calculations [17].

C. Internal Designation of the FP-CIM Processor

Our FP-CIM processor is specifically designed to support FSMR-motivated SpMV computations with IF64 precision. To achieve this, the processor is organized into three hierarchical levels, ranging from macro-computing tiles to PEs. Fig. 8 illustrates the internal architecture of the computing tile and

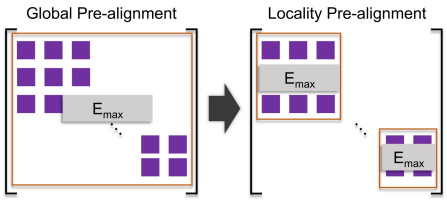


Fig. 7: Comparison of the global pre-alignment with locality pre-alignment for SpMV computation

PE. Each tile comprises 16 PEs for FP-MAC operations, a tile buffer for storing input data and MAC results, and a tile controller. The interconnection among PEs is structured as an H-tree to minimize data transfer delays and interconnection overhead [17].

Additionally, we suppose each PE includes sixty-four 128×128 ReRAM MAC cores, a PE input/output data register, a pre-alignment block, a core input block, a core output register, a shift-and-add (SAA) block, an exponent register, an exponent adder, and a PE controller. For SpMV computations, the FSMR-reordered sparse matrix is mapped to the CIM processor using the locality pre-alignment method. Input vectors are distributed to each PE, pre-aligned with the local maximum exponent, and truncated within the core input block. Fixed-point mantissa computations are executed in the ReRAM MAC cores. The partial products are stored in the core output register and subsequently processed by the SAA block. Finally, the mantissa MAC results are multiplied by the exponents in the exponent register and adder block. The final IF64 MAC results are then transferred to the PE data register.

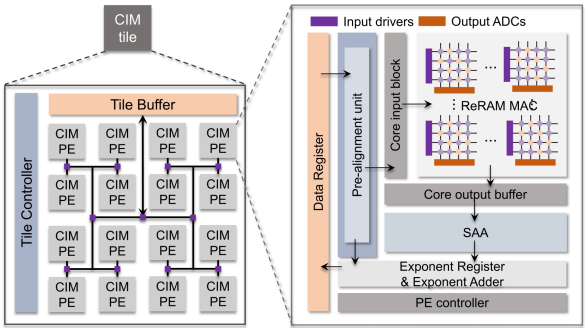


Fig. 8: Internal designation of the CIM-tile and PE

Fig. 9 (a) illustrates the detailed design of the pre-alignment unit. For a given set of input vectors, the maximum exponent, E_{max}, is first determined using a comparator tree. The difference between E_{max} and the exponent E is then computed using a subtractor. Subsequently, the mantissa is aligned by right-shifting according to the calculated exponent difference. The pre-aligned mantissa is truncated while being transferred to the core input block. Meanwhile, Fig.9 (b) demonstrates an example of the fixed-point MAC workflow, detailing the process from the core input block to the ReRAM crossbars. Initially, load balancing is performed within the input block to eliminate all-zero sequences introduced by pre-alignment processing, thereby reducing overall computational delay. When signals are driven into the array, a high voltage

level is applied to array rows for an input value of 1, while no signal is applied for an input value of 0. This sparsified input method is beneficial to the system's robustness to device randomness and improves computational accuracy [14].

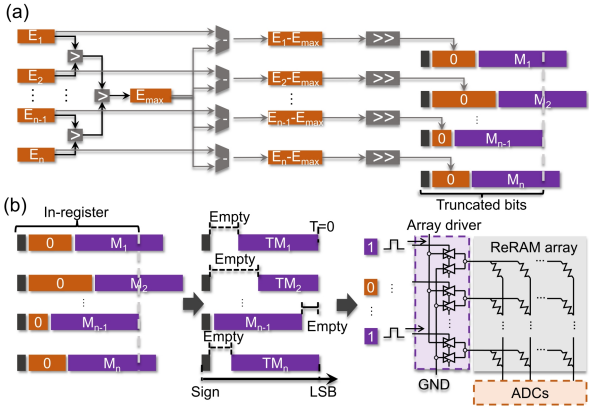


Fig. 9: (a) The pre-alignment block; (b) The fixed-point MAC workflow

Fig. 10(a) presents the detailed pipeline of the Shift-and-Add (SAA) block, which facilitates hybrid mapping using both SLC and MLC arrays while delivering fixed-point MAC results. Initially, the outputs from the ReRAM cores are stored in the output buffer. Subsequently, the buffer outputs (*Sum*) are fed into the 1st SAA module, which performs input-wise SAA operations. This involves binary input signals being sequentially shift-added. In the first cycle, the 1st SAA executes an inverse operation on the sums, followed by a one-bit left shift and an accumulation operation, as illustrated in Fig. 10(b). After completing one cycle of processing in the 1st SAA, its outputs are sequentially passed to the 2nd SAA module, which functions as the weight-wise shift-and-add unit. The structure of the 2nd SAA is shown in Fig. 10(c). Similar to the 1st SAA, the second SAA performs additional shift operations to accommodate different SLC and MLC configurations. Specifically, outputs from the SLC crossbar are shifted one bit to the left, while outputs from the MLC crossbar are shifted two bits to the left.

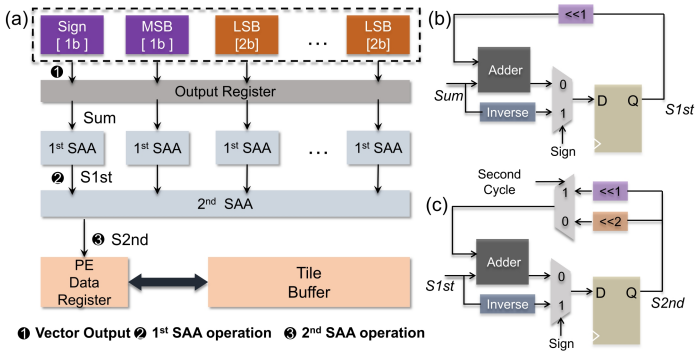


Fig. 10: (a) Pipeline of the SAA block for fixed-point output ; (b) Design of the 1st SAA; (c) Design of the 2nd SAA

D. Solution Framework

Fig. 11 illustrates the mixed-precision solution framework within the ReSMiPS system. For a given sparse matrix equation $Ax = b$, the coefficient sparse matrix A is first reordered using the FSMR algorithm, after which it is partitioned into sub-blocks and mapped to ReRAM arrays using the locality pre-alignment method. The reordered matrix A^F and the target vector b^f are processed for solution iteration. Specifically, the ReRAM-based CIM processor solves the error correction equation $A^F \cdot d = r$ using the BiCGSTAB algorithm in the IF64 domain (We set the iteration error of the BiCGSTAB algorithm to 10^{-3}) [18]. The high-precision solution is approximated by performing $x^f_{n+1} = x^f_n + d$ in the FP64 domain. Residual error is computed as $r = b^f - A^F \cdot x^F_n$ with FP64 precision. The iteration terminates when the residual norm falls below the preset tolerance of 10^{-15}. Finally, the solution x is obtained by reordering the vector x^f. Compared to existing mixed-precision solvers, ReSMiPS achieves high-efficiency SpMV through FSMR and enables efficient approximate solutions via IF64 computation. It further mitigates the effects of device variation while delivering FP64-equivalent computational accuracy.

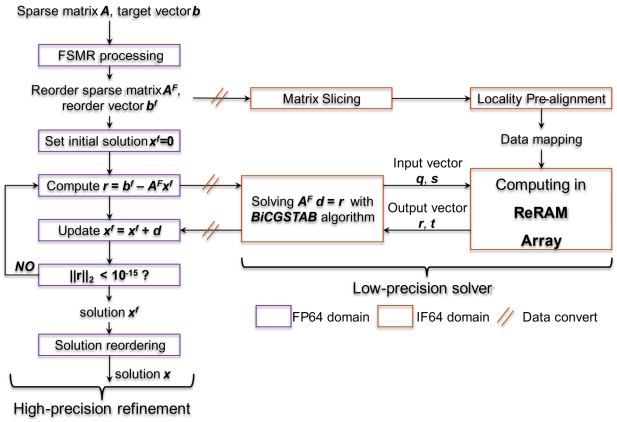


Fig. 11: Solution framework within the ReSMiPS system

IV. Evaluation

Based on our ReSMiPS design, we implemented the high-precision solution of large-scale sparse matrix equations and further evaluated the system's solution speed and energy consumption.

A. Evaluation Setup

We evaluate 12 real-world sparse matrices from the SuiteSparse Matrix Collection [19], which come from different fields such as computational fluid dynamics problems, power network problems, etc. Table 1 provides the name, size, non-zeros (NNZ), and the condition number (K) for each matrix. The row/column numbers of the matrices span from 292 to 1961, and the matrices condition number ranges from 2.1×10^{01} to larger than 9.2×10^{18}. Graphical visualization of each matrix is presented in Fig.12. It should be mentioned that the right-hand vector b is set to $b \equiv 1$ for each matrix.

We present the convergence traces (the residuals across each iteration) of our ReSMiPS system and CPU-based BiCGSTAB solver in Fig. 13. The specific configurations for each platform are provided in Table 2. Both solvers are set to a maximum iteration count of 500. The ReSMiPS system achieves high

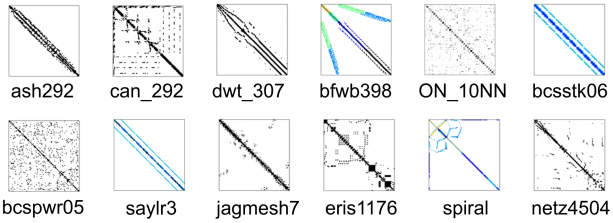


Fig. 12: Visualization of the Sparse matrices for benchmark

TABLE I: Sparse matrices for benchark

Name	Size	NNZ	K	Name	Size	NNZ	K
ash292	292	2208	1.1×10^{18}	bcspwr056	443	1623	7.5×10^{17}
can_292	292	2540	1.1×10^{18}	saylr3	1000	3750	inf
dwt_307	307	2523	3.6×10^{18}	jagmesh7	1138	7450	1.2×10^{04}
bfwb398	398	2910	2.1×10^{01}	eris1176	1176	18552	inf
ON_10NN	400	2879	1.0×10^{04}	spairl	1434	18228	3.9×10^{05}
bcsstk06	420	7860	7.5×10^{06}	netz4504	1961	5156	9.2×10^{18}

solution precision, with residuals consistently around 10^{-15} for all workloads. In contrast, the CPU solver fails to converge for certain ill-conditioned problems. These results demonstrate the effectiveness of the ReSMiPS system in solving large-scale sparse matrix equations. It is worth noting that we observed minor perturbations in the convergence curves of the ReSMiPS system, which are supposed to be caused by the resistive variation of the ReRAM. Nevertheless, the ReSMiPS maintains achieving high-precision solutions, underscoring its robustness.

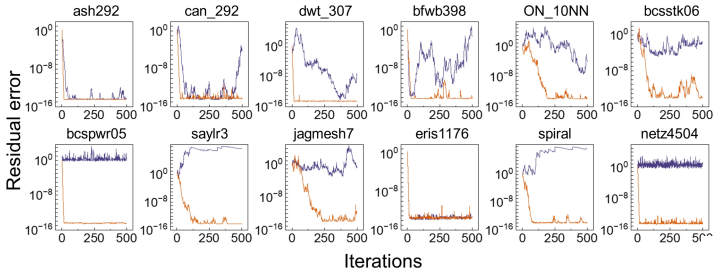


Fig. 13: Convergence traces of ReSMiPS and the CPU-based BiCGSTAB solver. Orange: ResMiPS; Purple: CPU

B. Speed and Energy Evaluation

TABLE II: Platform configuration

GPU (NIVIDA RTX 4070 Ti SUPER)			
Architecture	Ada Lovelace	CUDA Cores	448
Memory	16 GB GDDR6X	CUDA Version	11.6
CPU (Intel i7-14700)			
Architecture	Raptor Lake-R	Cores	8 P-core
Memory(L1/L2/L3)	1.8/28.0/33.0 MB	Frequency	Max 5.3 GHz
ReSMiPS (@22 nm CMOS technology)			
Tile	9	Crossbar Size	128 × 128
PE/Tile	16	Precision	IF64
Crossbars/PE	64	Comp.ReRAM	144 Mb
ADC: 0.35 mW 12-bit Pipelined SAR ADC @ 100 MS/s			
ReRAM cells: 2-bit Cell, Comp.Latency = 10 ns@(128 × 128)			

Fig. 14 compares the performance, in terms of speed, of a CPU-based and GPU-based solver with our ReSMiPS system. The time cost is measured by performing SpMV computations with different platforms. These results indicate that the ReSMiPS system achieves a significant speedup, outperforming the CPU and GPU implementations by factors of approximately $\sim 7.5 \times 10^3$ and $\sim 6 \times 10^2$, respectively. Notably, the exponential relationship between matrix dimensions and computational speedup highlights that the advantages of the ReSMiPS system become increasingly pronounced as matrix size grows. Furthermore, the FSMR algorithm enables an average $5\times$ speedup compared to the ReSMiPS system without FSMR, with up to $38\times$ speedup observed in $netz4501$ test case. These results demonstrate that the FSMR algorithm not only enhances array utilization but also contributes to the speedup of the SpMV computation.

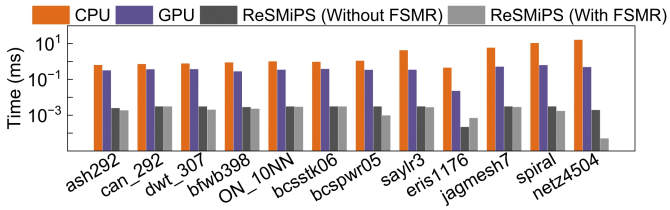


Fig. 14: Speed comparison of CPU, GPU, and our ReSMiPS system

In the meantime, Fig. 15 illustrates the energy consumption of 12 matrices from SuiteSparse across different computing platforms, including CPU, GPU, and our ReSMiPS system. The results reveal that the ReSMiPS system achieves energy savings of approximately 6.1×10^4 over the CPU and nearly 2.1×10^3 over the GPU. Additionally, the FSMR algorithm contributes to approximately $4.4\times$ energy savings in SpMV computations, with a remarkable $38\times$ energy savings observed in the $netz4501$ test case.

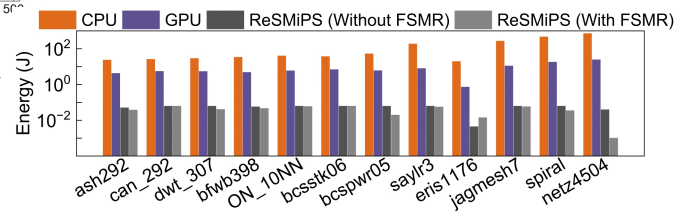


Fig. 15: Energy consumption comparison of CPU, GPU, and our ReSMiPS system

CONCLUSION

In summary, we present the ReSMiPS system, a ReRAM-based hybrid platform that integrates low-precision CIM processing with high-precision digital refinement to solve large-scale sparse matrix equations. Leveraging the proposed FSMR algorithm, IF64 data format, and a floating-point CIM architecture for accelerated SpMV computation, ReSMiPS achieves solution precision with residual errors below 10^{-15} while delivering remarkable speedups ($10^2 \sim 10^3\times$) and energy savings ($10^3 \sim 10^4\times$) compared to traditional CPU and GPU solvers. These results demonstrate that ReSMiPS sets a new benchmark for high-precision and efficient sparse matrix equation solvers using analog ReRAM technology.

REFERENCES

[1] W. M. Gentleman and A. George, *SPARSE MATRIX SOFTWARE*. Elsevier, 1976, pp. 243–261.

[2] M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, and J. P. Strachan, "Memristor-based analog computation and neural network classification with a dot product engine," *Advanced Materials*, vol. 30, no. 9, Jan. 2018.

[3] W. Zhang, B. Gao, J. Tang, P. Yao, S. Yu, M.-F. Chang, H.-J. Yoo, H. Qian, and H. Wu, "Neuro-inspired computing chips," *Nature Electronics*, vol. 3, no. 7, p. 371–382, Jul. 2020.

[4] C. Li, M. Hu, Y. Li, H. Jiang, N. Ge, E. Montgomery, J. Zhang, W. Song, N. Dávila, C. E. Graves, Z. Li, J. P. Strachan, P. Lin, Z. Wang, M. Barnell, Q. Wu, R. S. Williams, J. J. Yang, and Q. Xia, "Analogue signal and image processing with large memristor crossbars," *Nature Electronics*, vol. 1, no. 1, p. 52–59, Dec. 2017.

[5] M. A. Zidan, Y. Jeong, J. Lee, B. Chen, S. Huang, M. J. Kushner, and W. D. Lu, "A general memristor-based partial differential equation solver," *Nature Electronics*, vol. 1, no. 7, p. 411–420, Jul. 2018.

[6] M. Le Gallo, A. Sebastian, R. Mathis, M. Manica, H. Giefers, T. Tuma, C. Bekas, A. Curioni, and E. Eleftheriou, "Mixed-precision in-memory computing," *Nature Electronics*, vol. 1, no. 4, p. 246–253, Apr. 2018.

[7] X. Zhang, Z. Li, R. Liu, X. Chen, and Y. Han, "Fspa: An fefet-based sparse matrix-dense vector multiplication accelerator," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, Jul. 2023, p. 1–6.

[8] M. Fan, X. Chen, D. Yang, Z. Jin, and W. Liu, "Recg: Reram-accelerated sparse conjugate gradient," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, ser. DAC '24. ACM, Jun. 2024, p. 1–6.

[9] L. Song, F. Chen, H. Li, and Y. Chen, "Refloat: Low-cost floating-point processing in reram for accelerating iterative linear solvers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '23. ACM, Nov. 2023, p. 1–15.

[10] F. Tu, Y. Wang, Z. Wu, L. Liang, Y. Ding, B. Kim, L. Liu, S. Wei, Y. Xie, and S. Yin, "Redcim: Reconfigurable digital computing- in -memory processor with unified fp/int pipeline for cloud ai acceleration," *IEEE Journal of Solid-State Circuits*, vol. 58, no. 1, p. 243–255, Jan. 2023.

[11] J. Li, S.-g. Ren, Y. Li, L. Yang, Y. Yu, R. Ni, H. Zhou, H. Bao, Y. He, J. Chen, H. Jia, and X. Miao, "Sparse matrix multiplication in a record-low power self-rectifying memristor array for scientific computing," *Science Advances*, vol. 9, no. 25, Jun. 2023.

[12] W. Song, M. Rao, Y. Li, C. Li, Y. Zhuo, F. Cai, M. Wu, W. Yin, Z. Li, Q. Wei, S. Lee, H. Zhu, L. Gong, M. Barnell, Q. Wu, P. A. Beerel, M. S.-W. Chen, N. Ge, M. Hu, Q. Xia, and J. J. Yang, "Programming memristor arrays with arbitrarily high precision for analog computing," *Science*, vol. 383, no. 6685, p. 903–910, Feb. 2024.

[13] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, vol. 15, no. 7, p. 529–544, Mar. 2020.

[14] W.-H. Huang, T.-H. Wen, J.-M. Hung, W.-S. Khwa, Y.-C. Lo, C.-J. Jhang, H.-H. Hsu, Y.-H. Chin, Y.-C. Chen, C.-C. Lo, R.-S. Liu, K.-T. Tang, C.-C. Hsieh, Y.-D. Chih, T.-Y. Chang, and M.-F. Chang, "A nonvolatile al-edge processor with 4mb slc-mlc hybrid-mode reram compute-in-memory macro and 51.4-251tops/w," in *2023 IEEE International Solid- State Circuits Conference (ISSCC)*. IEEE, Feb. 2023.

[15] J. Cui and Q. Qiu, "Towards memristor based accelerator for sparse matrix vector multiplication," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, May 2016, p. 121–124.

[16] T.-H. Wen, H.-H. Hsu, W.-S. Khwa, W.-H. Huang, Z.-E. Ke, Y.-H. Chin, H.-J. Wen, Y.-C. Chang, W.-T. Hsu, C.-C. Lo, R.-S. Liu, C.-C. Hsieh, K.-T. Tang, S.-H. Teng, C.-C. Chou, Y.-D. Chih, T.-Y. J. Chang, and M.-F. Chang, "34.8 a 22nm 16mb floating-point reram compute-in-memory macro with 31.2tflops/w for ai edge devices," in *2024 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, Feb. 2024, p. 580–582.

[17] Z. Zhou, J. Li, H. Jia, L. Yang, H. Zhou, H. Bao, Y. Fu, Y. Li, and X. Miao, "Arpcim: An arbitrary-precision analog computing-in-memory accelerator with unified int/fp arithmetic," *IEEE Transactions on Circuits and Systems I: Regular Papers*, p. 1–14, 2024.

[18] G. L. G. Sleijpen, H. A. van der Vorst, and D. R. Fokkema, "Bicgstab(l) and other hybrid bi-cg methods," *Numerical Algorithms*, vol. 7, no. 1, p. 75–109, Mar. 1994.

[19] T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," *ACM Transactions on Mathematical Software*, vol. 38, no. 1, p. 1–25, Nov. 2011.

Approximate SMT Counting Beyond Discrete Domains

Arijit Shaw
Chennai Mathematical Institute, India
IAI, TCG CREST, Kolkata, India

Kuldeep S. Meel
Georgia Institute of Technology, USA
University of Toronto, Canada

***Abstract*—Satisfiability Modulo Theory (SMT) solvers have advanced automated reasoning, solving complex formulas across discrete and continuous domains. Recent progress in propositional model counting motivates extending SMT capabilities toward model counting, especially for hybrid SMT formulas. Existing approaches, like bit-blasting, are limited to discrete variables, highlighting the challenge of counting solutions projected onto the discrete domain in hybrid formulas.**

We introduce pact, an SMT model counter for hybrid formulas that uses hashing-based approximate model counting to estimate solutions with theoretical guarantees. pact makes a logarithmic number of SMT solver calls relative to the projection variables, leveraging optimized hash functions. pact achieves significant performance improvements over baselines on a large suite of benchmarks. In particular, out of 14,202 instances, pact successfully finished on 603 instances, while Baseline could only finish on 13 instances.

I. Introduction

Propositional model counting is the task of counting the number of satisfying assignments for a given Boolean formula. Recent advances in model counters have made them useful for solving a variety of real-world problems [1], [2] such as probabilistic inference [3], software verification [4], network reliability [5], and neural network verification [6]. The development of model counters has been motivated by the success of SAT solvers over the past few decades, which allowed researchers to explore problems beyond mere satisfiability.

Concurrently, the success of SAT solvers [7] led to an interest in solving the satisfiability of formulas where the variables are not just Boolean. This interest gave rise to the field of Satisfiability Modulo Theories (SMT), which includes a range of theories such as arithmetic, bitvectors, and data structures [8]. SMT theories, inspired by application needs, offer more succinct problem representations than Boolean satisfiability. There has been a significant development in the design of SMT solvers in recent years [9]–[13]. The compactness of SMT and recent solver advancements have made them useful in software and hardware verification [14], [15], security [16], test-case generation, synthesis, planning [17], and optimization [18].

In light of the availability of powerful SMT solvers, a natural next challenge is to explore the problem of model counting for SMT formulas. Despite demonstrated applications [4], [19], SMT counting remains underexplored. Most prior work has concentrated on problems where the underlying theory is discrete, such as bitvectors [20]–[22], linear integers [23]–[25], and strings [26]. Recent studies have shown that in these cases, reducing the problem to Boolean model counting is often the most effective approach [27]. On the other hand, SMT solvers are widely employed for applications that require reasoning over both continuous and discrete variables. For example, to encode hybrid systems in cyber-physical systems or planning, it is essential to have both variables. The existing work in SMT counting is unable to handle SMT formulas with continuous variables.

In this work, we seek to remedy the aforementioned situation. In particular, we focus on a large class of SMT formulas, which we refer to as *hybrid SMT formulas*. The hybrid SMT formulas are defined over both discrete and continuous variables, and we are interested in solutions projected over discrete variables. Our investigations for the development of counting techniques for hybrid SMT formulas are motivated by their ability to model several interesting and relevant applications, such as robustness quantification of cyber-physical systems and counting reachable paths in software (for detailed discussion, see Section I-A).

The primary contribution of this work is the development of a model counting tool, pact[1], for efficient projected counting of hybrid SMT formulas. The framework approximates the model count with (ε, δ) guarantees. pact supports SMT formulas with various theories, including linear and non-linear real numbers, floating-point arithmetic, arrays, bit-vectors, or any combination thereof. The projection variables are over bit-vectors. pact employs a hashing-based approximate model counting technique, utilizing various hash functions such as multiply-mod-prime, multiply-shift, and XOR. The algorithm makes $\mathcal{O}(\log(|S|))$ calls to the SMT oracle, where S is the set of projection variables. We have implemented a user-friendly tool based on CVC5, which will be released post-publication. pact supports a diverse array of theories including QF_ABV, QF_BVFP, QF_UFBV, QF_ABVFPLRA, QF_ABVFP, QF_BVFPLRA.

To demonstrate runtime efficiency, we conduct an extensive empirical evaluation over 14,202 benchmark instances. Out of these 14,202 instances, pact successfully finished on 603 instances, while Baseline could finish only on 13 instances. Importantly, Baseline fails on counts above 3,570, while pact handles instances with over 1.7×10^{19} solutions.

[1]The source code, experimental data, and technical report are available on the project's page at https://github.com/meelgroup/pact

A. Applications

We now discuss four motivating applications for counting over hybrid SMT formulas.

Robustness Analysis of Automotive Cyber-Physical Systems. Evaluating robustness is crucial in automotive cyber-physical systems (CPS), especially with the rise of autonomous vehicles. Koley et al. [28] encoded the problem using SMT to identify potential CPS attack vectors, incorporating both discrete and continuous variables to represent cybernetic and physical aspects, respectively. This framework extends to a quantitative approach, where the problem becomes an SMT counting query. Robustness is assessed by counting potential attack points, with the projection set defined by the system's input parameters.

Reachability Analysis of Critical Software. Consider a control-flow graph (CFG) of critical software, where we are interested in knowing how many different paths exist in that CFG, such that some violating conditions are reached. We can encode this problem as a counting problem on the SMT formula with discrete and continuous variables, and the projection set would contain Boolean variables indicating whether a node of CFG is reachable. The projected model count will give the number of satisfying paths in CFG.

Quantitative Software Verification. To ensure software reliability, identifying bugs is not always sufficient; a quantitative approach is vital for understanding their impact. A program with an assertion is converted into an SMT formula through a Single Static Assignment (SSA), revealing inputs that lead to assertion failures by counting these specific inputs. Teuber and Weigl [4] reduced the quantitative verification to projected counting over hybrid SMT formulas, wherein the underlying theory is QF_BVFP.

Quantification of Information Flow. In the domain of software reliability, the quantification of information flow represents a critical challenge, particularly in measuring information leakage within industrial software applications. Phan and Malacaria [29] showed that the problem of quantification of information flow in the case of standard programs can be reduced to the task of counting over hybrid SMT formulas defined over QF_BVFP.

II. Preliminaries

Satisfiability Modulo Theory (SMT) [8] combines Boolean satisfiability (SAT) with theories such as integer and real arithmetic, bit-vectors, arrays, enabling efficient and automated analysis of logical formulas involving various data types. *SMT solvers* solves the satisfiability of an SMT formula.

Hybrid SMT Formulas. Some SMT theories are discrete (such as bitvectors and integers), while others are continuous (such as reals and floating points). We define a *hybrid SMT formula* as an SMT formula that combines two or more theories, where there is at least one discrete theory and one continuous theory. For example, a formula in QF_BVLRA is a hybrid formula because it contains both real variables (continuous) and bitvector variables (discrete).

Projection Set and Projected Solutions. Let F represent an SMT formula, where $\mathsf{Vars}(F)$ signifies the set of all variables of F. A *projection set* S is a subset of $\mathsf{Vars}(F)$. Given an assignment τ to $\mathsf{Vars}(F)$, $T_{\downarrow S}$ denotes the projection of τ on S $\mathsf{Sol}(F)$ denotes the set of all solutions to the formula F. $\mathsf{Sol}(F)_{\downarrow S}$ represents the set of all solutions of F projected on S. In the context of this paper, S is a set of discrete variables and therefore, $\mathsf{Sol}(F)_{\downarrow S}$ is a finite set.

Model Counting. Given a formula F and a projection set S, the problem of model counting is to compute $|\mathsf{Sol}(F)_{\downarrow S}|$. An *approximate model counter* takes in a formula F, projection set S, tolerance parameter ε, and confidence parameter δ, and returns c such that $\Pr\left[\frac{|\mathsf{Sol}(F)_{\downarrow S}|}{1+\varepsilon} \leq c \leq (1+\varepsilon)|\mathsf{Sol}(F)_{\downarrow S}|\right] \geq 1-\delta$.

Hash functions. A *hash function* $h : U \to [m]$ maps elements from a universe U to a range $[m] = \{0, 1, \ldots, m-1\}$. A *pairwise independent hash function* is a hash function chosen from a *family* $\mathcal{H}$ of functions $h : U \to [m]$ such that, for any two distinct elements $x_1, x_2 \in U$ and for any $i_1, i_2 \in [m]$: $\Pr[h(x_1) = i_1 \wedge h(x_2) = i_2] = 1/m^2$. A *vector hash function* $h : U^d \to [m]$ extends this concept by mapping d-dimensional vectors of w-bit integers to the range $[m]$. In this paper, the term *hash function* refers to *hash-based constraint*, represented as $h(\mathbf{x}) = \alpha$. The solutions of the formula $F \wedge (h(\mathbf{x}) = \alpha)$ form a subset of the solutions to F, restricted to those where the hash function maps to α.

A. Problem Statement

We shall introduce the projected counting problem, defined by the specific theory of the formula and the projection variables.

Definition 1 ($\mathsf{Count}\mathcal{T}_{\downarrow\mathcal{P}}$(F,S))**.** *Given a logical formula F defined over the SMT theory $\mathcal{T} \cup \mathcal{P}$; and a projection set S on theory $\mathcal{P}$; where $\mathcal{T}$ is either discrete or continuous or combination of both, and $\mathcal{P}$ is a discrete theory,* $\mathsf{Count}\mathcal{T}_{\downarrow\mathcal{P}}(\mathsf{F},\mathsf{S})$ *refers to the problem of counting* $|\mathsf{Sol}(F)_{\downarrow S}|$.

In this work, we consider BV as $\mathcal{P}$. Any possible theory or combination of theories can serve as $\mathcal{T}$. Consequently, the resulting counting problems take forms such as $\mathsf{CountBVFPLRA}_{\downarrow\mathsf{BV}}$, $\mathsf{CountBVFP}_{\downarrow\mathsf{BV}}$, and similar variations.

B. Related Work

The success of propositional model counters, particularly approximate model counters, prompted efforts to extend the techniques to word-level constraints. Chistikov et al. [20] used bit-blasting to extend the propositional model counting technique to word-level benchmarks. Chakraborty et al. [21] designed SMTApproxMC by lifting the hash functions for word-level constraints. Kim and McCamant [22] designed a system to estimate model count of bitvector formulas. Ge et al. [30] developed a probabilistic polynomial-time model counting algorithm for bit-vector problems, and also developed a series of algorithms in the context of related SMT theories to compute or estimate [23], [24], [31] the number of solutions

for linear integer arithmetic constraints. A closely related problem in the hybrid domain of Boolean and rational variables is *Weighted Model Integration* (WMI) [32], which involves computing the volume given the weight density over the entire domain. Extensive research addresses WMI through techniques such as predicate abstraction and All-SMT [33], [34], as well as methods leveraging knowledge compilation [35].

Hashing-based approximate model counting has been extensively studied over the past decades [20], [21], [36]–[45]. Chakraborty et al. [1] showed that variations in a few key components in a generalized framework account for the diversity in prior approaches. While prior works focused on discrete domains such as Boolean variables [39], [40], [44] and bitvectors [20], [21], our approach extends this framework to hybrid SMT formulas.

III. Algorithm and Implementation

We introduce pact, our tool for approximate counting of SMT formulas. It processes a formula F, a set of projection variables S, a *tolerance* ε, and a *confidence* δ to produce an approximation of $|\mathsf{Sol}(F)_{\downarrow S}|$ within the desired tolerance and confidence. The main idea behind pact involves dividing the solution space into equally sized *cells* using hash functions and then enumerating the solutions within each *cell*.

Algorithm 1 pact$(F, S, \varepsilon, \delta, \mathsf{family})$

1: $\mathsf{L} \leftarrow \varnothing, \mathsf{it} \leftarrow 0$
2: $\mathsf{thresh}, \mathsf{numIt}, \ell \leftarrow \mathsf{GetConstants}(\varepsilon, \delta, \mathsf{family})$
3: $\mathsf{C}[0] \leftarrow \mathsf{SaturatingCounter}(F, S, \mathsf{thresh})$
4: **if** $\mathsf{C}[0] \neq \mathsf{T}$ **then return** $\mathsf{C}[0]$
5: **while** $\mathsf{it} < \mathsf{numIt}$ **do**
6: $\quad \mathsf{C} \leftarrow \varnothing, \mathsf{countFound} \leftarrow \bot, i \leftarrow 0$
7: $\quad H \leftarrow \mathsf{GenerateHash}(S, \ell, \mathsf{family})$
8: $\quad$ **while** $\mathsf{countFound} = \bot$ **do**
9: $\quad\quad i \leftarrow \mathsf{NextIndex}(\mathsf{C}, i)$
10: $\quad\quad \mathsf{C}[i] \leftarrow \mathsf{SaturatingCounter}(F \wedge H_{[i]}, P, \mathsf{thresh})$
11: $\quad\quad$ **if** $\mathsf{C}[i] < \mathsf{thresh} \wedge \mathsf{C}[i-1] = \top$ **then**
12: $\quad\quad\quad \mathsf{C}', H' \leftarrow \mathsf{FixLastHash}(F, S, \mathsf{C}, H, i, \ell)$
13: $\quad\quad\quad \mathsf{L}.\mathsf{append}(\mathsf{GetCount}(\mathsf{C}'[i], H'))$
14: $\quad\quad\quad \mathsf{countFound} \leftarrow \top, \mathsf{it}{+}{+}$
15: **return** $\mathsf{FindMedian}(\mathsf{L})$

Algorithm 1 presents the main algorithm pact. The algorithm starts by setting constants of the algorithm, the value for thresh and numIt from the values of ε and δ, depending on the hash family being used. The constants arise from technical calculations in the correctness proof of the algorithm, and the values are shown GetConstants subroutine (Algorithm 3). The value of thresh determines the maximum size of a *cell*. A cell is considered *small* if it has a number of solutions less than this threshold. The value of numIt determines how many times the main loop of the program (lines 5 - 14) is repeated. In each iteration of the main loop, an approximate count is generated, which is stored in the list L. While each of the approximate counts might fail to provide an estimate with the desired δ, the median of the counts of this list gives the approximation of model count with (ε, δ) guarantees.

The main loop of the algorithm begins with the subroutine GenerateHash, which produces a list of hash functions H selected from one of the families $\mathcal{H}_{shift}$, $\mathcal{H}_{prime}$, or $\mathcal{H}_{xor}$ to be used during the current iteration. In each iteration, pact maintains a list C of numbers, where the element $\mathsf{C}[i]$ represents the size of a *cell* after applying the first i hash functions from H, denoted as $H_{[i]}$. The subroutine NextIndex, called in line 9, uses a galloping search to identify an index i in C where the value of $\mathsf{C}[i]$ has been computed. The parameter ℓ in GenerateHash determines the range of hash functions generated. Specifically, (i) for $\mathcal{H}_{shift}$, the range of hash functions is set to 2^ℓ, and (ii) for $\mathcal{H}_{prime}$, GenerateHash constructs hash functions of a range of the smallest prime larger than 2^ℓ.

Following that, in line 10-11, pact checks by a call to SaturatingCounter whether $|\mathsf{Sol}(F \wedge H_{[i]})_{\downarrow S}| < \mathsf{thresh}$ and $|\mathsf{Sol}(F \wedge H_{[i]})_{\downarrow S}| \geq \mathsf{thresh}$. With this condition, pact is enabled a rough estimate of the model count, but to get the estimate within desired error bounds, pact uses the FixLastHash subroutine in line 12. This subroutine eliminates the last hash, $H[i]$, and introduces a new hash function that reduces the number of solution partitions. The subroutine iterates the procedure to find two hash functions h' and h'', such that h'' divides the space into $^k/_2$ parts, while h' divides into k parts. Now if $|\mathsf{Sol}(F \wedge H_{[i-1]} \wedge h')_{\downarrow S}| < thresh, |\mathsf{Sol}(F \wedge H_{[i-1]} \wedge h'')_{\downarrow S}| \geq \mathsf{thresh}$, FixLastHash returns $H' = H_{[i-1]} \wedge h'$ and $C' = |\mathsf{Sol}(F \wedge H_{[i-1]} \wedge h')_{\downarrow S}|$. However, when pact uses $\mathcal{H}_{xor}$, the call to FixLastHash is unnecessary, as it already partitions the solution space into two parts using one hash function.

The subroutine GetCount approximates $\mathsf{Sol}(F)_{\downarrow S}$ by multiplying C' with the number of partitions generated by all the hashes used in H'. This number is then appended to the list L, and pact continues to the next iteration of the main loop. Once the main loop generates count for numIt times, we take the median of all the counts, which is an approximation for $\mathsf{Sol}(F)_{\downarrow S}$ with desired guarantees.

Algorithm 2 FixLastHash$(F, S, \mathsf{C}, H, i, \ell)$

1: **if** family= $\mathcal{H}_{xor}$ **then return** C, H
2: **while** $\ell > 1$ **do**
3: $\quad \ell \leftarrow \lfloor \ell/2 \rfloor$
4: $\quad h^{(\ell)} \leftarrow \mathsf{GenerateHash}(S, \ell, \mathsf{family})$
5: $\quad c \leftarrow \mathsf{SaturatingCounter}(F \wedge H_{[i-1]} \wedge h^{(\ell)}, S, \mathsf{thresh})$
6: $\quad$ **if** $c \neq \top$ **then** $\mathsf{C}[i] = c, H[i] = h^{(\ell)}$
7: $\quad$ **else return** C, H
8: **return** $\bot$

A. Hash functions.

The choice of hash function is one of the most important parts in a hashing-based model counter like pact. In pact, we experiment with three different pairwise independent vector hash functions, which have been used in different literature.

Algorithm 3 GetConstants(ε, δ, family)

1: thresh $\leftarrow 1 + 9.84\left(1+\frac{\varepsilon}{1+\varepsilon}\right)\left(1+\frac{1}{\varepsilon}\right)^2$
2: **if** family $= \mathcal{H}_{xor}$ **then** iters $\leftarrow \lceil 17 \log \frac{3}{\delta} \rceil, \ell \leftarrow 1$
3: **else** iters $\leftarrow \lceil 23 \log \frac{3}{\delta} \rceil, \ell \leftarrow 4$
4: **return** thresh, iters, ℓ

- *Multiply mod prime ($\mathcal{H}_{prime}$)* [46]: For a prime number p, and independent random values $\mathbf{a} = a_0, \ldots, a_{d-1}, b \in [p]$, the hash function from $[p]^d$ to $[p]$ is given by:

$$h_{\mathbf{a},b}(\mathbf{x}) \equiv \left(\sum_{i\in[d]} a_i x_i + b\right) \bmod p = \alpha$$

- *Multiply-shift ($\mathcal{H}_{shift}$)* [47]: For independent random values $\mathbf{a} = a_0, \ldots, a_{d-1}, b \in [2^{\overline{w}}]$, where $\overline{w} \geq w + \ell - 1$ the hash function from $[2^w]^d$ to $[2^\ell]$ is given by:

$$h_{\mathbf{a},b}(\mathbf{x}) \equiv \left(\sum_{i\in[d]} a_i x_i + b\right) [\overline{w} - \ell, \overline{w}) = \alpha$$

- *Bitwise XOR ($\mathcal{H}_{xor}$)* [48]: This is a particular case of the multiply mod prime scheme when $p = 2$.

$$h_{\mathbf{a}}(\mathbf{x}) \equiv \bigoplus_{\{i|a_i=1\}} x_i = \alpha$$

Given the type of hash function to generate, the procedure GenerateHash generates the constraints of the form $h_{\mathbf{a},b} = \alpha$, where α is a randomly chosen element from the domain of the hash function. When this hash function-based constraint H is added to the input formula F, $F \wedge H$ satisfies only those solutions of H for which the hashed value by $H_{\mathbf{a},b}$ is α. therefore, the number of solutions of $F \wedge H$ is approximately p^{th} fraction of number of solution of F. $\mathcal{H}_{prime}$ and $\mathcal{H}_{shift}$ are word-level hash functions that allow us to choose a number of partitions we want to divide our solution space into. When pact uses these hash functions, along with the projection set S, and the family, GenerateHash takes a parameter ℓ, and generates a hash function with domain size p, such that $2^\ell \leq p < 2^{\ell+1}$. Specifically, in $\mathcal{H}_{shift}$, $p = 2^\ell$, in $\mathcal{H}_{prime}$, p is the smallest prime $> 2^\ell$. In case of $\mathcal{H}_{xor}$, $\ell = 1, p = 2$. As S is evident from the context, and $\mathbf{a}, b$ are randomly generated, we use the notation h instead of $h(S)$ to denote the hash functions.

While each of the hashing constraints partitions the solution space into p slices, we often want to partition it into more. To partition the solution space into p^c cells, we use the Cartesian product of c hash functions: $\mathcal{H} \times \mathcal{H} \times \cdots \times \mathcal{H}$. We use the notation $H_{[i]}$ to denote the Cartesian product of the first $i+1$ hash functions, i.e., $H_{[i]} = h_0 \times h_1 \times \cdots \times h_i$.

Slicing. In GenerateHash subroutine of pact, the hash functions have particular domain sizes. But, the bitvectors can have arbitrary width; we *slice* them into bitvectors of smaller width so that the value of (sliced) bitvector lies within the domain of the hash function. Instead of defining hash functions on the variables from S, we define hash functions on *slices* of the variables, defined as follows: For a bitvector $\mathbf{x}$ of width w, we define $\lceil w/\ell \rceil$ slices of width ℓ: $x(\lceil w/\ell \rceil - 1), \ldots, x(1), x(0)$, where $x(i) = x\,[(i+1)\ell - 1 : i\ell]$.

B. Enumerating in a cell.

Another crucial step in pact is to determine when the size of a cell is less than the threshold. pact uses the SaturatingCounter subroutine to enumerate solutions of the formula $F \wedge H_{[i]}$ to determine the size. An SMT solver is employed to find a solution res for the given formula F. The projection of this solution, $\text{res}_{\downarrow S}$, is then *blocked* by adding a constraint $\neg(\text{res}_{\downarrow S})$. Subsequently, the solver is asked to find another solution. This process is repeated in a loop until the solver finds thresh many solutions or reports UNSAT.

C. Searching for the number of partitions.

The number of index of C goes upto $\mathcal{O}(|S|)$. The task for NextIndex subroutine is to find the index, for which the cell size is in the range $[1, \text{thresh}]$. The NextIndex finds the index by $\mathcal{O}(\log(|S|))$ many calls by employing a galloping search method.

We defer the detailed descriptions of GenerateHash, NextIndex, SaturatingCounter, GetCount and to the technical report of the paper.

D. Analysis

Theorem 1. *Let F be a formula defined over a set of variables V and discrete projection variables S. Let* Est $=$ pact$(F, S, \varepsilon, \delta)$ *be the approximation returned by* pact*, and* $c = |\text{Sol}(F)_{\downarrow S}|$*. Then,*

$$\Pr\left[\frac{c}{1+\varepsilon} \leq \text{Est} \leq (1+\varepsilon)c\right] \geq 1 - \delta$$

Moreover, pact *makes $\mathcal{O}(\log(|S|)\frac{1}{\varepsilon^2}\log(\frac{1}{\delta}))$ many calls to an SMT solver.*

Proof. We defer the proof to the accompanying technical report. □

E. Impact of Hash Function Families

Our empirical evaluation indicates SaturatingCounter is computationally the most expensive subroutine during execution of pact. Recall that SaturatingCounter is invoked over the formula instance $F \wedge H_{[i]}$; naturally, the choice of the hash function family impacts the practical difficulty of the instance $F \wedge H_{[i]}$. Below, we highlight the tradeoffs offered by different hash function families along many dimensions.

- *Bit-level vs. Bitvector Operations:* The hash function $\mathcal{H}_{xor}$ operates at the bit level, whereas $\mathcal{H}_{shift}$ and $\mathcal{H}_{prime}$ function on bitvectors, making the latter two more amenable to SMT reasoning. This fundamental difference in operation also enables $\mathcal{H}_{prime}$ and $\mathcal{H}_{shift}$ to vary the hash function's domain sizes, a flexibility not available with $\mathcal{H}_{xor}$.
- *Number of hash functions:* As each hash function cannot partition the solution space by more than two partitions, $\mathcal{H}_{xor}$ requires more number of constraints for counting

TABLE I
NUMBER OF INSTANCES COUNTED. (PROJECTION ON BV VARIABLES.)

Logic	Baseline	pact$_{prime}$	pact$_{shift}$	pact$_{xor}$
QF_ABVFPLRA (30)	–	–	–	4
QF_ABVFP (333)	9	1	1	31
QF_ABV (2922)	–	–	–	287
QF_BVFPLRA (55)	–	–	–	37
QF_BVFP (10434)	2	73	88	227
QF_UFBV (428)	2	9	2	17
Total (14202)	13	83	91	603

an instance compared to $\mathcal{H}_{shift}$ and $\mathcal{H}_{prime}$. A number of constraints generally make the problem harder for the solver to solve efficiently.

- *Complexity of required operations:* Multiplication and modulus operations impose significant computational overhead on SMT solvers, making $\mathcal{H}_{prime}$ and $\mathcal{H}_{shift}$ more complicated than $\mathcal{H}_{xor}$. Moreover, when pact uses $\mathcal{H}_{xor}$, it leverages the native XOR reasoning capability of CryptoMiniSat SAT solver inside pact, further increasing the counter's performance.
- *Requirement of bitwidth:* To represent the value of $\sum a_i x_i$, a bitvector of width $2w$ is required in $\mathcal{H}_{shift}$, whereas a bitwidth of $2w + d$ is needed in $\mathcal{H}_{prime}$, as the hash value in $\mathcal{H}_{shift}$ is calculated modulo 2^{2w}, while in $\mathcal{H}_{prime}$ is computed modulo a prime. Since SMT solver performance degrades quickly with increasing bitwidth, $\mathcal{H}_{shift}$ is a more favorable choice in this context. In an alternative implementation of $\mathcal{H}_{prime}$, each term $a_i x_i$ could be represented modulo a prime, but this would require an additional d modulo operations, increasing complexity.

F. Implementation and Supported Theories

We implement pact on top of CVC5, a modern SAT solver. pact uses CVC5 for parsing the formula and running the SMTSolve procedure. In the problem of Count$\mathcal{T}_{\downarrow\mathcal{P}}$, pact solves all of the theories and theory combinations in SMT-Lib for $\mathcal{T}$ and theories of bit-vectors for $\mathcal{P}$. To enhance efficiency, we utilize the SMT solver in its incremental mode, allowing each subsequent query to leverage the information gained from previous calls. Similar incremental calls are different calls to SaturatingCounter at different iterations of pact.

IV. EXPERIMENTAL EVALUATION

We implemented the proposed algorithm on top of state-of-the-art SMT solver CVC5. We used the following experimental setup in the evaluation:

Baseline. As mentioned in Section I, the existing tools for SMT counting are unable to handle Count$\mathcal{T}_{\downarrow\mathcal{P}}$ problems. The situation is similar to the early days of propositional model counting, wherein enumeration-based counters were employed as a baseline. In the same vein, we developed an enumeration-based counter, referred to as Baseline, that uses the state-of-the-art SMT solver, CVC5, the same solver employed by pact. Baseline operates by asking for a solution from the SMT solver and then appending a blocking clause that includes assignments to the projection variable. Subsequently, it asks the solver for another solution, continuing this process until the solver indicates UNSAT.

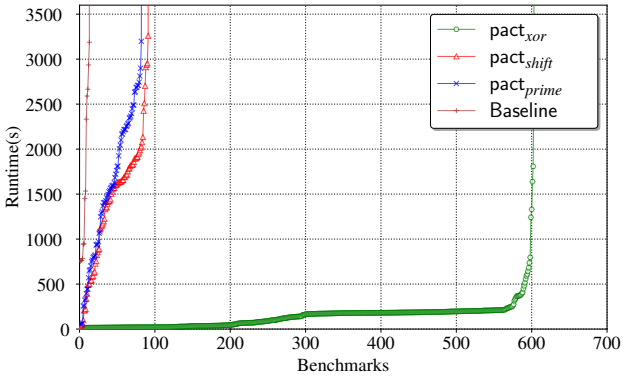


Fig. 1. Comparison of time taken to count by pact and Baseline.

Benchmarks. Our benchmark suite comprises 14,202 instances from the SMT-Lib 2023 release. To minimize bias, we adopted a benchmark selection methodology inspired by early works on propositional model counting. We initially selected all instances supported by six theories. Subsequently, we filtered out instances where the number of solutions was very small (less than 500 models) or where even satisfiability was computationally challenging, as determined by CVC5's inability to find a satisfying assignment within 5 seconds.

Environment. We conducted all our experiments on a high-performance computer cluster, with each node consisting of Intel Xeon Gold 6148 CPUs. We allocated one CPU core and an 8GB memory limit to each solver instance pair. To adhere to the standard timeout used in model counting competitions, we set the timeout for all experiments to 3600 seconds. We use values of $\varepsilon = 0.8$ and $\delta = 0.2$, in line with prior work in the model counting community.

We conduct extensive experiments to understand the following:

RQ1) How does the runtime performance of pact compare to that of Baseline, and how does the performance vary with different hash function families?

RQ2) How accurate is the count computed by pact in comparison to the exact count?

Summary of Results. pact solves a significant number of instances from the benchmarks. It solved 603 instances, while the Baseline counted only 13 instances. Among different hash families, pact performs the best while it uses $\mathcal{H}_{xor}$ hash functions. The accuracy of pact is also noteworthy; the average approximation error is 3.3% while using $\mathcal{H}_{xor}$ hashes.

A. Performance of pact

We evaluate the performance of pact based on two metrics: the number of instances solved and the time taken to solve those instances. To differentiate pact utilizing different hash function families, we use the notations pact$_{prime}$, pact$_{shift}$, and pact$_{xor}$.

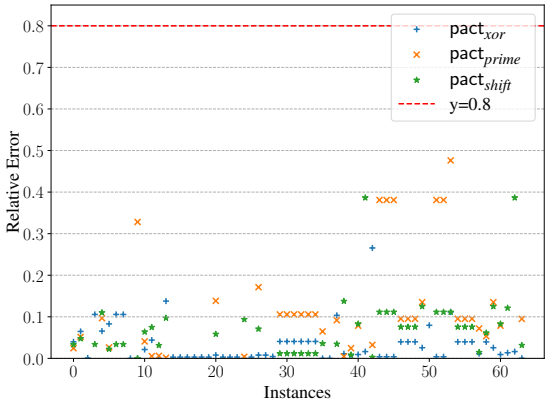


Fig. 2. Accuracy check: observed error in pact vs. the theoretical bound.

Instances solved. For each of the logic, we look at the number of instances solved. Out of 14,202 instances, Baseline could solve only 13 instances. Conversely, pact$_{xor}$ could solve 603 instances, demonstrating a substantial improvement compared to Baseline. The performance varies across different logics, which we represent in Table I. The number of instances solved is 4.2% of the total number of instances, which is expected, given the target problem for these instances is satisfiability, and pact solves a more complex problem.

Comparison of Hash Functions. In the Table I, we compare the performance of pact when it utilizes different hash functions for partitioning the solution space. While all of them perform much better in comparison to Baseline, the best performance is shown by pact$_{xor}$, which solved 603 instances. The performances of pact$_{prime}$ and pact$_{shift}$ are similar, solving 83 and 91 instances.

Solving time comparison. A performance evaluation of Baseline and pact is depicted in Figure 1, which is a cactus plot comparing the solving time. The x-axis represents the number of instances, while the y-axis shows the time taken. A point (i, j) in the plot represents that a solver solved j benchmarks out of the 14,202 benchmarks in the test suite in less than or equal to j seconds. The different curves plot the performance of Baseline and pact with different hash functions.

Performance against the number of solutions. The primary constraint in the enumeration-based method is its capacity for solution counting. The baseline model caps at 3,570 solutions, while pact counts up to 1.7×10^{19}. The performance of pact$_{xor}$ is better than pact$_{shift}$ or pact$_{prime}$ in this regard as well - the maximum count returned by them are in magnitudes of 10^7, while pact$_{xor}$ is of 10^{19}.

B. Quality of Approximation

From our benchmark set, only 13 instances were solved by Baseline. To increase the number of instances for which we know the exact count, we also include the benchmarks with model counts between 100 and 500 in this section of the paper - resulting in 64 instances. We quantify the quality of approximation with the parameter error $e = \max\left(\frac{b}{s}, \frac{s}{b}\right) - 1$, where b is the count from Baseline and s from pact. this definition of error aligns with the ε used in the algorithm's theoretical guarantees and can be interpreted as the observed value of ε. Analysis of all 64 cases found that for pact$_{xor}$, the maximum e to be 0.26 and the average to be 0.03, signifying pact substantially outperforms its theoretical bounds, which is 0.8. In Figure 2, we illustrate the quality of approximation for these instances. The x-axis lists the instances, while the y-axis displays the relative error exhibited by a configuration of pact. A dot (x, y) in the graph indicates x^{th} instance showed a relative error of y. The graph indicates that, for most instances, the error lies below 0.2, with a few instances falling between 0.2 and 0.8. The error for pact$_{shift}$ and pact$_{prime}$ is relatively higher than pact$_{xor}$, with average error being 0.07 and 0.12 and maximum error being 0.39 and 0.48. Our findings underline pact's accuracy and potential as a dependable tool for various applications.

V. Conclusion and Future Work

In this work, we introduced pact, a projected model counter designed for hybrid SMT formulas. Motivated by the diverse applications of model counting and the role of hashing, we explored the impact of various hash functions, examining both bit-level and bitvector-level approaches. Our empirical evaluation demonstrates that pact achieves strong performance on a broad application benchmark set. A dedicated XOR reasoning engine significantly enhanced pact's performance, suggesting that further development of specialized reasoning engines for bit vector-level hash functions could be a promising research direction. Additionally, pact's theoretical framework supports the SMT theory of integers (with specified bounds) as projection variables; this feature is not yet implemented and remains an avenue for future work.

Acknowledgement

We are thankful to Jaroslav Bendik, Supratik Chakraborty, Ashwin Karthikeyan, Aina Niemetz, Mathias Preiner, Uddalok Sarkar, Mate Soos and Jiong Yang for the many useful discussions. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) [RGPIN-2024-05956]. Part of the work was done when Arijit Shaw was a visiting graduate student at the University of Toronto. Computations were performed on the Niagara supercomputer at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario Research Fund: Research Excellence; and the University of Toronto.

References

[1] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "Approximate model counting," in *Handbook of Satisfiability*. IOS Press, 2021.
[2] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting," in *Handbook of satisfiability*. IOS press, 2021.
[3] M. Chavira and A. Darwiche, "On probabilistic inference by weighted model counting," *Artificial Intelligence*, vol. 172, 2008.
[4] S. Teuber and A. Weigl, "Quantifying software reliability via model-counting," in *Proc. of QEST*, 2021.

[5] L. Duenas-Osorio, K. Meel, R. Paredes, and M. Vardi, "Counting-based reliability estimation for power-transmission grids," in *Proc. of AAAI*, 2017.

[6] T. Baluta, S. Shen, S. Shinde, K. S. Meel, and P. Saxena, "Quantitative verification of neural networks and its security applications," in *Proc. of CCS*, 2019.

[7] N. Froleyks, M. Heule, M. Iser, M. Järvisalo, and M. Suda, "Sat competition 2020," *Artificial Intelligence*, vol. 301, 2021.

[8] D. Kroening and O. Strichman, *Decision procedures*. Springer, 2016.

[9] R. Brummayer and A. Biere, "Boolector: An efficient SMT solver for bit-vectors and arrays," in *Proc. of TACAS*, 2009.

[10] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The mathsat5 SMT solver," in *Proc. of TACAS*, 2013.

[11] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of satisfiability*. IOS Press, 2021.

[12] H. Barbosa *et al.*, "cvc5: A versatile and industrial-strength smt solver," in *Proc. of TACAS*, 2022.

[13] A. Niemetz and M. Preiner, "Bitwuzla," in *Proc. of CAV*, 2023.

[14] C. Mattarei, M. Mann, C. Barrett, R. G. Daly, D. Huff, and P. Hanrahan, "Cosa: Integrated verification for agile hardware design," in *Proc. of FMCAD*, 2018.

[15] Á. Hajdu and D. Jovanović, "solc-verify: A modular verifier for solidity smart contracts," in *Proc. of VSTTE*, 2020.

[16] J. Backes *et al.*, "Stratified abstraction of access control policies," in *Proc. of CAV*, 2020.

[17] M. Cashmore, D. Magazzeni, and P. Zehtabi, "Planning for hybrid systems via satisfiability modulo theories," *Journal of Artificial Intelligence Research*, vol. 67, 2020.

[18] E. Schkufza, R. Sharma, and A. Aiken, "Stochastic program optimization," *Communications of the ACM*, vol. 59, no. 2, 2016.

[19] G. Girol, B. Farinier, and S. Bardin, "Not all bugs are created equal, but robust reachability can tell the difference," in *Proc. of CAV*, 2021.

[20] D. Chistikov, R. Dimitrova, and R. Majumdar, "Approximate counting in SMT and value estimation for probabilistic programs," in *Proc. of TACAS*, 2015.

[21] S. Chakraborty, K. Meel, R. Mistry, and M. Vardi, "Approximate probabilistic inference via word-level counting," in *Proc. of AAAI*, 2016.

[22] S. Kim and S. McCamant, "Bit-vector model counting using statistical estimation," in *Proc. of TACAS*, 2018.

[23] C. Ge, F. Ma, X. Ma, F. Zhang, P. Huang, and J. Zhang, "Approximating integer solution counting via space quantification for linear constraints," in *Proc. of IJCAI*, 2019.

[24] C. Ge and A. Biere, "Decomposition strategies to count integer solutions over linear constraints." in *Proc. of IJCAI*, 2021.

[25] C. Ge, "Approximate integer solution counts over linear arithmetic constraints," in *Proc. of AAAI*, 2024.

[26] A. Aydin, L. Bang, and T. Bultan, "Automata-based model counting for string constraints," in *Proc. of CAV*, 2015.

[27] A. Shaw and K. S. Meel, "CSB: A Counting and Sampling Tool for Bitvectors," in *Proc. of SMT Workshop at CAV*, 2024.

[28] I. Koley, S. Dey, D. Mukhopadhyay, S. Singh, L. Lokesh, and S. V. Ghotgalkar, "CAD Support for Security and Robustness Analysis of Safety-critical Automotive Software," *ACM Transactions on Cyber-Physical Systems*, 2023.

[29] Q.-S. Phan and P. Malacaria, "Abstract model counting: a novel approach for quantification of information leaks," in *Proc. of ASIACCS*, 2014.

[30] C. Ge, F. Ma, T. Liu, J. Zhang, and X. Ma, "A new probabilistic algorithm for approximate model counting," in *Proc. of IJCAR*, 2018.

[31] C. Ge, F. Ma, and J. Zhang, "VolCE: An Efficient Tool for Solving #SMT(LA) Problems," in *Proc. of PRUV Workshop at IJCAR*, 2018.

[32] V. Belle, A. Passerini, and G. Van den Broeck, "Probabilistic inference in hybrid domains by weighted model integration," in *Proc. of IJCAI*, 2015.

[33] P. Morettin, A. Passerini, and R. Sebastiani, "Efficient weighted model integration via smt-based predicate abstraction," in *Proc. of AAAI*, 2017.

[34] P. Morettin, A. Passerini, and R. Sebastiani, "Advanced smt techniques for weighted model integration," *Artificial Intelligence*, 2019.

[35] S. Kolb, M. Mladenov, S. Sanner, V. Belle, and K. Kersting, "Efficient symbolic integration for probabilistic inference," in *Proc. of IJCAI*, 2018.

[36] D. Achlioptas, Z. Hammoudeh, and P. Theodoropoulos, "Fast sampling of perfectly uniform satisfying assignments," in *Proc. of SAT*, 2018.

[37] D. Achlioptas and P. Theodoropoulos, "Probabilistic model counting with short xors," in *Proc. of SAT*, 2017.

[38] V. Belle, G. Van den Broeck, and A. Passerini, "Hashing-based approximate probabilistic inference in hybrid domains," in *Proc. of UAI*, 2015.

[39] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "A scalable approximate model counter," in *Proc. of CP*, 2013.

[40] S. Chakraborty, K. S. Meel, and M. Y. Vardi, "Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls." in *Proc. of IJCAI*, 2016.

[41] C. P. Gomes, A. Sabharwal, and B. Selman, "Model counting: A new strategy for obtaining good bounds," in *Proc. of AAAI*, 2006.

[42] M. Soos and K. S. Meel, "BIRD: engineering an efficient CNF-XOR SAT solver and its applications to approximate model counting," in *Proc. of AAAI*, 2019.

[43] L. Stockmeyer, "The complexity of approximate counting," in *Proc. of STOC*, 1983.

[44] J. Yang and K. S. Meel, "Rounding Meets Approximate Model Counting," in *Proc. of CAV*, 2023.

[45] S. Zhao, S. Chaturapruek, A. Sabharwal, and S. Ermon, "Closing the gap between short and long xors for model counting," in *Proc. of AAAI*, 2016.

[46] M. Thorup, "High speed hashing for integers and strings," *arXiv preprint arXiv:1504.06804*, 2015.

[47] M. Dietzfelbinger, "Universal hashing and k-wise independent random variables via integer arithmetic without primes," in *Proc. of STACS*, 1996.

[48] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in *Proc. of STOC*, 1977.

"OOPS!": Out-Of-Band Remote Power Side-Channel Attacks on Intel SGX and TDX

Nimish Mishra
Indian Institute of Technology Kharagpur
nimish.mishra@kgpian.iitkgp.ac.in

Kislay Arya
Indian Institute of Technology Kharagpur
kislayarya536@kgpian.iitkgp.ac.in

Sarani Bhattacharya
Indian Institute of Technology Kharagpur
sarani@cse.iitkgp.ac.in

Paritosh Saxena
Intel Corporation, USA
paritosh.saxena@intel.com

Debdeep Mukhopadhyay
Indian Institute of Technology Kharagpur
debdeep@cse.iitkgp.ac.in

***Abstract*—Prior work shows that remote power attacks on Intel processors are possible through two Model Specific Registers (MSRs): `MSR_PKG_Energy_Status` and `MSR_PP0_Energy_Status`. In response, Intel introduced a defense: a bit in MSR `IA32_MISC_PACKAGE_CTLS` allows users to enable/disable "filtering" mechanism that adds additional noise to energy measurements to harden against power side-channel attacks.**

In this work, we demonstrate that "filtering" does not cover *all* possible avenues of measuring power. On Intel server-grade platforms, components like out-of-band management interface (OOB) exist which also expose telemetric information like in-band energy consumption. For this, we first reverse engineer the protocol structure over which OOB communicates with in-band components. We then show how OOB allows read-only access to the Package Configuration Space (PCS) and note that energy readings through PCS are outside the scope of filtering.

Using this, we establish remote power side-channels on Intel SGX and TDX operational on Intel Sapphire Rapids. We first construct a synchronization mechanism to align in-band execution with out-of-band measurements by leveraging deliberately disabled MSRs. We then use energy readings through OOB PCS to recover 2048-bit RSA keys from MbedTLS operational within in-band Intel SGX (with generic single-stepping assumption). Finally, we also leak AESNI keys from within in-band Intel TDX (without any single-step assumption).

Prior to our work, the literature on side-channels has been focused on attacks leveraging in-band interfaces. Our work establishes the importance of evaluating confidential computing architectures against attack vectors that *combine* abilities of both in-band and out-of-band interfaces to achieve adversarial objectives (that both in-band and out-of-band interfaces cannot independently achieve).

***Index Terms*—Power Side-Channel, Intel SGX, Intel TDX**

I. Introduction

Processor vendors like Intel ship Model Specific Registers (MSRs) which relay telemetric information (like energy consumption) to software for monitoring platform health. For instance, Running Average Power Limit (RAPL) MSRs `MSR_PKG_Energy_Status` and `MSR_PP0_Energy_Status` allow software to readout energy consumption by Package and Power Plane 0 domain respectively. Prior work [1] demonstrates how these MSRs can be used to successfully mount *remote* power side-channels on Intel platforms. The attack exploits the fact that, when executing workloads like AESNI, CMOS circuitry exhibits a power consumption pattern dependent on secret data, even across boundaries of Trusted Execution Environment (TEE) like Intel SGX. This implies these MSRs capture power consumption from cores that execute Intel SGX threads, and expose this consumption information to non-SGX processes. Likewise, subsequent works [2], [3] exploit the correlations between energy consumption and dynamic scaling of processor frequency to mount similar exploits as [1].

In response, Intel released a mitigation: *filtering* of RAPL MSRs [4]. The RAPL filtering mechanism scales noise in the RAPL measurements as the power consumption of the package increases, to harden against power side-channel attacks in [1]–[3]. This defense is exposed to software through the MSR `IA32_MISC_PACKAGE_CTLS`, which software can use to enable/disable filtering. Once enabled, filtering can no longer be disabled until the next reboot. It is general consensus that remote power attacks on patched Intel processors are infeasible. As such, in this work, we ask the following question:

With filtering enabled on RAPL MSRs, do there exist other interfaces which still report unfiltered energy consumption status to software?

In this work, we show that even when RAPL filtering is enabled, there exist interfaces which report unfiltered energy consumption, which an adversary can exploit remotely to compromise Intel SGX and TDX[1]. Concretely, on Intel server-grade processors (like Intel Xeon server products), there exists an out-of-band management interface (named Baseboard Management Controller or BMC) which can access the in-band processor Package Configuration Space (or PCS). The PCS comprises several telemetric readouts, including (but not limited to) package energy consumption. We show this PCS energy consumption readout is *unfiltered*, essentially allowing us to mount remote power side-channels on Intel Xeon processors, even when filtering is enabled. To summarize, we make the following contributions:

[1]Intel Trust Domain Extensions (TDX): the successor to Intel SGX for their Trusted Execution Environment solutions.

1) We first uncover the communication protocol structure between in-band processor package and out-of-band BMC, which is used by the BMC to access in-band processor components. We then detail two access commands allowing readout of in-band telemetric information: `RdIAMSR`, and `RdPkgConfig`. We then state our main observation that `RdPkgConfig` allows BMC access to unfiltered in-band package energy consumption.
2) There does not exist any documented mechanism of communicating from in-band processor to out-of-band BMC. This complicates aligning in-band workload execution with out-of-band telemetric measurements. To circumvent this, we construct a novel unidirectional communication channel to allow in-band software to signal the BMC when to start/stop collecting telemetric measurements.
3) Using the unfiltered package energy consumption reported by `RdPkgConfig`, we mount key recovery attacks on MbedTLS RSA executing within Intel SGX. This attack assumes the presence of a mechanism to perform single-step and zero-step.
4) Likewise, we also use `RdPkgConfig` to perform key recovery attacks against AESNI executing within Intel TDX without any single-step/zero-step requirement. For this, we first establish that energy consumption readouts from `RdPkgConfig` are outside the scope of TDX virtualization, implying `RdPkgConfig` energy consumption readouts leak the execution context of TDX. We then detail how to mount Correlation Power Attack (CPA) styled key recovery against AESNI executing within Intel TDX.

Responsible Disclosure: We responsibly disclosed our findings to Intel in June 2024. Intel completed their analysis by November 6, 2024. Intel has clarified side-channel attack using. OOB power telemetry as out of scope for SGX and TDX in this generation of products. Side channels based on in-band power telemetry are in scope for SGX and TDX.

II. Uncovering out-of-band and in-band communications

We first discuss details of how the out-of-band management interface is configured on our test setup, and how it communicates with in-band system management interface. We then discuss different kinds of in-band measurements possible to be readout from out-of-band, and conclude with key takeaways that eventually allow us to mount remote power side-channel attacks in-band even in the presence of filtering.

A. Threat Model and Experimental Setup

Throughout this work, all analysis and results are reported on Intel(R) Xeon(R) Platinum 8481C (i.e. Intel Xeon Generation 4) processor (codenamed: Xeon Sapphire Rapids) with microcode version `0x2b0004d0`. The Baseboard Management Controller on our system runs the OpenBMC Linux Distribution [5], which is a stripped down version of Linux for management controllers. It uses Yocto, OpenEmbedded, systemd, and D-Bus. In addition to this, the BMC also has three Field Replaceable Units (or FRUs): a single builtin FRU device, one external SOLUM IS162F22 FRU, and one external Intel FFPANEL FRU.

On the BMC side, we assume presence of vanilla software as shipped by the OpenBMC Linux Distribution (i.e. without any adversarial modifications). The adversary can only readout in-band telemetric measurements which the platform management protocols of Intel Xeon natively allow. On the in-band side, we assume all victim code to execute within a Trusted Execution Environment (either Intel SGX or Intel TDX). This implicitly assumes that all non-SGX (alternatively non-TDX) software is inherently untrusted.

As the later sections also detail, our "adversary" is a *combination* of the ① BMC and ② the in-band non-SGX/non-TDX software. Note that both the BMC as well as the in-band non-SGX/non-TDX software are outside the Trusted Computing Base (TCB) of Intel SGX/TDX. This makes a *combination* of the two to be also outside the Trusted Computing Base (TCB) of Intel SGX/TDX.

B. Platform Management Protocols

Intel Xeon systems are a combination of several heterogeneous components glued cohesively together. As such, there exist several system management protocols servicing each of these components. We summarize a few below:

- **In-Band Management**: This refers to managing the components within the silicon (i.e. the die) which communicate through a four-lane ring interconnect [6]. Management of the in-band resources can be done through Control and Status Registers (CSRs) or through Model Specific Registers (MSRs).
- **Message Control Transport Protocol** (MCTP): The MCTP protocol is used over the system management bus (SMBus) for management of peripherals (like Intel FPGA Programmable Acceleration Card) over Inter-Integrated Circuit (I2C) or Peripheral Component Interconnect Express (PCIe).
- **Intelligent Platform Management Interface** (IPMI): This protocol allows the BMC to poll platform sensors for availability, to efficiently readout sensor data, and make such data available to higher level management systems. IPMI predominantly helps in ensuring platform health by consistently reporting on data-points including (but not limited to) core voltage, core temperature, fan speed, Dual-In-Line Memory Module (DIMM) temperatures, and so on.
- **Platform Environment Control Interface** (PECI): PECI is an Intel proprietary interface allowing out-of-band management of in-band processors. PECI is predominantly used for power management and real-time configuration of in-band processor features.

Out of these protocols, IPMI and PECI report telemetric information, and are thus of direct interest to us[2]. IPMI

[2]As a side-note, SMBus has been exploited in [7] for BMC based fault attacks. Fault attacks are outside the scope of our analysis though.

reports telemetric information like CPU energy consumption, CPU average power, DIMM average power, each logical CPU's temperature, fan speeds, and a lot more. Likewise, PECI allows access to in-band measurement interfaces like Model Specific Registers (MSRs) and Package Configuration Space (PCS) which also allow readouts of similar telemetric information. However, PECI offers much more granular measurements than IPMI. To summarize:

Takeaway: Telemetric Measurements from PECI vs IPMI. IPMI offers coarse-granular measurements (i.e. in units of Watts and degree Celsius) which are unsuitable for the adversarial objectives considered in this work. On the other hand, PECI allows readouts from MSRs and PCS, which report more fine-grained measurements (like micro-joules for energy consumption) and are suitable for our adversarial objectives.

We thus restrict our discussion to PECI henceforth. IPMI still may be of interest for *other* adversarial objectives like neural network model fingerprinting as considered in [8] and we leave its investigation to future work.

C. Reversing PECI protocol structure

PECI is an Intel proprietary technology, and as such, not much is documented about it. However, Intel actively contributes to maintaining the PECI ioctl [9] in the OpenBMC Linux Distribution [5], and it provides a handle to reverse engineer PECI protocol structure.

Each PECI command has a fixed base address of `0x30` against which requests are made by the BMC over PECI. A set of completion codes signal what kind of response was received for the PECI request. For instance `PECI_DEV_CC_SUCCESS` (i.e. `0x40`) signals a successful response to a PECI request. Likewise, `PECI_DEV_CC_UNAVAIL_RESOURCE` (i.e. `0x82`) signals the requested resource was unavailable. In addition to this, out of the commands supported by PECI on Intel Xeon, only `GetTemp` and `RdIAMSR` have documented counterparts in-band. We also note that although `WrIAMSR` is an available command in [9], it is (by default) disabled on the Intel Xeon platform experimented upon in this work. For our adversarial objectives, we are only interested in `RdIAMSR` and `RdPkgConfig` since they allow readouts of telemetric information like processor package power consumption.

1) `RdIAMSR`: We first detail `RdIAMSR` since it offers direct access to in-band MSRs, and hence aligns with our adversarial goals. We summarize the protocol structure of `struct peci_rd_ia_msr_msg` on our platform in Fig. 1. For the `RdIAMSR` Request packet, the write length, read length, and the command are always fixed at `0x05`, `0x09`, and `0xB1` respectively. Hyperthread ID allows defining the logical processor whose MSR to read, and MSR address defines the address of the MSR to read. The Response packet comprises the completion code and (if successful) the value of MSR which was read. We now perform two experiments to further investigate `RdIAMSR`'s mapping with in-band MSRs.

In the first experiment, we fix a generic AESNI code [3] on a CPU core `x` in-band, and measure `IA32_FIXED_CTR0`[4] on CPU core `x` through `msr-tools`[5] from another CPU core `y`. Our experimental setup internally uses `smp_call_function_single`[6], which runs the majority of measurement code on CPU core `y`, but schedules a *low-latency* function on CPU core `x` leveraging the operating system scheduler. In our case, this low-latency function is a simple `rdmsr` on `IA32_FIXED_CTR0`. This setup allows CPU core `y` to read MSRs of CPU core `x` without adding too much noise to MSRs of CPU core `x`. Once we have in-band measurements of `IA32_FIXED_CTR0` on CPU core `x`, we now leverage `RdIAMSR` by fixing the hyperthread ID (see Fig. 1) to `x` and MSR address as the address of `IA32_FIXED_CTR0`. We then re-run AESNI *in-band* on CPU core `x` and collect measurements through `RdIAMSR`. We have the following observation:

Takeaway: Hyperthread mapping in-band and out-of-band is identical. MSR measurements in-band on CPU core `x` are *highly* correlated (**average correlation** 0.84 **across** 100 **experimental runs**) with measurements through `RdIAMSR` with Hyperthread ID set to `x`. This suggests that the logical-core-to-hyperthread-ID mapping out-of-band is exactly same as that in-band, saving the adversary additional effort of first reverse engineering this mapping.

Our second experiment is identical to the first, except that we now measure the RAPL MSR `MSR_PKG_ENERGY_STATUS` in place of `IA32_FIXED_CTR0`. We first disable RAPL filtering through `IA32_MISC_PACKAGE_CTLS`[7] and collect RAPL MSR measurements both in-band and out-of-band. Then, we re-enable RAPL filtering by setting the relevant bit of `IA32_MISC_PACKAGE_CTLS` and redo measurements both in-band and out-of-band. We have the following observation:

Takeaway: Both in-band and out-of-band RAPL MSR measurements are filtered. RAPL MSR measurements in-band on CPU core `x` are *highly* correlated (**average correlation** 0.74 **over** 100 **experimental runs**) with measurements through `RdIAMSR` with Hyperthread ID set to `x` for both cases (when filtering was enabled and when it was disabled).

Our experiments lead to the conclusion that enabling filtering in-band also enables filtering on `RdIAMSR` measurements. Moreover, the second MSR reported in [1] (i.e. `MSR_PP0_Energy_Status`) is disabled on Intel Xeon platform. *This makes out-of-band measurements of RAPL MSRs*

[3] https://github.com/intel/cryptography-primitives

[4] Fixed performance counter: counts number of x86 retired instructions.

[5] https://github.com/intel/msr-tools

[6] https://github.com/torvalds/linux/blob/master/kernel/smp.c

[7] And also by additionally disabling Intel SGX and Intel TDX through platform BIOS.

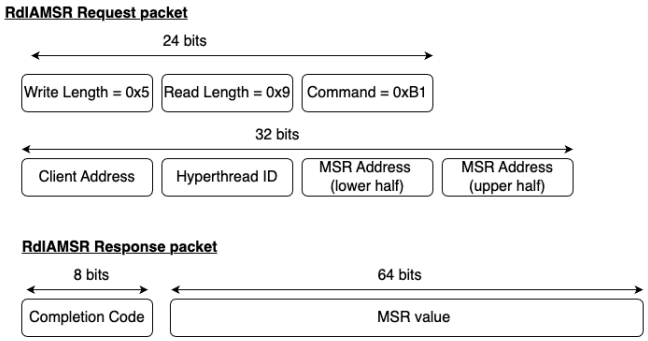


Fig. 1. Uncovered protocol structure of `RdIAMSR` on Intel Xeon (Saphhire Rapids). Each individual box is 8-bits wide.

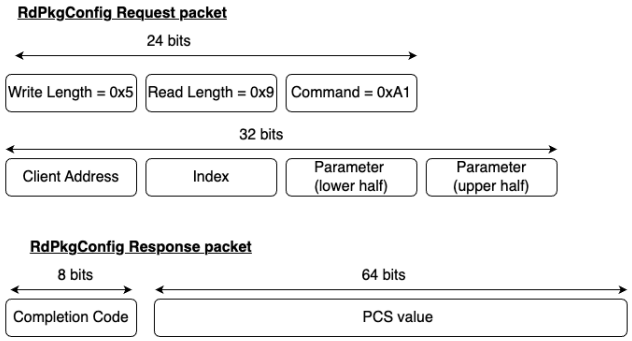


Fig. 2. Uncovered protocol structure of `RdPkgConfig` on Intel Xeon (Saphhire Rapids). Each individual box is 8-bits wide.

through `RdIAMSR` unsuitable for our adversarial goals. We thus shift focus on `RdPkgConfig`.

2) `RdPkgConfig`: We now detail `RdPkgConfig` as it also allows access to telemetric information, and aligns with our adversarial objectives. We summarize the protocol structure of `struct peci_rd_pkg_cfg_msg` in Fig. 2. Compared to the protocol structure of `RdIAMSR`, there are two differences: ① the command is `0xA1`, and ② `RdPkgConfig` takes an 8-bit `index` and 16-bit `parameter` which signifies the aspect of the Package Configuration Space (PCS) to be read. A list of `index` can be found in the definition of `struct peci_rd_pkg_cfg_msg` in [9]. The `parameter`, on the other hand, is undocumented and needs to be fuzzed across all 16-bit possibilities. The Response packet has a completion code and a 64-bit value of the PCS parameter if the request was successful.

We now design the following experiment. First, on the in-band side, we execute a generic AESNI code [8] on a fixed CPU core. We also attach high-priority threads to other CPU cores on the package with negligible workload intensity to reduce noise in captured energy consumption. Then, on the out-of-band side, we first fix the `index` of our `RdPkgConfig` request packet to `PECI_MBX_INDEX_ENERGY_COUNTER`[9] (i.e. the "Energy counter") and fuzz through all possibilities of the 16-bit `parameter`. We summarize our observation:

Takeaway: `RdPkgConfig` with `index = 3` and `parameter = 0xff` readout PCS energy consumption at intervals of 1 millisecond. Moreover, any `parameter` value other than `0xff` readouts a 0, implying `0xff` is the only `parameter` for `index = 3` which readouts valid energy consumption data. Finally, we observe a *poor* correlation (**average correlation** 0.04 **over** 100 **runs**) between data collected from `RdIAMSR` and data from `RdPkgConfig`, with standard deviation of data from `RdPkgConfig` being lesser (i.e. less noise).

[8]https://github.com/intel/cryptography-primitives

[9]There are other `index` which also report RAPL domain data, like `PECI_MBX_INDEX_PKG_RAPL_PERF` but we did not observe any exploitable leakage with it. It is an interesting future work direction to try out other `index` wrt. the adversarial objectives we explore here.

Recall that the RAPL filtering algorithm [4] explicitly adds more noise, increasing the standard deviation of RAPL MSR data. In our experiments, we observe a poor correlation of `RdPkgConfig` with the data from `RdIAMSR`[10] (with filtering enabled), as well as the fact that the distribution of `RdPkgConfig` is much *thinner* wrt. standard deviation (refer Fig. 3). For completeness, we also correlate readouts from `RdPkgConfig` with the *unfiltered* `RdIAMSR` readings; in this experiment, we observe a strong correlation: 0.712. We thus conclude that energy readings from `RdPkgConfig` (`index=3` and `parameter=0xff`) are not filtered. In the next section, we use this fact to craft remote power side-channels.

III. Using `RdPkgConfig` to craft remote power side-channel attacks on Intel SGX and TDX

In this section, we explain how the `RdPkgConfig` (`index=3` and `parameter=0xff`) measurements can be utilized to mount remote power side-channel attacks on Intel Xeon Generation 4 processors in presence of Trusted Execution Environments like Intel SGX and TDX. We first reason why energy consumption is visible across Intel SGX/TDX trust boundaries. We then state (and fix) a roadblock to mounting successful attacks: synchronizing victim execution in-band and adversarial measurements out-of-band by using intentionally disabled MSR `IA32_PMC0`. Finally, we state our attack results.

A. Investigating Visibility of Energy Consumption outside Intel SGX/TDX Trust Boundaries

One precondition of a successful remote power attack is the fact that energy consumption of Intel SGX/TDX is visible across their Trusted Computing Base (TCB) boundary to a non-SGX/non-TDX process. We investigate this further here.

1) Intel SGX: The work in [1] already demonstrates that energy consumption is visible across Intel SGX boundaries. We attribute the success of [1] to the fact that RAPL MSRs are Package scoped (i.e. a single MSR services *all* cores in the Package), while Intel SGX context is applicable only on the logical core where it is executing. Contrastingly to this

[10]We read `MSR_PKG_Energy_Status` from the out-of-band interface.

observation, MSR IA32_SPEC_CTRL[11] is a Core scoped MSR. Thus, when Intel SGX context is applicable on a CPU core, this MSR can be made to behave differently (like disallowing any disables on the mitigations) than it would under non-SGX context [10]. However, a similar restriction cannot be made for a Package scoped MSR (like RAPL MSRs) since they are associated with *multiple* CPU cores (and thus multiple execution contexts). It is possibly for this reason why Intel (in response to [1]) opted for a filtering countermeasure rather than disabling RAPL MSR when Intel SGX context is operational. We note that RdPkgConfig (index=3 and parameter=0xff) is also a Package scoped measurement (since it is readout from the Package Configuration Space), and is thus outside the scope of Intel SGX trust boundaries following a similar reasoning.

2) Intel TDX: Intel TDX follows a slightly different approach to defining its trust boundaries than Intel SGX. Concretely, Intel TDX performs MSR virtualization [11] which virtualizes a given set of MSRs when TDX is operational. Intel TDX also supports context-switching the virtualized MSRs as well. This approach allows TDX to access its own virtualized MSRs as well as save MSR context across context-swiches without reflecting the increments in TDX's MSRs on the MSRs of the non-TDX processes. A concrete example of such virtualized MSRs are Performance Monitoring Counters (PMCs) [11]. However, we note the following:

Takeaway: RAPL MSRs are not virtualized by Intel TDX. Intel TDX chooses not to explicitly virtualize RAPL MSRs [11], and willingly accepts reflecting energy consumption of Intel TDX execution context to the Package scoped RAPL MSRs. We hypothesize that such a design choice was made since RAPL MSRs are anyway protected by RAPL filtering. However, **with the introduction of our side-channel through RdPkgConfig, the design choice of not virtualizing energy consumption allows us to craft remote power-channels on Intel TDX.**

B. Aligning in-band execution and out-of-band measurements

The experiments detailed in the previous section aimed at uncovering properties of RdPkgConfig, and thus we resorted to manual synchronization. However, for actual exploits this violates the threat model we cannot assume any control over victim code within Intel SGX/TDX. We thus construct an unidirectional communication channel from in-band non-SGX/non-TDX context to out-of-band RdPkgConfig that helps synchronize the two. To do so, we leverage the fact that non-SGX/non-TDX MSRs related to performance monitoring unit accept general purpose programming. We can thus *disable* these MSRs (i.e. make their values architecturally invalid), and then use them for communicating between in-band and out-of-band. The detailed experimental setup is enumerated below. We have three entities: Intel SGX/TDX execution context (abbr. **V**), in-band non-SGX/non-TDX execution context (abbr. **A1**), and out-of-band RdPkgConfig measurement context (abbr. **A2**). **V** is the victim, and the *combined* adversary is **(A1, A2)**.

1) **A1** first disables MSR IA32_PMC0 by programming its control MSR IA32_PERFEVTSEL0. **A1** ensures IA32_PMC0 remains disabled throughout the duration of the attack. Meanwhile, **A2** continuously polls the LSB of IA32_PMC0 through RdIAMSR.
2) **A1** then sets the least-significant-bit (LSB) of IA32_PMC0. This triggers **A2** to initiate a measurement from RdPkgConfig (index=3 and parameter=0xff). Call this measurement M_1.
3) **A1** then triggers execution of **V** by relinquishing a CPU core to let **V** execute. **A1** also attaches several high-priority threads (with negligible workload intensity) on other CPU cores in the package to reduce overall noise in the reported energy consumption of the package.
4) Once **V** is done executing, **A1** unsets the LSB of IA32_PMC0. This triggers **A2** to initiate another measurement from RdPkgConfig (index=3 and parameter=0xff). Call this measurement M_2.
5) **A2** reports a *single* sample point as $(M_2 - M_1)$, which captures the energy consumed throughout execution of **V**. The entire process is reported for N sample points.

Note that **V** can be execution context of either Intel SGX or Intel TDX. We now use this setup to mount remote power side-channels on Intel SGX and TDX.

C. Leaking RSA keys across Intel SGX trust boundary

We now show how to use the power side-channel from RdPkgConfig to mount side-channel attacks on the control flow of RSA executions, eventually extracting the RSA private keys from MbedTLS implementation within SGX. We now summarize our attack principle: Depending upon the private key bits, RSA implementations take varying control flow paths which execute different sequence of instructions. Isolation of specific instructions in these different control flow paths and an eventual statistical comparison of their associated power side-channel footprints leads to leaking secret dependent control flow. Moreover, as was the case in [1], we make an additional assumption: single/zero stepping (like [12] for instance) is operational on Intel SGX.

We perform our attack on MbedTLS implementation (version 3.5.1) running inside Intel SGX. We follow similar attack semantics as [1]. MbedTLS supports the windowed modular exponentiation implementation, where a left-to-right sliding window of size d (dependent on the size of input secret exponent) contributes to the computation of the final result in each iteration. Briefly, an adversary needs to perform three sequential steps: ① leak the number of leading zeros, ② determine the secret exponent dependent square/multiply control flow sequence, and ③ from the leaked control flow sequence, recover the secret key.

It is sufficient to perform steps ① and ②, and then rely upon known key extraction algorithms [13] to recover the

[11]Controls enabling/disabling mitigations against speculative attacks.

secret key. If the bit of the secret exponent is 1, a square operation occurs which relies upon Montgomery multiplication (`mpi_montmul`). Among other operations, `mpi_montmul` relies upon a `memset` to initialize and segregate memory. In Intel SGX, this `memset` is replaced by Intel's `fast_memset` implementation, which in turn uses AVX instructions. In summary, whenever the current bit of the secret exponent is 1, an AVX memory instruction (like `movdqa`) is executed; otherwise, a non-scalar `x86` instruction (like `movzx`) is executed. Using a similar observation as [1], the energy consumption profile of `movdqa` and `movzx` is substantially different once they are zero-stepped, easily allowing an adversary to recover secret-dependent control flow, and hence the secret. In our experiments with RSA-2048 (for which MbedTLS chooses the default window size as 5), we invoke zero-steps and measure 10000 samples from `RdPkgConfig` at a sampling rate of 1 millisecond. Subsequently, **we are able to leak the 97.4% of the 2048 bit RSA key in about 5 hours**[12].

D. Leaking AESNI keys across Intel TDX trust boundaries

For AESNI, we follow a similar setup as for RSA but with two main differences: ① we place the AESNI code within Intel TDX, and ② we require no single/zero step assumption.

We follow a similar AESNI setup as [1] and [3]. Within Intel TDX, we implement AESNI through Intel IPP library [13]. As a first experiment, to establish that different keys to AESNI can be distinguished by our side channel, we perform two similar experiments. In the first experiment, the AESNI key is `0x0000000000000000`, while in the second experiment, the AESNI key is `0xffffffffffffffff`. We show the result in Fig. 3, where the energy consumption footprint of the two experiments is clearly distinguishable.

Next we perform CPA on AESNI by choosing the hypothetical power model as the Hamming Weight of Round-0 (in order to recover the round 0 key, from which other keys can be trivially leaked by simulating the AESNI key schedule). We collect 10000 sample points in each trace, with 100000 invocations of `ippsAESEncryptECB` (16 bytes of plaintext in each invocation) contributing to a single sample point. We collected over a million traces for the overall CPA experiment. Overall, we collect over 400 hours worth of data in our setup. We were successfully able to recover **13 bytes out of the unknown 16 bytes of victim AESNI key executing within Intel TDX**[14], thereby demonstrating the applicability of using `RdPkgConfig` to mount CPA attacks on AESNI executing within Intel TDX.

IV. Conclusion

Following the public disclosure of remote power attacks through RAPL MSRs, Intel followed up with a "filtering" defense. We shared our findings regarding `RdPkgConfig` side channel on SGX and TDX with Intel. Intel has clarified

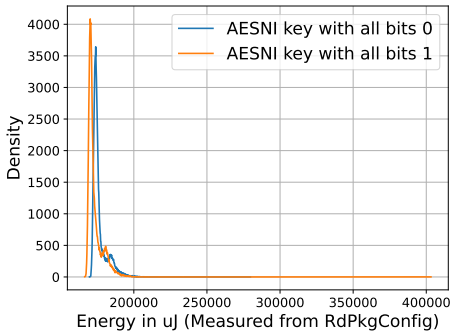


Fig. 3. Distinguishing two AESNI keys through `RdPkgConfig` measurements. T-test score of these experiments is 12.41, implying the two distributions are clearly distinguishable.

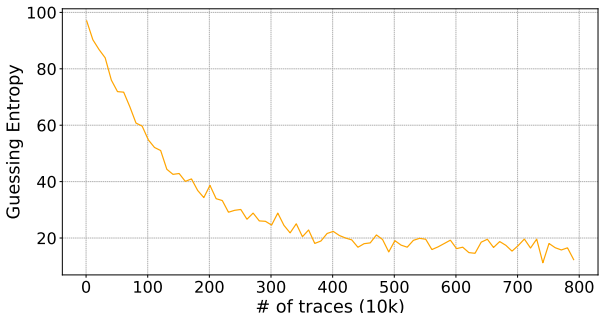


Fig. 4. Guessing Entropy trend (for AESNI key) with increase in number of collected traces.

to us that it was a conscious design decision made based on upon conversations with its customers and their threat models. Intel considers side-channel attack using OOB power telemetry as out of scope in this generation of products. Our work underlines the importance of a OOB power telemetry as a potential source of side channel to confidential compute and it should be comprehended as the threat model for SGX and TDX evolves in the future.

Acknowledgment

The authors would like to thank the anonymous reviewers whose comments helped in improving the final version of the manuscript. Nimish Mishra, Sarani Bhattacharya, and Debdeep Mukhopadhyay would additionally like to acknowledge the Centre on Hardware-Security Entrepreneurship Research and Development (C-HERD), MeitY, Govt. of India, and Information Security Education and Awareness (ISEA) initiative, MeitY, Govt. of India, for partially funding this research.

[12]Remaining secret bits can be easily leaked through lattice reduction.

[13]https://github.com/intel/cryptography-primitives

[14]Entropy of remaining 3 bytes is insufficient to prevent direct extraction, thereby allowing leakage of the entire 16 byte key.

REFERENCES

[1] M. Lipp et. al., "Platypus: Software-based power side-channel attacks on x86," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 355–371.

[2] Y. Wang et. al., "Hertzbleed: Turning power {Side-Channel} attacks into remote timing attacks on x86," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 679–697.

[3] C. Liu et. al., "Frequency throttling side-channel attack," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1977–1991.

[4] Intel, "Rapl advisory intel-sa-00389," 2022.

[5] OpenBMC, "Openbmc linux distribution," https://github.com/openbmc/openbmc/tree/master, 2024.

[6] R. Paccagnella et. al., "Lord of the ring (s): Side channel attacks on the {CPU}{On-Chip} ring interconnect are practical," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 645–662.

[7] Z. Chen et. al., "Pmfault: Faulting and bricking server cpus through management interfaces: Or: A modern example of halt and catch fire," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 1–23, 2023.

[8] N. Mishra et. al., "Too hot to handle: Novel thermal side-channel in power attack-protected intel processors," in *2024 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE, 2024, pp. 378–382.

[9] Intel, "Peci ioctl," https://github.com/openbmc/linux/blob/dev-5.4/include/uapi/linux/peci-ioctl.h, 2024.

[10] ——, "Speculative store bypass. intel-sa-00115," 2018.

[11] ——, "Intel tdx specification documentation," 2023.

[12] D. Skarlatos et. al., "Microscope: Enabling microarchitectural replay attacks," in *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 318–331.

[13] D. J. Bernstein et. al., "Sliding right into disaster: Left-to-right sliding windows leak," in *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*. Springer, 2017, pp. 555–576.

A Fast, Iterative Clock Skew Scheduling Algorithm with Dynamic Sequential Graph Extraction

Shijian Chen[1,2,3], Yihang Qiu[3], Biwei Xie[1,2,3], Mingyu Chen[1,3], Xingquan Li[2,✉]
[1]State Key Lab of Processors, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]Pengcheng Laboratory, Shenzhen, China
[3]University of Chinese Academy of Sciences, Beijing, China
{chenshijian22, qiuyihang23}@mails.ucas.ac.cn, {xiebiwei, cmy}@ict.ac.cn, lixq01@pcl.ac.cn

***Abstract*—Clock skew scheduling (CSS) is a well-known technique that improves design timing slack by adjusting clock latency to flip-flops. CSS requires obtaining timing path information between sequential elements (including flip-flops and I/O ports), known as sequential graph extraction, which is the most time-consuming part of advanced CSS. In this paper, to quickly identify the potential of clock skew in slack optimization, we propose an iterative CSS algorithm that leverages timing propagation to facilitate sequential graph extraction. Then, we provide a comprehensive skew calculation method that considers multiple clock latency constraints, obtaining the target latency of each flip-flop. Finally, we present slack optimization techniques to achieve the target latencies. Our algorithm achieves a 49.11× speedup compared to the advanced CSS algorithm based on partial graph extraction, reducing 90.05% of the extracted edges. Compared to a state-of-the-art CSS-based slack optimization methodology, our algorithm delivers a 27.01× speedup with superior slack improvement.**

I. Introduction

Slack improvement is critical for achieving timing closure, and clock skew scheduling (CSS) is a well-known technique that enhances design timing slack by adjusting clock latency to flip-flops [1], also called useful skew optimization. Previously, Chan et al. proposed NOLO ("no-loop") [2], which applied CSS during post-synthesis to determine predictive skew. This predictive skew facilitated slack optimization, reducing runtime by 66% and improving total negative slack (TNS) by 5%. Similarly, Kim et al. [3] introduced the Fast Predictive Useful Skew Methodology (FPM), which applied CSS to compute predictive skew during placement, leading to significant speedups and improvements in negative slack. These findings highlight the significant potential of CSS in slack optimization.

In fact, CSS has been well-studied [4] [5] [6] [7]. Graph-based CSS [8] is the dominant approach, which uses a parametric shortest path algorithm to find the maximum mean weight cycle (MMWC). The time complexity of the MMWC solution has been proven as $O(nm+n^2 \log n)$, where n and m represent the vertices and edges of the sequential graph[1], respectively. In practice, the MMWC solution achieves significant timing slack improvements and is currently the standard approach for useful skew optimization in commercial EDA tools [2]. However, Albrecht [9] noted that the runtime required to compute the MMWC is negligible compared to the time required to extract the sequential graph from the gate-level timing graph[2]. Consequently, Albrecht [9] first introduced incremental clock skew scheduling (IC-CSS), demonstrating that extracting only 20% of the edges in the sequential graph is sufficient to identify the maximum mean cycle, reducing overall runtime to 5.8%. Subsequently, Wang [10] extended the MMWC solution for slack optimization and proposed an effective cycle-handling method. Notably, Wang continued to employ the same edge extraction method as IC-CSS, and the runtime reported in the study does not account for the time spent on edge extraction.

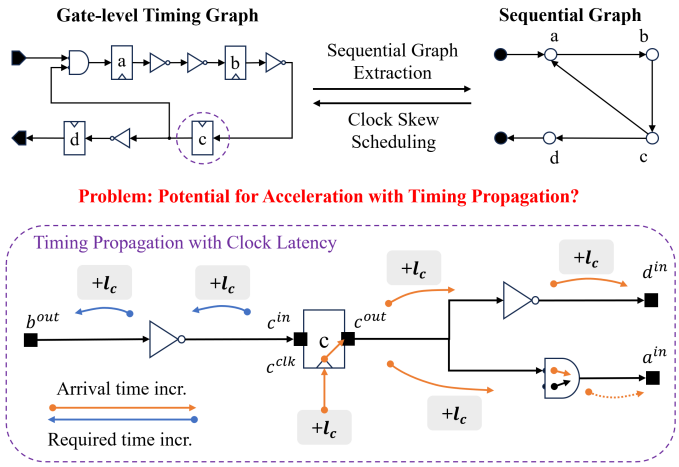


Fig. 1: Default clock skew scheduling flow: (i) Sequential graph extraction, (ii) Clock skew scheduling on the sequential graph. - Inspiring acceleration by leveraging timing propagation.

Despite advancements in CSS and efforts in IC-CSS, two challenges remain. First, we observe that the sequential graph extraction in IC-CSS still retrieves many non-essential edges, indicating that the extraction runtime can be further optimized. Section III provides the details on this issue. Second, no further studies have explored CSS under clock latency constraints. In fact, considering these constraints inevitably requires extracting additional sequential edges, and solving for the MMWC under such conditions may not always be feasible.

Fig. 1 illustrates the key challenges in CSS that our work aims to address. Recently, many static timing analysis (STA) timers have been open-sourced [11] [12] [13], enabling researchers to utilize these timers effectively for slack optimization. Additionally, significant progress has been made in accelerating timers [14] [15] [16] [17] [18]. Since the sequential graph is a subgraph of the gate-level timing graph, this motivates the use of timing propagation to improve CSS.

In this work, to quickly identify the potential for useful skew in slack optimization, we propose an iterative clock skew scheduling algorithm with dynamic sequential graph extraction inspired by timing propagation. The key contributions are summarized as follows.

- We propose a fast iterative clock skew scheduling algorithm that extracts only the essential sequential graph for skew calculation.
- We design a comprehensive skew calculation method based on the essential sequential graph, guaranteeing slack enhancement in each iteration while considering multiple latency constraints.
- We apply reconnection and cell movement techniques to achieve the desired latency.
- Compared to the modified IC-CSS method, our iterative sequential graph extraction reduces the search for essential edges and achieves a shorter runtime. Compared to state-of-the-art CSS-

This work is supported in part by the Major Key Project of PCL (No. PCL2023A03), the National Natural Science Foundation of China (No. 62090021), the Natural Science Foundation of Fujian Province (No. 2024J09045).

[1]Sequential graph: a directed graph with flip-flops or I/O ports as vertices and timing paths as edges.

[2]Gate-level timing graph: a graph used in timing propagation, where vertices represent gates or pins, and edges represent timing arcs or nets.

based slack optimization methodologies, we achieve notable runtime reductions and improved slack.

II. Slack Improvement via Clock Skew Scheduling

A. Problem Formulation

There are two types of timing slack: early slack and late slack.

$$s^E_{u,v} = l_u - l_v + t^{c2q}_u - t^{c2q}_v + d^{min}_{u,v} - t^{hold}_v \tag{1}$$

$$s^L_{u,v} = l_v - l_u + T - t^{c2q}_u - d^{max}_{u,v} - t^{setup}_v \tag{2}$$

Here, $s^E_{u,v}$ and $s^L_{u,v}$ represent the early and late slack on the timing path from u to v, l_u and l_v denote clock latencies, t^{c2q}_u and t^{c2q}_v are the clock-to-Q delays, $d^{min}_{u,v}$ and $d^{max}_{u,v}$ represent the minimum and maximum path delays, T is the given clock period, t^{hold}_v and t^{setup}_v are the hold time and setup time of the flip-flop node v.

Focusing on evaluating and applying CSS to incremental slack optimization, we treat all variables in Eq. (1) and Eq. (2), except clock latencies, as constants. The relationship between slack variation and clock latencies is expressed as follows:

$$\Delta s^E_{u,v} = (l_u - l_v), \quad \Delta s^L_{u,v} = (l_v - l_u) \tag{3}$$

In the following discussion, we use $(l_v - l_u)$ to represent the skew increment for the flip-flop node from u to v.

Define a slack sequence $s = \{s_1, s_2, \ldots, s_{2n}\}$ that includes all early and late slack variables, where n is the number of timing paths. The sequence s is defined to satisfy the following conditions:

- The slack variables are in non-decreasing order, expressed as $s_1 \leq s_2 \leq \cdots \leq s_{2n}$.
- When slack > 0, set slack $= 0$.

We impose constraints of all edges E' on clock skew:

$$l_u - s^L_{u,v} \leq l_v, \text{ and } l_u + s^E_{u,v} \geq l_v, \quad \forall e_{u,v} \in E' \tag{4}$$

We impose constraints on the upper and lower bounds of latency:

$$l^{min}_v \leq l_v \leq l^{max}_v, \quad \forall v \in V \tag{5}$$

Our slack optimization problem consists of finding a solution of Eq. (4) and Eq. (5) such that the sequence s is lexicographically maximal. For example, between $s = \{-5, -4, -2\}$ and $s' = \{-5, -3, -3\}$, we say s' is greater than s because their second elements differ ($-3 > -4$). Since only $s < 0$ requires optimization, it is also referred to as the negative slack optimization (NSO) problem of clock skew scheduling.

Then, we construct a directed sequential graph instance $G = (V, E', w)$ for our CSS problem. Here, V consists of all flip-flop nodes along with two supernodes representing the input and output ports. The timing paths between all nodes form the edge set E' of G, and w represents the edge weights, which correspond to the negative slacks. We direct the late edges from the launch flip-flop to the receive flip-flop, assigning the late slack s^L as the edge weight w. Conversely, we direct early edges from the receive flip-flop back to the launch flip-flop, using the early slack s^E as the edge weight w. By setting the edges in different directions, we incorporate $\Delta s^E_{u,v}$ and $\Delta s^L_{u,v}$ from Eq. (3) into graph G. Finally, we define the weight of each vertex in graph G.

$$w^{out}_u = \min_{e_{u,v} \in E'} w_{u,v}, \quad \forall u \in V \tag{6}$$

To clarify skew calculation, we define the concept of an arborescence and its associated properties. An arborescence is characterized by the following properties: It is a directed acyclic graph, and except for the root vertex, which has no incoming edges, every other vertex has exactly one incoming edge.

We refer to the path from the root to a node of the arborescence as a *path* and define functions for the vertices along it.

$$\alpha(v) = \sum_{e_{u,v} \in P_{\text{root}\to v}} w_{u,v}, \text{ and } \beta(v) = |P_{\text{root}\to v}| \tag{7}$$

where $P_{\text{root}\to v}$ is the path from root to vertex v.

B. Preliminaries of Incremental Clock Skew Scheudling

By extracting only partial edges from the sequential graph, Albrecht [9] achieves incremental clock skew scheduling (IC-CSS), significantly reducing runtime. The key to partial sequential graph extraction is that IC-CSS calculates the maximum delay of outgoing timing paths only once, denoted as d^{out}. When a node u satisfies Eq. (8), it is considered as a critical vertex and has potential critical edges in the outgoing direction:

$$l_u + d^{out}_u - T \geq 0 \tag{8}$$

where l_u is the latency increment and T is the period variable defined in IC-CSS. Since the goal of IC-CSS is to find the minimum period T, it iteratively reduces T. When a critical vertex u satisfying Eq. (8) is identified, it invokes a call-back mechanism to extract all outgoing edges of u. The algorithm terminates when encountering a cycle, which is also referred to as the maximum mean cycle. As not all vertices are critical and do not need their outgoing edges retrieved, IC-CSS completes clock skew scheduling using only a partial sequential graph. We refer to the T-variable updates and latency calculations in IC-CSS as shown in Eq. (9).

$$T = \alpha(v)/\beta(v), \text{ and } l_v = \alpha(v) - \beta(v) \cdot T \tag{9}$$

The detailed process is provided in Alg. 2 of [9].

III. Our Algorithm

A. Exploring Further Acceleration on Sequential Graph Extraction

IC-CSS demonstrates that minimizing the cycle period in CSS does not require extracting the complete sequential graph. To maintain the global minimum cycle period, IC-CSS must extract all outgoing edges for critical vertices to find MMWC, as discussed in Section II-B. Many of these outgoing edges remain unused, presenting opportunities for further acceleration. Moreover, IC-CSS does not incorporate a CSS method under latency constraints, which can impact the identification of the MMWC and require extracting additional edges.

To rapidly solve the proposed NSO problem using clock skew, we circumvent the need to maintain the MMWC, which enables faster batch extraction of all essential sequential edges. Since the STA timer quickly identifies negative (essential) edges in a sequential graph, which corresponds to reporting timing slack violations, we use it to extract and update essential sequential edges. Updating essential edges is achieved by first controlling the clock latency of flip-flops, followed by timing propagation. Next, we further explain the iterative process of extracting a sequential graph and the method for calculating skew within the partial sequential graph, considering latency constraints.

B. Iterative Sequential Graph Construction

If an edge in the sequential graph consistently satisfies $s \geq 0$, it does not require CSS for negative slack optimization. Such edges can be omitted from extraction, thereby reducing runtime. We refer to edges required for skew optimization as **Essential Edges**, while the others are classified as **Non-Essential Edges**.

Fig. 2 shows the core idea of our iterative sequential graph extraction, compared to the sequential graph extraction approach used in IC-CSS. As observed in Fig. 2, the call-back mechanism of IC-CSS extracts many non-essential edges. To extract only essential edges for CSS, we utilize an STA timer to track changes in essential edges and extract them in each iteration, a process known as the Update-Extract Mechanism. We propose a comprehensive process as follows:

i. Extract the partial sequential graph of essential edges.
ii. Generate multiple arborescences based on the partial sequential graph and perform skew calculation.
iii. Add the clock latency values from skew calculation to corresponding flip-flop nodes, perform timing propagation, and repeat the iterative sequential graph construction process.

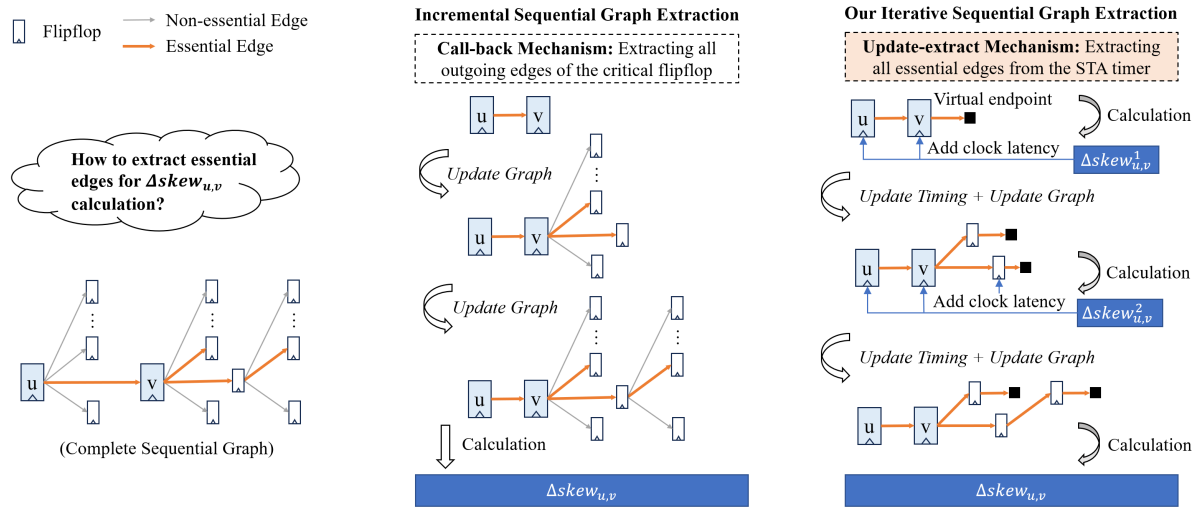


Fig. 2: Comparison of incremental sequential graph extraction from [9] and our iterative sequential graph extraction.

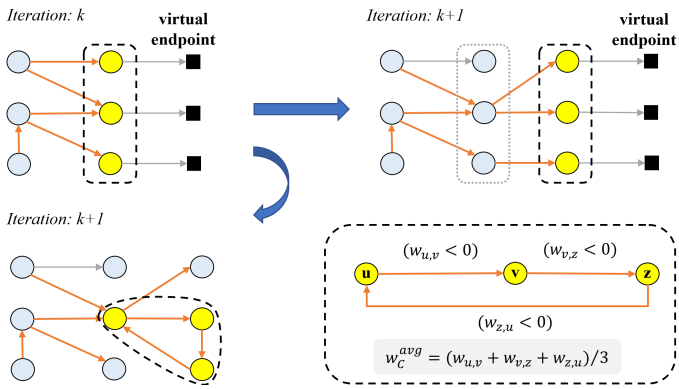


Fig. 3: A snapshot from the iterative sequential graph construction process. The dashed box highlights the vertices that need processing: adding virtual endpoint connections or addressing a cycle.

Finally, we demonstrate that the proposed algorithm reduces the time complexity of sequential graph extraction from nm' to km', where n is the number of vertices, m' is the number of nets in the gate-level timing graph, and k is the number of iterations. Generally, k is much smaller than n.

1) Update-Extract Mechanism

We propose the Update-Extract Mechanism to iteratively extract the sequential graph, which is crucial for avoiding the extraction of non-essential edges. It first updates all computed clock latency values for the clock inputs of the selected flip-flops, followed by timing propagation to update the slack for the relevant timing paths. Finally, the violation timing paths will be extracted as essential edges. In practice, we focus on the newly violated timing endpoints to minimize redundant searches for essential edges and identify new essential edges from these endpoints. The update method for the remaining essential edges is shown in Eq. (10).

$$w^{k+1}_{u,v} = w^k_{u,v} + (l^k_v - l^k_u), \quad \forall e_{u,v} \in E' \tag{10}$$

where $w^k_{u,v}$ represents edge weight in the k-th iteration, and $(l^k_v - l^k_u)$ is the skew increment of the k-th iteration.

2) Iterative Sequential Graph Update Process

Fig. 3 shows a snapshot from the iterative sequential graph construction process. In each iteration, we construct a virtual endpoint for specific vertices, which are used in the skew calculation discussed in Section III-C2. After the update and extract process, the sequential graph construction for the next iteration can result in two scenarios: first, new essential edges are generated; second, a cycle is encountered, as shown in Fig. 3 with the example "u → v → z → u". The existence of a cycle indicates that it is impossible to eliminate the negative slack of all edges within the cycle through skew adjustments. And the negative slack improvement in the cycle C is up to $w^{avg}_C = (w_{u,v} + w_{v,z} + w_{z,u})/3$. According to Eq. (9), treating w^{avg}_C as T and rewriting $l_v = \beta(v) \cdot T - \alpha(v)$, we can calculate the clock latency for all vertices within the cycle C. After setting the clock latency values, we fix them within the cycle, update the timing propagation, and refrain from arranging clock latency to the vertices within the cycle afterward. Finally, we define the termination condition for the algorithm's iteration as the lack of new latency increments across all vertices, indicated by $w^{k+1}_{u,v} = w^k_{u,v}$ in Eq. (10).

C. Skew Calculation on Partial Sequential Graph

Based on Eq. (3), skew represents the latency difference between two flip-flops. Since skew calculation is performed on an arborescence, we set the root latency to zero as the baseline. The latencies of other flip-flops in the arborescence are calculated relative to the baseline. In this section, we will first introduce the constraints on latency, then propose a method for constructing a non-negative latency arborescence, and finally present the complete process for calculating latency.

1) Clock Latency Constraints

Similar to IC-CSS, we set all latencies to be non-negative. During the algorithm's iterative process, we gradually increase the latency of the flip-flops until the final upper bound is reached. The upper bound of latency is constrained by two factors: first, encountering the cycle illustrated in Section III-B2, indicating that further slack optimization is no longer possible. Second, latency cannot arise from another type of slack (for example, not introducing new early violations while optimizing late violations).

According to Eq. (3), in CSS, the late slack increment and early slack increment are mutually constrained. In the following discussion, we focus on late slack optimization, as the case for early slack is analogous. As illustrated in Fig. 4, when there is a late timing

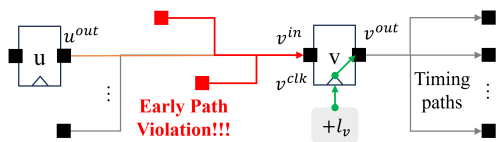


Fig. 4: Increasing clock latency for late slack improvement may introduce new early slack violations.

violation in the timing path from flip-flop u to flip-flop v, we can improve the slack by increasing the clock latency l_v at the clock input of flip-flop v. According to Eq. (1), increasing l_v may lead to new early violation paths with v^{in} as the timing path endpoint. Therefore, the increase in l_v is limited by an upper bound $\hat{s}^E_v$.

$$\hat{s}^E_v = \max\{0, s^E_v\} \tag{11}$$

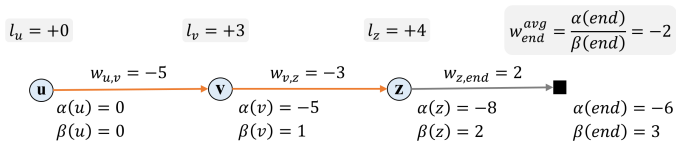


Fig. 5: Latency calculation on a non-decreasing arborescence.

where s_v^E is the minimum early slack among all timing paths with v as the endpoint. Notably, during clock skew scheduling, s_v^E varies dynamically as latency changes. Generally, during clock skew scheduling, all early paths ending at v must be extracted to ensure the upper bound $\hat{s}_v^E$ is not violated. However, our algorithm performs timing propagation in each iteration, during which the timer updates s_v^E. In other words, our algorithm update the latency upper bound $\hat{s}_v^E$ for node v without extracting its early paths, as shown in Eq. (11). This is another key factor enabling our algorithm to minimize sequential edge extraction.

2) Construction of Non-Negative Latency Arborescence

In IC-CSS, the arborescence is constructed using the parametric shortest path algorithm. Since the parametric shortest path algorithm is not the focus of this paper, we do not provide additional details here. The detailed process of arborescence generation can be referred to the Algorithm 1 in [9]. After constructing the arborescence, we calculate latencies based on its structure. We claim that if the edge weights of an arborescence A are non-decreasing from the root to the leaf nodes, the latencies of all vertices on A satisfy $l_u \geq 0$, $\forall u \in A$. Instead of a formal proof, we illustrate this with the example in Fig. 5. In this figure, the edge weights are increasing, and the calculations of α and β follow Eq. (7). The average terminal weight, w_{end}^{avg}, is first computed, after which the latency of each vertex is calculated using the formula $l_u^{max} = \beta(u) \cdot w_{end}^{avg} - \alpha(u), \quad u \in A$. For example, the latency of node z is $l_z = 2 \cdot (-2) - (-8) = +4$. The same method applies to nodes u and v. As a result, the latencies of all nodes satisfy $l_u \geq 0$, $\forall u \in A$. To ensure a non-decreasing arborescence, it suffices to enforce $w_{u,v} < w_v^{out}$ when adding edge $e_{u,v}$ to the arborescence.

3) Clock Latency Calculation by Two-pass Traversal

To ensure that the allocated clock latency for the target vertices remains within the established bounds while maximizing slack improvement, we propose a two-pass traversal method for latency calculation. In the first pass, a reverse topological traversal determines the maximum allowable latency for each vertex. In the second pass, a topological traversal computes the actual latency values.

For each vertex u, we introduce w_u^{avg}, which considers the maximum allowable latency of its successor vertices:

$$w_u^{avg} = \max_{\{u,v\} \in E} \{ \frac{\alpha(u) + w_{u,v} + l_v^{max}}{\beta(u) + 1} \} \tag{12}$$

Using w_u^{avg}, the maximum allowable latency l_u^{max} is calculated as:

$$l_u^{max} = \beta(u) \cdot w_u^{avg} - \alpha(u) \tag{13}$$

Additionally, l_u^{max} can be utilized to enforce the latency upper bound

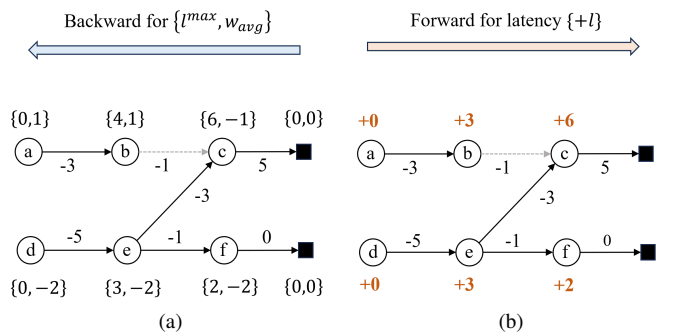


Fig. 6: An example of two-pass traversal for latency calculation.

Algorithm 1 Our Iterative Clock Skew Scheduling

Input: The gate-level timing graph G_t; The sequential graph $G = (V, E, w)$, E and w is given implictly by G_t.
Output: The target latency l_v^* for each node $v \in V$

1: Initialize Target Latency $l_v^* = 0$ $(\forall v \in V)$, Arborescences A, $k = 0$
2: **repeat**
3: $G^k \leftarrow (V, \text{extract_edges}(G_t), w)$ [Section III-B1]
4: $A \leftarrow$ Non-Negative Construction [Section III-C2]
5: **if** exist cycle C in A **then**
6: $l_v^k \leftarrow$ Cycle Latency Calculation [Section III-B2]
7: $l_v^* = l_v^* + l_v^k$
8: Update Timing & Continue
9: **end if**
10: $l_v^k \in V \leftarrow$ Two-Pass Calculation [Section III-C3]
11: $l_v^* = l_v^* + l_v^k$
12: Update Timing & $(k = k + 1)$
13: **until** $l_v^k = 0$, $\forall v \in V$
14: **return** l_v^*

constraints outlined in Section III-C1. Finally, the actual latency l_v is scheduled in topological order:

$$l_v = \min\{l_v^{max}, l_u - w_{u,v}\} \tag{14}$$

To illustrate the latency calculation process, we refer to Fig. 6. In Fig. 6(a), the traversal begins from a virtual end node. The l_{end}^{max} for all virtual vertices is set to 0. Since virtual vertices lack outgoing edges, w_{end}^{avg} is therefore 0. The traversal continues, calculating up to vertex e using Eq. (12) and Eq. (13). For example, when calculating from vertex e to c, we find: $w_e^{avg} = \frac{(-5)+(-3)+6}{1+1} = -1$. Similarly, for the edge from e to f: $w_e^{avg} = \frac{(-5)+(-1)+2}{1+1} = -2$. The value $w_e^{avg} = -2$ is then used to calculate l_e^{max}. For the edge $e_{b,c}$, even though vertices b and c belong to different arborescences, the calculation method remains consistent. Ultimately, all $\{l^{max}, w_{avg}\}$ pairs are determined for latency calculation according to Eq. (14), as shown in Fig. 6(b). Notably, vertex b requires only a latency of +3 to resolve the timing violation on edge $e_{a,b}$.

D. Algorithm and Analysis

Alg. 1 presents the complete process of iterative clock skew scheduling. After initializing the necessary variables (Line 1), the algorithm extracts essential edges to construct the sequential graph G^0 for the first iteration (Line 3). Note that we use the function extract_edges(G_t) to invoke the timer for essential edge extraction. It then constructs the non-negative latency arborescences for skew calculation (Line 4). When a cycle C occurs in the arborescence, the algorithm calculates and records the latencies in that cycle (Lines 5-8). It then fixes the latency on C and reconstructs the sequential graph. Otherwise, it performs the two-pass latency calculation (Line 10). The algorithm records each iteration's target latency l^* (Lines 7,11). After finishing the clock skew scheduling, it puts the latency to each flip-flop and performs timing propagation for the next iteration (Lines 8,12). Finally, if all latencies are equal to zero, indicating no further slack improvement, the algorithm records k and returns all the target latencies.

IC-CSS extracts the outgoing edges for the critical vertices, potentially giving a time complexity up to $O(nm')$, where n is the number of flip-flops, and m' is the number of nets in the gate-level timing graph. Our algorithm performs timing propagation at each iteration. Therefore, the final time complexity for our sequential graph extraction is $O(km')$, where k is the number of iterations. Generally, k is much smaller than n. The significant reduction in time for our sequential graph extraction is primarily due to two factors: first, the absence of MMWC identification; and second, our focus on searching only essential edges during iterations.

E. The Modified Incremental Clock Skew Scheduling

We adapt and extend the IC-CSS algorithm from [9] to address the same NSO problem as our proposed algorithm, with modifications to satisfy latency constraints in clock skew scheduling. The following three functionalities are modified:

i. Termination Condition Adjustment: The termination condition of the IC-CSS algorithm is replaced with the cycle latency calculation method described in Section III-B2. The algorithm continues generating arborescences after processing the cycle.
ii. Latency Constraint Edge Extraction: We introduce functionality to extract latency constraint edges, as outlined in the extension of [9]. When the IC-CSS algorithm adds a critical edge $e_{u,v}$ to the arborescence, it computes the latency l_v for vertex v and compares it with $\hat{s}_v^E$ (or $\hat{s}_v^L$ for early improvement), as defined in Section III-C1. If $l_v > \hat{s}_v^E$, it triggers a callback mechanism to extract all constraint edges associated with vertex v.
iii. Latency Calculation Update: The latency calculation in IC-CSS is replaced with the approach described in Section III-C3. For each arborescence generated by IC-CSS, latencies are computed, and edge weights are updated according to Eq. (10). Subsequently, the $\hat{s}_v^E$ values for all vertices are refreshed. This process iterates until no further latency increments are possible.

We refer to the modified version of the IC-CSS algorithm as IC-CSS+.

IV. Slack Optimization Techniques

In this section, we introduce two slack optimization techniques: LCB-FF reconnection and cell movement.

A. LCB-FF Reconnection

As shown in Fig. 7(a), we introduce a flip-flop reconnection technique to local clock buffers (LCBs) to effectively increase clock latency. Alg. 1 provides the target latency l^* for flip-flops. Accordingly, the objective of LCB-FF reconnection can be formulated as:

$$\min |l_v - l_v^*| \tag{15}$$

where l_v represents the actual latency of vertex v. However, we observed that when an LCB is connected to multiple flip-flops located far apart, it can result in uncontrollable Clock Pessimism Path Reduction (CPPR) issues. We should mitigate the negative impact on the steiner tree topology caused by reconnecting an LCB to multiple flip-flops. Therefore, we prohibit reconnection for two types of LCBs: first, when the LCB's fanout has reached its maximum limit, and second, when the LCB has already undergone reconnection. We start flip-flop reconnection in descending order of l_v^* values, using the Elmore delay model to convert l_v^* into target distance $Dist_v^*$.

$$Dist_v^* = Elmore(l_v^*) \tag{16}$$

We then define the number of candidates and select LCBs for the candidate set from the distance matrix in descending order based on $Dist_v^*$. Notice that we maintain a distance matrix between the flip-flops that require reconnection and all LCBs during initialization. Finally, the reconnection cost is computed based on Eq. (15). In practice, we incorporate the negative impact on other flip-flops into the reconnection cost.

B. Cell Movement to Refine Early Violations

As shown in Fig. 7(b), we employ cell movement refinement to mitigate newly generated early violations. We first identify all movable cells along the early violation path and attempt to shift each cell in the north, south, east, and west directions. After each move, we perform a local timing update. If a cell achieves a longer arrival time for early slack improvement, further movement of that cell is halted. The cell movement distance is centered around the current position, starting at 0.1 times the maximum displacement constraint and gradually increasing until the maximum displacement is reached.

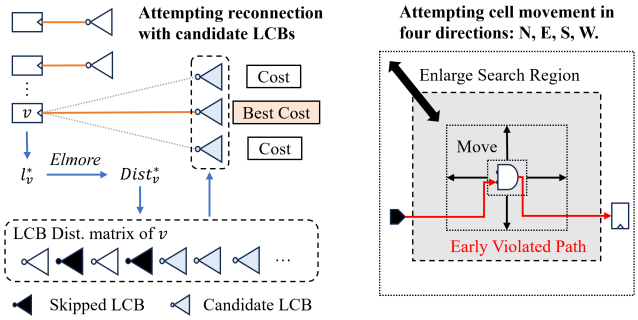


Fig. 7: The proposed slack optimization techniques: (a) LCB-FF reconnection and (b) Cell movement for early violations.

V. Experimental Results

Our CSS algorithm and slack optimization techniques are implemented in C++11 and executed on an Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz with 64GB of RAM. Timing propagation and essential edge extraction are conducted using the open-source timer described in [13]. The evaluation dataset is derived from the ICCAD 2015 contest benchmark [19], with detailed information summarized in TABLE I. To demonstrate the effectiveness of clock skew scheduling for slack improvement, we utilize the first-place result of the ICCAD 2015 contest as the input for our incremental timing optimization. Notably, we adhere to the constraint that each LCB can connect to a maximum of 50 flip-flops. All results are evaluated using the official ICCAD 2015 evaluator, and no violations of the contest constraints are observed in our experimental results. We divide our slack optimization process into two stages: early slack optimization under late slack constraints, followed by late slack optimization under early slack constraints.

TABLE I presents the comprehensive early/late worst negative slack (WNS) and total negative slack (TNS), runtime details including clock skew scheduling (CSS), slack optimization (OPT), the number of extracted edges, and the increase in HPWL. For early optimization alone, our algorithm (Ours-Early) achieves a 22.74% further improvement in WNS and a 7.22% in TNS compared to the Fast Predictive Useful Skew Methodology (FPM), with a 27.01× speedup over FPM, accompanied by a negligible increase in HPWL.

In the complete slack optimization task, our algorithm and the Modified Incremental Clock Skew Scheduling algorithm (IC-CSS+) achieve an 87.5% improvement in early WNS and approximately 88% improvement in early TNS compared to the baseline (Contest 1st). For late optimization, both achieve a 2.02% improvement in WNS and approximately 12.3% improvement in TNS. During the clock skew scheduling phase, our proposed iterative clock skew scheduling method is 49.11× faster, as it reduces the extraction of essential edges by 90.05%. Even after applying the same slack optimization phase, our proposed slack improvement method still achieves an 11.83× speedup. Experimental results demonstrate that clock skew scheduling can be accomplished with significantly less sequential graph information, enabling a rapid evaluation of the potential of clock skew for negative slack optimization.

According to TABLE I, our algorithm eliminates all early violations except for superblue7, which demonstrates its effectiveness. Furthermore, we report runtime information for Ours-Early. The results indicate that CSS is completed within a few seconds, since only a small number of edges are extracted. The runtime bottleneck is primarily due to the OPT phase. These findings highlight the efficiency of our algorithm in clock skew scheduling and underscore the potential of useful skew for optimizing negative slack.

As shown in Fig. 8, we present the iterative process of our algorithm on superblue18. The algorithm first performs early clock skew

TABLE I: Comparison of the Fast Predictive Useful Skew Methodology (FPM) [3], the Modify Incremental Clock Skew Scheduling (IC-CSS+) based on [9], and our algorithm. Since FPM only optimizes for the early violations, we introduce Ours-Early for a fair comparison. The runtime of FPM is reconstructed based on Figure 7(b) in [3]. Similar to our algorithm, IC-CSS+ conducts comprehensive optimization for both early and late violations. All algorithms use the Contest 1st results as input.

Benchmark	Statistics			Solution	Early(ps)		Late(ns)		Runtime(s)			#Extract Edge	HPWL Incr(%)
	#Cells	#FFs	#LCBs		WNS	TNS	WNS	TNS	CSS	OPT	Total		
superblue1	1.21M	144K	7.2K	Contest 1st	-16.65	-80.89	-4.57	-351.23	—	—	—	—	—
				FPM [3]	-0.43	-0.43	-4.57	-351.21	—	—	495	—	0.0003
				Ours-Early	0	0	-4.57	-351.23	1.75	22.36	24.11	32	0.0016
				IC-CSS+	0	0	-4.51	-316.94	1335.73	114.38	1450.11	3212965	0.1619
				Ours	0	0	-4.51	-316.95	7.47	97.52	104.99	81002	0.1576
superblue3	1.21M	168K	8.4K	Contest 1st	-13.13	-214.03	-8.71	-1160.04	—	—	—	—	—
				FPM [3]	-5.54	-29.05	-8.70	-1160.07	—	—	1330	—	0.0046
				Ours-Early	0	0	-8.71	-1160.20	2.18	21.52	23.7	77	0.0101
				IC-CSS+	0	0	-8.44	-1107.61	249.11	118.59	367.7	384685	0.2626
				Ours	0	0	-8.44	-1107.63	7.05	106.32	113.37	21769	0.2635
superblue4	796K	177K	8.8K	Contest 1st	-12.28	-53.84	-5.76	-2464.56	—	—	—	—	—
				FPM [3]	0	0	-5.76	-2462.93	—	—	358	—	0.0006
				Ours-Early	0	0	-5.76	-2464.53	2.39	12.99	15.38	47	0.0017
				IC-CSS+	0	0	-5.76	-2222.36	2512.37	158.22	2670.59	4524898	1.7098
				Ours	0	0	-5.76	-2223.34	92.68	125.55	218.23	802408	1.6994
superblue5	1.09M	114K	5.7K	Contest 1st	-36.77	-618.27	-24.29	-5842.23	—	—	—	—	—
				FPM [3]	-36.77	-268.60	-24.29	-5842.28	—	—	755	—	0.0038
				Ours-Early	0	0	-24.29	-5842.23	2.85	23.93	26.78	177	0.0096
				IC-CSS+	0	0	-24.29	-5065.66	2645.23	220.94	2866.17	8918052	1.7802
				Ours	0	0	-24.29	-5066.23	20.31	240.92	261.23	30380	1.7831
superblue7	1.93M	270K	13.5K	Contest 1st	-6.75	-1958.34	-15.22	-1510.76	—	—	—	—	—
				FPM [3]	-6.38	-1858.48	-15.21	-1510.79	—	—	1237	—	0.0008
				Ours-Early	-6.75	-1872.18	-15.22	-1510.69	3.35	61.91	65.26	449	0.0023
				IC-CSS+	-6.75	-1870.91	-15.22	-1387.73	803.96	232.11	1036.07	502292	0.5069
				Ours	-6.75	-1874.54	-15.22	-1388.43	31.73	227.50	259.23	5610	0.5084
superblue10	1.88M	241K	12.1K	Contest 1st	-5.15	-373.75	-16.08	-31517.8	—	—	—	—	—
				FPM [3]	-2.20	-3.47	-16.07	-31518.0	—	—	1049	—	0.0010
				Ours-Early	0	0	-16.08	-31518.1	3.19	37.10	40.29	220	0.0060
				IC-CSS+	0	0	-15.83	-29449.7	11006.17	411.93	11418.1	15481826	3.5442
				Ours	0	0	-15.83	-29448.8	202.55	378.89	581.44	2395607	3.5442
superblue16	982K	143K	7.1K	Contest 1st	-7.55	-37.64	-3.85	-265.57	—	—	—	—	—
				FPM [3]	0	0	-3.84	-265.57	—	—	430	—	0.0006
				Ours-Early	0	0	-3.85	-265.57	1.28	13.53	14.81	25	0.0005
				IC-CSS+	0	0	-3.49	-181.98	212.93	100.00	312.93	429939	0.4397
				Ours	0	0	-3.49	-182.94	7.00	89.01	96.01	4147	0.4412
superblue18	768K	104K	5.2K	Contest 1st	-1.95	-6.86	-3.82	-775.84	—	—	—	—	—
				FPM [3]	0	0	-3.81	-775.87	—	—	301	—	0.0015
				Ours-Early	0	0	-3.82	-775.84	0.91	9.24	10.15	11	0.0006
				IC-CSS+	0	0	-3.78	-659.27	182.88	72.19	255.07	347689	0.8218
				Ours	0	0	-3.78	-659.24	15.17	72.60	87.77	22508	0.8221
Avg. Ratio (Based on 1st)				FPM [3]	+64.76%	+80.83%	+0.06%	+0.01%	—	—	+744.38s	—	**-0.0017%**
				Ours-Early	**+87.50%**	**+88.05%**	+0.00%	+0.00%	+2.24s	+25.32s	**+27.56s**	+130	-0.0041%
				IC-CSS+	+87.50%	**+88.06%**	+2.02%	**+12.33%**	+2368.55s	+178.54s	+2547.09s	+4225293	-1.1534%
				Ours	+87.50%	+88.03%	+2.02%	+12.27%	**+47.99s**	**+167.29s**	**+215.28s**	**+420429**	**-1.1524%**
Avg. Improvement				FPM [3]	**+22.74%**	**+7.22%**	-0.06%	-0.01%	—	—	**27.01×**	—	-0.0024%
				IC-CSS+	+0.00%	-0.03%	+0.00%	-0.06%	**49.11×**	1.04×	**11.83×**	**-90.05%**	+0.001%

scheduling, followed by early slack optimization using the LCB-FF reconnection and cell movement techniques. Similarly, the algorithm carries out late slack optimization. To enhance stability, a portion of early violations is amplified during early slack optimization. Notably, the first round of late clock skew scheduling reveals a significant slack improvement, indicating that substantial skew is available on adjacent paths to help resolve timing violations.

VI. Conclusion

In this paper, we address the negative slack improvement problem through a fast, iterative clock skew scheduling algorithm. This method rapidly evaluates the potential for slack improvement and uses the calculated latencies to guide slack optimization techniques, thereby achieving significant timing improvements and runtime reduction. Furthermore, our algorithm supports controlling flip-flop clock latency constraints, enabling customized clock skew scheduling. Fast clock skew scheduling enables the incorporation of useful skew considerations into the EDA physical design flow. In future work, we plan to apply our algorithm to open-source flows to guide clock tree synthesis and integrate it with logic path optimization.

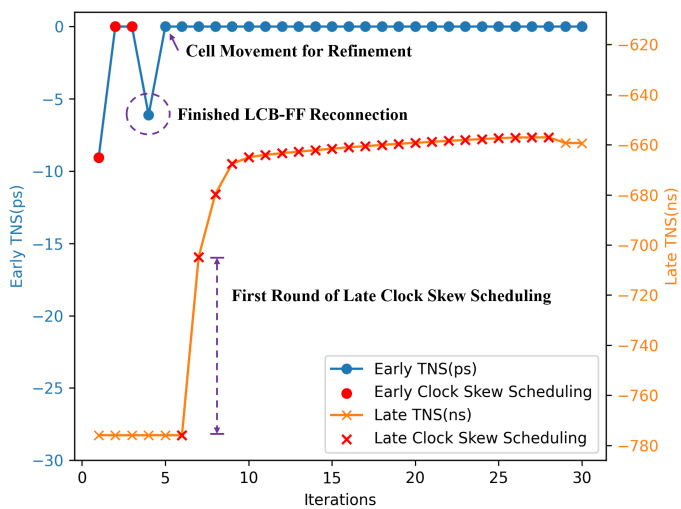


Fig. 8: Slack optimization iterations of superblue18.

REFERENCES

[1] Y.-C. Lu, W.-T. Chan, D. Guo, S. Kundu, V. Khandelwal, and S. K. Lim, "RL-CCD: Concurrent clock and data optimization using attention-based self-supervised reinforcement learning," in *Proceedings of the 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[2] T.-B. Chan, A. B. Kahng, and J. Li, "NOLO: A no-loop, predictive useful skew methodology for improved timing in IC implementation," in *Proceedings of the 15th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2014, pp. 504–509.

[3] S. Kim, S. Do, and S. Kang, "Fast predictive useful skew methodology for timing-driven placement optimization," in *Proceedings of the 54th ACM/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[4] J. P. Fishburn, "Clock skew optimization," *IEEE transactions on computers*, vol. 39, no. 7, pp. 945–951, 1990.

[5] M. Ni and S. O. Memik, "A revisit to the primal-dual based clock skew scheduling algorithm," in *Proceedings of the 11th International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2010, pp. 755–764.

[6] G. Zgheib, Y. S. Lu, and I. Ganusov, "Clock skew scheduling: Avoiding the runtime cost of mixed-integer linear programming," in *Proceedings of the 31st International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2021, pp. 327–333.

[7] A. Wagle, J. Yang, N. Kulkarni, and S. Vrudhula, "A new approach to clock skewing for area and power optimization of ASICs using differential flipflops and local clocking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 11, pp. 4164–4176, 2023.

[8] C. Albrecht, B. Korte, J. Schietke, and J. Vygen, "Maximum mean weight cycle in a digraph and minimizing cycle time of a logic chip," *Discrete Applied Mathematics*, vol. 123, no. 1-3, pp. 103–127, 2002.

[9] C. Albrecht, "Efficient Incremental Clock Latency Scheduling for Large Circuits," in *Proceedings of the Design Automation & Test in Europe Conference (DATE)*. Munich, Germany: IEEE, 2006, pp. 1–6.

[10] K. Wang, H. Fang, H. Xu, and X. Cheng, "A fast incremental clock skew scheduling algorithm for slack optimization," in *Proceedings of the 2008 Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE, 2008, pp. 492–497.

[11] T.-W. Huang and M. D. Wong, "OpenTimer: A high-performance timing analysis tool," in *Proceedings of the 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 895–902.

[12] "OpenSTA", *https://github.com/The-OpenROAD-Project/OpenSTA*.

[13] G. Flach, M. Fogaça, J. Monteiro, M. Johann, and R. Reis, "Rsyn: An extensible physical synthesis framework," in *Proceedings of the 2017 ACM on International Symposium on Physical Design (ISPD)*, 2017, pp. 33–40.

[14] T.-W. Huang, G. Guo, C.-X. Lin, and M. D. Wong, "OpenTimer v2: A new parallel incremental timing analysis engine," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 40, no. 4, pp. 776–789, 2020.

[15] G. Guo, T.-W. Huang, Y. Lin, and M. Wong, "GPU-accelerated critical path generation with path constraints," in *Proceedings of the 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[16] Z. Guo, T.-W. Huang, and Y. Lin, "Accelerating static timing analysis using CPU–GPU heterogeneous parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 12, pp. 4973–4984, 2023.

[17] T.-W. Huang, B. Zhang, D.-L. Lin, and C.-H. Chiu, "Parallel and heterogeneous timing analysis: Partition, algorithm, and system," in *Proceedings of the 2024 International Symposium on Physical Design (ISPD)*, 2024, pp. 51–59.

[18] S. Lin, G. Guo, T.-W. Huang, W. Sheng, E. Young, and M. Wong, "GCS-Timer: Gpu-accelerated current source model based static timing analysis," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[19] M.-C. Kim, J. Hu, J. Li, and N. Viswanathan, "ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite," in *Proceedings of the 2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2015, pp. 921–926.

Location is Key: Leveraging LLM for Functional Bug Localization in Verilog Design

Bingkun Yao
City University of Hong Kong
bingkun.yao@cityu.edu.hk

Ning Wang
City University of Hong Kong
nwang227-c@my.cityu.edu.hk

Jie Zhou
Southeast University
zhoujie_0428@163.com

Xi Wang
Southeast University
xi.wang@seu.edu.cn

Hong Gao
Zhejiang Normal University
honggao@zjnu.edu.cn

Zhe Jiang*
Southeast University
101013615@seu.edu.cn

Nan Guan*
City University of Hong Kong
nanguan@cityu.edu.hk

***Abstract*—In Verilog code design, identifying and locating functional bugs is an important yet challenging task. Existing automatic bug localization methods have limited capabilities; they only suggest a set of potential buggy lines rather than precisely identifying the bug. Moreover, they depend on verification tools like testbenches and reference models, which require expert input and are time-consuming to develop. This paper introduces LiK (Location is Key), an open-source Large Language Model (LLM) to precisely locate functional bugs in Verilog code without the need for expert-written verification tools. LiK is developed from the open-source coding LLM Deepseek-Coder-Lite-Base-16B through a three-step training process: continuous pre-training to enhance foundational knowledge, supervised fine-tuning to learn how to output localization results, and reinforcement learning to reduce output errors. Experiment results demonstrate that LiK achieves superior functional bug localization accuracy, outperforming both the SOTA traditional method Strider, and SOTA closed-source LLMs like GPT-o1-preview and Claude-3.5-Sonnet. Moreover, integrating LiK into the SOTA LLM-based Verilog debugging tool significantly boosts its functional bug fixing success rate from 76.47% to 90.54%. This underscores LiK's potential to enhance the performance of end-to-end automatic Verilog debugging tools.**

I. Introduction

To fix bugs in Verilog code design, the first step is to identify and locate them within the code. Unlike syntax bugs, functional bugs can pass syntax checks and synthesis but cause the circuit to behave unexpectedly, making them much more challenging to detect and pinpoint. Therefore, an effective tool that automatically locates functional bugs would be immensely beneficial to Verilog developers, significantly enhancing their debugging efficiency.

Existing methods [1]–[9] for automatically locating functional bugs in Verilog code have significant limitations. They typically suggest a set of potential buggy lines rather than pinpointing the exact line containing the bug. Additionally, these methods rely on verification tools like testbenches and reference models, which require significant expertise and are time-consuming to develop. These limitations reduce the practical value of existing automated bug localization techniques for Verilog developers.

In this paper, we propose a novel approach to automate Verilog functional bug localization using Large Language Models (LLMs). Our method aims to identify buggy lines precisely by analyzing the code directly, eliminating the need for expert-written verification tools or Electronic Design Automation (EDA) tools. Over the past two years, researchers have extensively investigated the application of LLMs in software debugging [10]–[12]. By leveraging the powerful code comprehension capabilities of LLMs, these approaches have demonstrated remarkable effectiveness in automated bug detection and repair. These promising results motivate us to explore LLM-based approaches for Verilog functional bug localization, an area that remains unexplored in current research.

We introduce LiK (Location is Key), an open-source LLM for *functional* bug localization in Verilog code. LiK is developed from Deepseek-Coder-Lite-Base-16B [13], a widely-used open-source LLM specialized for coding tasks. Our approach includes three steps: (1) continuous pre-training to enhance LiK's foundational knowledge in Verilog bug localization; (2) supervised fine-tuning to teach LiK to output localization results and (3) reinforcement learning [14] to reduce common localization errors. Our experiments show that LiK achieves superior localization accuracy, surpassing both the SOTA non-LLM method Strider [9] and leading closed-source LLMs including GPT-o1-preview and Claude-3.5-Sonnet.

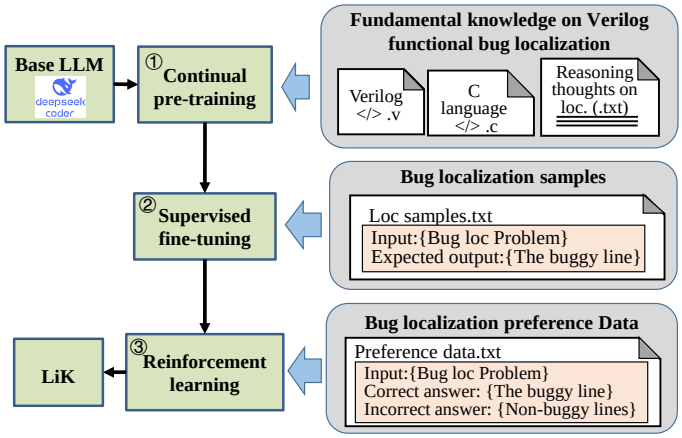


Fig. 1. The training process of LiK, including three steps: ① Unsupervised continuous pre-training to enhance the foundational knowledge in Verilog bug localization; ② Supervised fine-tuning to enable the LLM to output the correct bug location; ③ Reinforcement learning to reduce common errors in localization results.

LiK serves two practical applications: it functions as an IDE add-on to assist developers, and as an enhancement module for auto-verification tools to support debugging. We incorporate LiK into GPT-3.5, GPT-4, and the SOTA LLM-based Verilog debugging system MEIC [15], which significantly improves their bug fixing performance. In particular, MEIC's success rate on functional bug fixing increases from 76.47% to 90.54%. This result demonstrates LiK's potential in strengthening the performance of automatic Verilog debugging systems. Datasets and code of this research will be open-sourced at https://github.com/bingkunyao/Location-is-key.

II. Our Method

We develop LiK from Deepseek-coder-V2-Lite-base-16B [13] (referred to as the base LLM), an open-source LLM that excels in coding tasks. As shown in Fig. 1, we obtain LiK from the base LLM through three training steps:

1) *Continual pre-training*. Since the base LLM lacks specialized knowledge in Verilog bug localization, we perform two rounds of

* Corresponding authors.

unsupervised continual pre-training to enhance its fundamental knowledge in this domain.

2) *Supervised fine-tuning*. We fine-tune the model under the supervision of bug localization samples to enable it to generate bug localization results.
3) *Reinforcement learning*. While supervised fine-tuning teaches the model "what is correct", we leverage reinforcement learning [14] to help it learn "what to avoid" from both correct and incorrect answers, thereby reducing common localization errors.

A. Continual Pre-training

We perform two rounds of unsupervised continual pre-training on the base LLM. The first round enhances the model's fundamental knowledge of Verilog, while the second round focuses on knowledge specific to functional bug localization.

We collect Verilog data for training from three sources: (a) VGen [16] dataset which includes all open-source Verilog repositories from GitHub; (b) OpenCores [17], which is a major open-source hardware design platform, and (c) an internal code dataset acquired from our partnered hardware design companies. Even with our exhaustive data collection efforts, the total data size remains limited (∼253MB). This stems from the closed nature of the hardware design industry, where most Verilog designs are private and inaccessible. Given this data scarcity, we employ the LoRA [18] algorithm for continual pre-training, which updates only a small subset of all the model parameters. This targeted parameter updating approach effectively mitigates the risk of overfitting when conducting continual pre-training with limited data.

1) Round 1: For Basic Knowledge of Verilog

We pre-train the base LLM with fundamental knowledge pertaining to Verilog. Notably, our training corpus includes not only Verilog but also C language. This is motivated by the significant syntactical similarities between them [19]. Although the base LLM already contains C language knowledge from its initial training, additional C language exposure during the pre-training phase can further enhance the model's understanding of these shared syntactical features [20], leading to better comprehension of Verilog.

The Verilog corpus comes from the VGen [16] dataset and consists of two parts. The first part comprises open-source Verilog files from GitHub, where each file contains "module" and "endmodule" statements and has fewer than 20000 characters. The second part consists of sections from 70 Verilog textbooks in e-PDF format, where sections without Verilog codes are filtered out.

For the C language corpus, we use the documentation from the CodeSearchNet [21] dataset. The execution of C is fundamentally different from that of Verilog: C language executes sequentially, while Verilog describes hardware circuits where multiple operations can occur simultaneously. Therefore, we limit the C language content to less than 20% of the total corpus. This helps maintain the LLM's focus on Verilog's hardware description features.

We apply MinHash [22] and LSH [23] algorithms for data deduplication and remove any syntactically incorrect Verilog and C code. Through these preprocessing steps, we refine the final corpus to 91787 entries, where each entry is a code snippet or documentation of Verilog/C language. The size of the final corpus is 206.48MB, with Verilog content comprising 171.03MB and C language materials accounting for 35.45MB.

2) Round 2: For Knowledge Specific to Bug Localization

After enhancing the model's fundamental Verilog knowledge, our next goal is to incorporate specialized knowledge for bug localization.

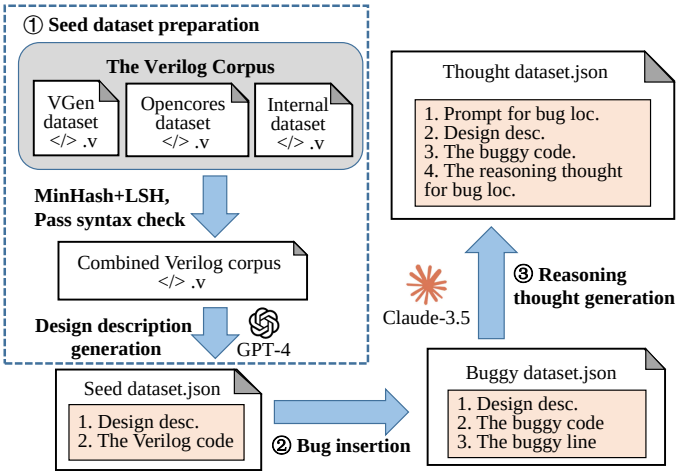


Fig. 2. The process of synthesize the training data including the reasoning thoughts for bug localization.

Research [24] demonstrates that integrating reasoning processes that lead to correct answers into pre-training helps the models better understand the input tasks and generate more relevant responses. Following this principle, we use reasoning thoughts for bug localization as our pre-training corpus.

We formulate each training data as:

$$f \circ d \circ b \circ r, \tag{1}$$

where $\circ$ represents the concatenate operator, f denotes a prompt that instructs the model to locate the buggy line, d represents the design description of the code, b denotes the buggy code, and r denotes the reasoning thought that explains how to locate bugs in b. Fig. 3 illustrates an example of this training data. Since such data is scarce in open-source communities, we develop a synthetic data generation approach, which is shown in Fig. 2: First, we prepare a seed dataset containing paired design descriptions and code. Next, we insert bugs into the code. Finally, we employ Claude-3.5-Sonnet to generate reasoning thoughts for bug localization.

(1) **Seed dataset**. Our seed dataset integrates Verilog code from three sources: VGen [16], OpenCores [17], and an internal code dataset acquired from our partnered hardware design companies. Each dataset entry is a Verilog module, which serves as the basic units for our training data. We remove duplicate modules by the MinHash [22] and LSH [23] algorithms, and include modules that pass syntax checks and contain more than 20 lines of code. For each module, we use GPT-4 to generate the corresponding design description. The final seed dataset comprises 7802 entries, totaling 36.92MB.

(2) **Bug insertion**. Based on our analysis of [25], we insert bugs according to five common functional error patterns in Verilog:

- *Operator errors*: Misuse of operators, e.g., confusing "+" with "-", "&&" with "||".
- *Numerical value errors*: Incorrect numerical values or bit widths. e.g., setting a clock divider ratio to 8 instead of the required 16, declaring a 4-bit signal when 6 bits are required.
- *Keyword errors*: Incorrect use of complementary keywords, e.g., confusing "wire" with "reg", "input" with "output".
- *Variable name errors*: Incorrect variable references, e.g., confusing "counter1" with "counter2".
- *Edge errors*: Incorrect specifications on edge conditions, e.g., confusing a rising edge "posedge" with a falling edge "negedge".

For each data entry in the seed dataset, we select one of the above patterns and use regular expressions to modify a random location in the code, thus obtaining the buggy code and buggy line. Since

The prompt of bug localization *f*:

You are a hardware engineer proficient in Verilog. Please locate the functional bug in the code according to the design description and the code.

Design description *d*:

// Provide the input/output interfaces and functional requirements

Module name: …

Interface Description: Input ports: … Output ports: …

Functional requirements: …

The buggy code *b*:

{The buggy code}

The reasoning thought for bug localization *r*:

Let's analyze the code step-by-step to locate the buggy line.

// Analyze the code from the front to the end

1. Module Interface: It seems that there is an error with the output port. According to the design description, …

2. Internal Registers: The internal registers matches well with design descriptions, …

…

After careful analysis, we conclude that the buggy line is ``output [16 + 46 - 1:0] C; `` *// Provide the buggy line at the end of reasoning thought*

Fig. 3. A training sample for the second round of continual pre-training.

we focus on functional bugs, we retain only those entries where the buggy code is syntactically correct.

(3) **Reasoning thought generation**. For each data entry, we construct a prompt containing the design description *d*, the buggy code *b*, and the buggy line, then feed it to Claude-3.5-Sonnet to generate reasoning thoughts that pinpoint the buggy line in *b*. Our prompt instructs the model to follow a step-by-step reasoning process, assuming the buggy line is unknown at the beginning of the reasoning process. This prevents the model from directly stating the buggy line at the beginning of its reasoning, which aligns with real-world scenarios where developers do not know the bug location when starting to debug. Additionally, some generated thoughts fail to locate the actual buggy line, and manually checking each reasoning thought would be labor-intensive. Therefore, we require the model to conclude the reasoning thought with the specific format "*the buggy line is* {...}.". This formal specification allows us to automatically verify whether the reasoning leads to the correct buggy line.

We randomly select 100 generated thoughts for manual review. We observe that each thought follows a systematic reasoning process similar to human debugging. The reasoning process examines the buggy code from front to end and ultimately pinpoints the buggy line. Fig. 3 illustrates an example of these generated thoughts. After removing thoughts that fail to locate the actual buggy line, we obtain a final dataset of 7622 entries, totaling 46.83MB in size.

B. Supervised Fine-tuning

The base LLM acquires essential knowledge for functional bug localization through continual pre-training. However, due to the unsupervised nature of pre-training, the base LLM cannot directly answer the question "Where is the bug?". To address this issue, we fine-tune the model under the supervision of bug localization samples. Same as continual pre-training, since our training data is quite limited (36.72MB), we employ the LoRA algorithm for supervised fine-tuning to prevent overfitting.

Each training sample is an input-output pair (q, a). The input q is a natural language description of the bug localization problem, including both the design description and the buggy code. For the expected output a, which indicates the bug location, there are two possible formats: the line number (as in existing non-LLM solutions) or the content of the actual buggy line. We choose the latter because LLMs are better at processing natural language text than numerical reasoning, as these are trained primarily using textual data. Fig. 4 illustrates an example of such a training sample.

We apply the same seed dataset and method as in Sec. II-A2 to generate the training data for supervised fine-tuning, producing 7622 data samples with a total size of 36.72MB.

Input *q*:

You are a hardware engineer proficient in Verilog. Below I give you a hardware design description and the corresponding code implementation with the buggy line. Your task is to identify this buggy line.

Hint: the buggy line may not conform to the design description, or has logic/functional error.

Requirements: You should only output the SPECIFIC buggy line identified.

// The output is the specific content of the buggy line

Design description: {Design description}

Code implementation: {Buggy code}

Expected output *a*:

always @ (negedge start) begin *//The content of the buggy line*

Fig. 4. A training sample for supervised fine-tuning.

While recent literature [24] suggests that prompting LLMs to explain their reasoning before providing answers can improve accuracy, we only use the buggy line content as the expected output of the model, without including the reasoning thoughts for bug localization. This is because reasoning thoughts may introduce noise during the supervised fine-tuning, leading to a decrease in localization accuracy. We discuss this issue in detail in Sec. IV-B.

C. Reinforcement Learning

Supervised fine-tuning trains the model using correct bug localizations but does not actively discourage incorrect ones. To overcome this limitation, we employ reinforcement learning (RL) techniques [14] to let the model learn from both correct and incorrect answers. This helps avoid common localization errors, thereby further enhancing the model's overall performance.

We implement SimPO algorithm [26] for RL training. SimPO directly aligns the reward with the model's generation process. This alignment encourages outputs matching correct answers while discouraging those that resemble incorrect ones.

During training, SimPO optimizes model parameters through gradient descent by maximizing an objective function. This function measures the reward difference between correct and incorrect answers, thereby enhancing the alignment between model outputs and correct bug locations. The following sections describe our training data, reward function, and objective function in detail.

1) Training Data: Correct and Incorrect Answers

Each training sample is a triple (q, a_w, a_l), where q represents a bug localization problem, a_w denotes the correct answer (i.e., the actual buggy line), and a_l represents the incorrect answer, which is a non-buggy line that the model should avoid. To construct these training samples, we first generate (q, a_w) pairs using the approach and seed dataset described in Sec. II-A2. To identify a_l for each pair, we repeatedly feed q to the model, then select the non-buggy line that the model misidentifies most frequently. Fig. 5 illustrates an example of such a training sample.

We initially generate 7622 (q, a_w) pairs. For 4080 of these pairs, the model consistently returns the actual buggy line, making it impossible to obtain a suitable a_l. After removing these pairs, our final dataset includes 3542 samples with a total size of 19.82MB.

2) The Reward and Objective Functions of SimPO

SimPO aligns its reward function with the LLM's generation process. Let π_θ denote our model. Given input problem q, the length-normalized probability of the model generating an answer a is:

$$p_\theta(a \mid q) = \frac{1}{|a|} \log \pi_\theta(a \mid q) = \frac{1}{|a|} \sum_{i=1}^{|a|} \log \pi_\theta(a_i \mid q, a_{<i}) \quad (2)$$

Bug localization problem q:
You are a hardware engineer proficient in Verilog. Below I give you a hardware design description and the corresponding code implementation with the buggy line. Your task is to identify this buggy line.
Hint: the buggy line may not conform to the design description, or has logic/functional error.
Requirements: You should only output the SPECIFIC buggy line identified.
Design description: {Design description}
Code implementation: {Buggy code}
Correct answer a_w:
always @ (negedge start) begin // *The actual buggy line*
Incorrect answer a_l:
if (cnt == 3) // *The non-buggy line that is most frequently misidentified as buggy by the model*

Fig. 5. A training sample for reinforcement learning.

where $\log \pi_\theta(a_i \mid q, a_{<i})$ [1] denotes the probability of generating the i-th token a_i given the input problem q and the first i-1 tokens of a.

For input problem q and answer a, SimPO defines the reward as $\beta \times p_\theta(y \mid x)$, where β is a constant that controls the reward gap between different answers.

SimPO then applies the Bradley-Terry objective function [28] to compute the difference between the rewards of a_w and a_l. To enhance a_w's advantage over a_l, SimPO adds a difference term γ to ensure a_w's reward exceeds a_l's by at least γ. The resulting objective function is:

$$\mathcal{L}(\pi_\theta) = \mathbb{E}\left[\log \sigma\left(\beta \times \left(p_\theta(a_w \mid q) - p_\theta(a_l \mid q)\right) - \gamma\right)\right] \quad (3)$$

SimPO directly employs generation probabilities to define reward and objective functions. By maximizing the objective function, the model is more likely to produce the preferred output a_w instead of a_l. Since a_l is the non-buggy line that the model tends to misidentify, this approach reduces common localization errors.

III. Evaluation

A. Experimental Settings

1) Training Setups

We train LiK using 8 A800-80G GPUs. We use cosine annealing to adjust learning rate during the training process, with 10% of total steps allocated for warmup. The continual pre-training, supervised fine-tuning, and reinforcement learning take approximately 1, 3, and 20 hours respectively.

2) Evaluation Metrics

We apply pass@k, a standard metric for evaluating language models' performance on coding-related tasks [29]. It measures the probability of finding at least one correct answer among k repeated generations from models:

$$pass@k := \mathbb{E}\left[1 - \frac{\binom{n-c}{k}}{\binom{n}{k}}\right] \quad (4)$$

where n denotes the number of generated outputs for each test case. Following prior work on LLM-based Verilog coding [30]–[33], we set n to 20 to obtain reliable performance estimates. We evaluate both pass@1 and pass@5 metrics, where pass@1 measures the model's accuracy with single generations, and pass@5 assesses the performance when generating multiple solutions.

[1]This value is generally calculated through forward propagation in the transformer model. For details, please refer to [27].

3) Baselines

We compare LiK with both LLM-based and non-LLM-based Verilog functional bug localization techniques:

- **LLM baselines**. Since no prior work has applied LLMs to Verilog functional bug localization, we establish two baseline categories: (a) Commercial LLMs, including GPT-series (GPT-3.5, GPT-4 and GPT-o1-preview) and Claude-3.5-Sonnet; (b) LLMs specially designed for tasks related to Verilog coding, including Thakur et al. [34], RTLCoder [30] and VeriSeek [33]. In our experiments, although we explicitly instruct the model to pinpoint the specific buggy line, these LLMs sometimes respond with complete code blocks, and their output formats vary significantly across different responses. Given this inconsistency, we use a relaxed evaluation criterion: we consider a response to be correct as long as the actual buggy line appears in the response. This yields more optimistic results than the actual performance.
- **The traditional non-LLM method**. We evaluate Strider [9], which is the SOTA Verilog functional bug localization method. Strider identifies a list of possible buggy lines by comparing the waveform outputs between the buggy and golden implementations. Following the same evaluation method as LLM baselines, we consider Strider's output to be correct if its output list includes the actual buggy line.

For LiK, we apply a stricter evaluation criterion: its output must exactly match the actual buggy line. We mark the output as incorrect if it includes any additional lines.

4) Test Dataset

Currently, there is no benchmark for Verilog functional bug localization using LLMs. Test datasets employed in existing works for Verilog bug localization do not have code descriptions, making them unsuitable for our problem where natural language understanding is essential. Therefore, we create a test dataset based on RTLLM [35], a benchmark widely used for evaluating LLMs on Verilog coding tasks. RTLLM consists of 29 typical hardware module designs, where each design includes:

- A design description crafted by experienced hardware engineers. It specifies only the functional requirements and interfaces, without revealing any internal implementation details.
- A golden Verilog implementation (35-175 lines of code).
- A testbench that includes input stimuli and expected outputs.

To create our test dataset, we systematically introduce bugs into the golden code according to the common Verilog error types discussed in Sec. II-A2. Each buggy code maintains correct syntax but fails the testbench verification. The final dataset contains 102 test cases, with bug types distributed as follows:

Bug type	Operators	Numerical value	Variable name	Keyword	Edge
Number	25	26	24	13	14

B. Experimental Results

1) Localization Accuracy

Our experimental results demonstrate the effectiveness of LiK in locating functional bugs in Verilog code. As shown in Table I, the base model exhibits limited performance. This is because the base model's training corpus lacks sufficient data related to Verilog programming and bug localization. Through our training on specialized datasets, LiK extends the base model and demonstrates strong capability in locating functional bugs in Verilog code.

TABLE I
RESULTS ON LOCALIZATION ACCURACY.

Type	Method	pass@1(%)	pass@5(%)
LLM	Thakur et al.,	10.44	20.29
	RTLCoder	17.55	19.12
	Veriseek	16.32	23.84
	GPT-3.5	40.25	50.87
	GPT-4	77.64	85.86
	GPT-o1-preview	78.08	91.83
	Claude-3.5	90.8	93.11
non-LLM	Strider	58.8	58.8
Base Model	Deepseek-Coder-V2-Lite-16B	13.43	19.59
Ours	LiK	**93.33**	**94.10**

Table I shows that LiK outperforms all LLM baselines in our experiments. Note that as mentioned in Sec. III-A3, the reported performance of LLM baselines is higher than the actual, as their responses may coincidentally include the buggy line rather than identifying it through accurate localization. Additionally, Verilog-specific LLMs demonstrate relatively poor performance. This is because they are primarily trained for code generation tasks rather than bug localization. Their training datasets mainly consist of hardware design specifications and their corresponding Verilog implementations, with few content related to bug localization.

LiK demonstrates superior localization accuracy compared to Strider. Additionally, we observe that Strider's output contains 6.67 lines for each test case on average. This multi-line output is inherent to its methodology: it first identifies mismatched output signals, then traces through the code's abstract syntax tree to find all statements that might affect these signals. Furthermore, Strider requires expert-written testbenches to generate waveform files for bug localization. In comparison, LiK directly locates bugs without the need for verification tools like testbenches, reference models, or other EDA tools. This makes LiK more efficient for bug localization.

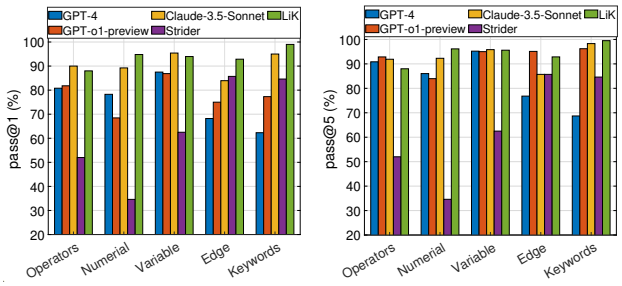
2) Ablation Studies

Table II demonstrates LiK's performance improvements across different training phases. Here, LiK$_{PT_{1_w/oC}}$ represents the model pre-trained using only Verilog corpus without C corpus and reasoning thoughts. LiK$_{PT_1}$ and LiK$_{PT_{12}}$ denote the model after the first and second round of continual pre-training, and LiK$_{PT_{12}+SFT}$ and LiK$_{PT_{12}+SFT+RL}$ represent the model after supervised fine-tuning and reinforcement learning, respectively.

TABLE II
ABLATION STUDY OF LIK'S TRAINING STAGES

Model	**pass@1(%)**	**pass@5(%)**
LiK$_{PT_{1_w/oC}}$	16.63	24.31
LiK$_{PT_1}$	18.97	27.01
LiK$_{PT_{12}}$	29.15	38.76
LiK$_{PT_{12}+SFT}$	87.54	90.83
LiK$_{PT_{12}+SFT+RL}$	**93.33**	**94.10**

LiK pre-trained with only Verilog corpus achieves a modest performance. Adding C corpus brings slight improvements, with pass@1 reaching 18.97%. The incorporation of bug localization reasoning thoughts further enhances the model's performance to 29.15% pass@1, which indicates enhanced understanding of bug localization problems. However, these performances remain relatively low. This weakness stems from the unsupervised nature of continual pre-training, during which the model has not explicitly learned to locate buggy lines.



(a) Pass@1. (b) Pass@5.

Fig. 6. The performance on different types of bugs.

Supervised fine-tuning leads to a substantial performance increase, with pass@1 accuracy reaching 87.54%. This improvement is attributed to the supervised nature of this training phase, which directly aligns the model's outputs with bug localization objectives.

The reinforcement learning phase further improves the model's performance, pushing the pass@1 metric to 93.33%. This phase helps the model avoid common errors and prioritize correct bug localization. We further discuss its impact in Sec. IV-A.

3) Results with Different Bug Types

This part examines the performance in locating different types of bugs. As shown in Fig. 6(a) and 6(b), LiK demonstrates superior performance compared to all baselines in locating bugs related to numerical values, keywords, and edge conditions. For bugs involving variable names and operators, Claude-3.5-Sonnet achieves better results than LiK. Compared to GPT series, LiK shows higher pass@1 but slightly lower pass@5 in these categories.

The performance variation can be explained by the inherent characteristics of different bug types. Edge-related and keyword-related bugs usually directly conflict with design descriptions. On the other hand, bugs related to numerical values, operators, and variable names are often embedded in the internal implementation logic, thus requiring more complex reasoning to be localized.

Two main factors limit LiK's ability in complex reasoning tasks: its smaller model size and insufficient training data. Despite our best efforts to collect data, due to the closed nature of the hardware design community, we are only able to obtain ∼300MB of training data. Future work will focus on expanding the training dataset to improve model performance.

Notably, all LLMs including ours consistently outperform Strider, the SOTA non-LLM method, across all bug categories. This demonstrates the effectiveness of LLMs in Verilog bug localization.

4) Can LiK benefit LLM-based Verilog debugging?

With the rapid development of LLMs, leveraging them for automated Verilog debugging has emerged as a promising direction [15], [31], [36], [37]. The ultimate goal of bug localization is to facilitate the automated debugging processes. To demonstrate how LiK can benefit LLM-based Verilog debugging systems, we conducted experiments combining LiK's results with three baseline debugging methods using LLMs: GPT-3.5-based, GPT-4-based, and the SOTA LLM-based Verilog debugging system MEIC [15].

For two GPT-based methods, we input the design description and buggy code, and ask GPT to fix functional bugs. MEIC implements an end-to-end debugging framework with iterative cycles, during which two fine-tuned LLMs work in tandem: one repairs the buggy code based on compiler/simulator feedback, while the other scores the repairs to guide subsequent iterations. This process runs until the code passes testbench verification or reaches 10 iterations.

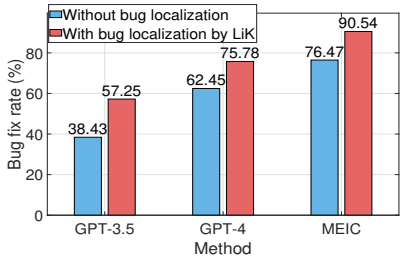


Fig. 7. The success rate of bug fixing after incorporating the result of LiK.

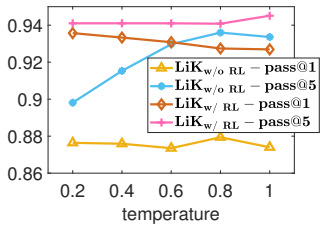


Fig. 8. Performance vs temperature.

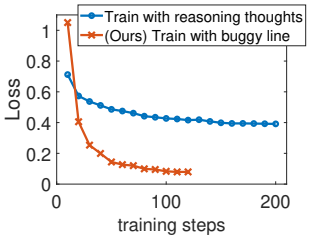


Fig. 9. The training loss with different types of data.

To assess the impact of LiK, we add two elements at the end of the LLM prompts for baseline methods: the buggy lines identified by LiK, and a clear statement indicating these are buggy lines. For MEIC, we add this information to the prompt of the LLM responsible for code repairing. We conduct 20 repeated experiments on our test dataset and measure the bug-fixing success rate. A bug fix is considered successful only when the repaired code passes all the test cases in its testbench.

As shown in Fig. 7, the integration of LiK's bug localization significantly improves the performance of all baselines. Notably, MEIC's bug-fixing success rate is increased by 14%. Also, we observe that LiK+MEIC achieves 99.35% success rate on the test cases that MEIC could handle alone. These results highlight LiK's potential in enhancing end-to-end automatic Verilog debugging systems.

IV. Discussions

A. The Impact of Reinforcement Learning

To further analyze the impact of the reinforcement learning phase, we examine LiK's performance before and after it across different temperature values. Temperature is a parameter ranging from 0 to 1 that controls the randomness of model's output, in which higher temperature leads to more diverse outputs.

As shown in Fig. 8, before reinforcement learning, LiK's pass@5 varies significantly across temperatures and is much higher than its pass@1. This gap indicates the output instability for identical test cases, leading to errors in bug localization. After reinforcement learning, the gap between pass@5 and pass@1 becomes smaller, and pass@5 becomes more stable with changes in temperature. Moreover, both pass@1 and pass@5 increase across all temperature settings. This shows that by utilizing incorrect bug localization as negative examples, reinforcement learning reduces errors caused by output instability, thereby improving bug localization accuracy.

B. Why not Using Reasoning Thoughts for Supervised Fine-tuning?

Previous research [24] shows that prompting LLMs to generate reasoning thoughts rather than directly providing the answers can improve accuracy. This suggests that LiK might benefit from generating reasoning thoughts for bug localization. For this to work, during the supervised fine-tuning, we need to use reasoning thoughts as expected outputs, rather than just the buggy line.

To test this hypothesis, we compare the performance of two models: one trained with bug localization reasoning thoughts and another with only buggy lines (our current approach). We generate reasoning thoughts for bug localization using the method and seed dataset in Sec. II-A2, and use these thoughts as training data for the reasoning-based model. We evaluate both models on the test set used in Sec. III. Results show that the model trained with reasoning thoughts achieves pass@1 and pass@5 scores of 73.48% and 77.69%, while our current model trained with only buggy lines performs better, reaching 87.54% and 90.83%.

To understand this performance gap, we analyze the training loss of both models. As shown in Fig. 9, the model trained with reasoning thoughts has higher training loss. This suggests that reasoning thoughts introduce additional noise into the training process. Intuitively, reasoning thoughts are not our target output - they are intermediate steps towards the bug localization. The extra complexity from reasoning thoughts makes it harder for the model to focus on the main task of bug localization. Therefore, we use only buggy lines as the expected output during supervised fine-tuning.

V. Related Works

Several non-LLM approaches [1]–[9] have been proposed for Verilog bug localization. They typically provide a set of potential buggy lines rather than pinpointing the exact bug location. Moreover, they rely on verification tools that require expertise to develop.

No prior research has explored the use of LLMs for Verilog bug localization. Other related works can be categorized into two groups:

(1) The first category [16], [30], [33], [34], [38] proposes LLMs specifically for Verilog. However, their primary focus is on code generation, with their training data mainly consisting of design documents and corresponding implementations, containing little content on Verilog bug localization. Therefore, they are not suitable for the bug localization problem.

(2) The second category [15], [31], [36], [37] develops LLM-based frameworks for Verilog debugging, which includes several LLMs as agents and each agent completes one intermediate step. However, none of these frameworks includes bug localization. Our experiments demonstrate that integrating LiK into the SOTA LLM-based functional bug repairing framework MEIC [15] significantly improves its performance. This highlights the importance of bug localization in LLM-based debugging workflows.

VI. Conclusion

This paper proposes LiK, an LLM for locating the functional bugs in Verilog code. LiK is trained through two rounds of continuous pre-training, supervised fine-tuning, and reinforcement learning. Experimental results show that LiK achieves high localization accuracy. By integrating LiK's results, the performance of the SOTA LLM-based Verilog debugging system is significantly improved. This shows LiK's potential to support auto-verification tools for Verilog debugging.

As for the future work, we plan to enhance LiK by leveraging additional expert knowledge and EDA tool feedback, and generating more training data specialized for bug localization. We will also extend LiK for more complex hardware designs and other hardware description languages such as VHDL, SystemVerilog, and Chisel.

Acknowledgement

We would like to thank the anonymous reviewers for their helpful feedback. This work is supported by National Key Research and Development Program (Grant No.2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grants No. BK20243042, BG2024010), the Start-up Research Fund of Southeast University (Grant No. RF1028624005), the National Natural Science Foundation of China (Grants No. U22A2025 and 92464301).

REFERENCES

[1] J. Wu, Z. Zhang, D. Yang, X. Meng, J. He, X. Mao, and Y. Lei, "Fault localization for hardware design code with time-aware program spectrum," in *IEEE ICCD 2022*, pp. 537–544.

[2] Z. Zhang, Y. Li, L. Xia, J. Xue, J. Wu, and X. Mao, "An effective fault localization approach for verilog based on enhanced contexts," *Frontiers of Computer Science*, vol. 18, 06 2024.

[3] D. Pal and S. Vasudevan, "Symptomatic bug localization for functional debug of hardware designs," in *IEEE VLSID 2016*, pp. 517–522.

[4] B. Kumar, V. S. Vineesh, P. Nemade, and M. Fujita, "Aries: A semi-formal technique for fine-grained bug localization in hardware designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5709–5721, 2022.

[5] S. Srinivasan and R. Vemuri, "Model checking leveraged error localization for complex rtl designs," in *IEEE ICCD 2022*, pp. 585–592.

[6] P. Santiesteban, Y. Huang, W. Weimer, and H. Ahmad, "Cirfix: Automated hardware repair and its real-world applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 7, pp. 3736–3752, 2023.

[7] J. Wu, Z. Zhang, D. Yang, J. Xu, J. He, and X. Mao, "Knowledge-augmented mutation-based bug localization for hardware design code," *ACM Trans. Archit. Code Optim.*, apr 2024, just Accepted.

[8] G. Stracquadanio, S. Medya, S. Quer, and D. Pal, "Veribug: An attention-based framework for bug localization in hardware designs," in *IEEE/ACM DATE 2024*, pp. 1–2.

[9] D. Yang, J. He, X. Mao, T. Li, Y. Lei, X. Yi, and J. Wu, "Strider: Signal value transition-guided defect repair for hdl programming assignments," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 5, pp. 1594–1607, 2024.

[10] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, and Q. Wang, "Software testing with large language models: Survey, landscape, and vision," *IEEE Transactions on Software Engineering*, vol. 50, no. 4, pp. 911–936, 2024.

[11] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *IEEE S&P 2023*, pp. 2339–2356.

[12] S. B. Hossain, N. Jiang, Q. Zhou, X. Li, W.-H. Chiang, Y. Lyu, H. Nguyen, and O. Tripp, "A deep dive into large language models for automated bug localization and repair," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, 2024.

[13] DeepSeek-AI, Q. Zhu, and D. G. et al., "Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence," 2024. [Online]. Available: https://arxiv.org/abs/2406.11931

[14] A. J. et al., Yuntao Bai, "Training a helpful and harmless assistant with reinforcement learning from human feedback," 2022. [Online]. Available: https://arxiv.org/abs/2204.05862

[15] K. Xu, J. Sun, Y. Hu, X. Fang, W. Shan, X. Wang, and Z. Jiang, "Meic: Re-thinking rtl debug automation using llms," in *IEEE ICCAD 2024*, to appear. [Online]. Available: https://arxiv.org/abs/2405.06840

[16] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 29, no. 3, 2024.

[17] "Opencores." [Online]. Available: https://opencores.org

[18] E. J. Hu, yelong shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models," in *ICLR 2022*. [Online]. Available: https://openreview.net/forum?id=nZeVKeeFYf9

[19] Z. Pei, H. Zhen, M. Yuan, Y. Huang, and B. Yu, "BetterV: Controlled verilog generation with discriminative guidance," in *ACM ICML 2024*, vol. 235, pp. 40 145–40 153.

[20] K. Gupta, B. Thérien, A. Ibrahim, M. L. Richter, Q. Anthony, E. Belilovsky, I. Rish, and T. Lesort, "Continual pre-training of large language models: How to (re)warm your model?" 2023. [Online]. Available: https://arxiv.org/abs/2308.04014

[21] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, and M. Brockschmidt, "Codesearchnet challenge: Evaluating the state of semantic code search," 2020. [Online]. Available: https://arxiv.org/abs/1909.09436

[22] A. Broder, "On the resemblance and containment of documents," in *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No.97TB100171)*, 1997, pp. 21–29.

[23] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev, "A survey on locality sensitive hashing algorithms and their applications," 2021. [Online]. Available: https://arxiv.org/abs/2102.08942

[24] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *NeurIPS 2022*, vol. 35, pp. 24 824–24 837.

[25] S. Sudakrishnan, J. Madhavan, E. J. Whitehead, and J. Renau, "Understanding bug fix patterns in verilog," in *IEEE/ACM MSR 2008*, p. 39–42.

[26] Y. Meng, M. Xia, and D. Chen, "Simpo: Simple preference optimization with a reference-free reward," 2024. [Online]. Available: https://arxiv.org/abs/2405.14734

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[28] "Bradley–terry model." [Online]. Available: https://en.wikipedia.org/wiki/Bradley%E2%80%93Terry_model

[29] M. Chen and J. T. et al., "Evaluating large language models trained on code," 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[30] S. Liu, W. Fang, Y. Lu, J. Wang, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[31] Y.-D. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language models," 2024. [Online]. Available: https://arxiv.org/abs/2311.16543

[32] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Invited paper: Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–8.

[33] W. Ning, Y. Bingkun, Z. Jie, W. Xi, J. Zhe, and G. Nan, "Large language model for verilog generation with golden code feedback," 2024. [Online]. Available: https://arxiv.org/abs/2407.18271

[34] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in *IEEE/ACM DATE 2023*, pp. 1–6.

[35] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *IEEE/ACM ASP-DAC 2024*, p. 722–727.

[36] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards llm-powered verilog rtl assistant: Self-verification and self-correction," 2024. [Online]. Available: https://arxiv.org/abs/2406.00115

[37] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," 2024. [Online]. Available: https://arxiv.org/abs/2403.11671

[38] W. Fu, S. Li, Y. Zhao, H. Ma, R. Dutta, X. Zhang, K. Yang, Y. Jin, and X. Guo, "Hardware phi-1.5b: A large language model encodes hardware domain specific knowledge," in *IEEE/ACM ASP-DAC 2024*, pp. 349–354.

DSPlacer: DSP Placement for FPGA-based CNN Accelerator

Baohui Xie[1†], Xinrui Zhu[1†], Zhiyuan Lu[1], Yuan Pu[2], Tongkai Wu[1], Xiaofeng Zou[3], Bei Yu[2], Tinghuan Chen[1*]
[1]Chinese University of Hong Kong, Shenzhen [2]Chinese University of Hong Kong
[3]Shandong Yunhai Guochuang Cloud Computing Equipment Industry Innovation Co., Ltd.

***Abstract*—Deploying convolutional neural networks (CNNs) on hardware platforms like Field Programmable Gate Arrays (FPGAs) has garnered significant attention due to their inherent flexibility and parallelism. Achieving optimal timing closure remains a critical challenge, as placement directly impacts clock frequency and throughput. Existing approaches often face scalability issues with large designs or fail to formalize placement rules into automated algorithms. In this paper, we propose DSPlacer, a novel DSP placement framework designed for diverse CNN accelerator architectures in the context of FPGA design. The proposed approach iteratively optimizes the placement of datapath DSPs to enhance timing performance. To achieve this, DSPlacer integrates several advanced techniques, including graph convolutional network-based datapath DSP identification, DSP graph construction, min-cost-flow DSP assignment, and integer linear programming (ILP)-based cascade constraint legalization. These techniques collectively address two key requirements for datapath DSP placement: (1) cascading datapath DSPs to achieve a compact layout, and (2) preserving direct datapath information between the processing system and programmable logic. The framework has been evaluated on multiple academic benchmarks and compared against AMD Xilinx Vivado 2020.2 and AMF-Placer 2.0. Experimental results demonstrate that DSPlacer improves Worst Negative Slack (WNS) by 32% and 65%, respectively, highlighting its efficacy and superiority.**

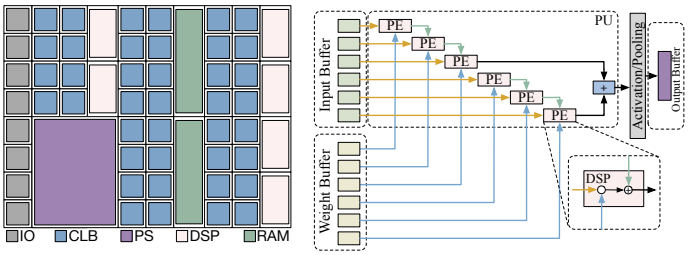
I. Introduction

Convolutional Neural Networks (CNNs) have succeeded in various computer vision applications. Field Programmable Gate Array (FPGA) is a promising hardware platform to deploy the CNN model due to its reconfigurability [1]–[3]. Besides, in modern FPGA, processing systems (PSs) such as CPU and programmable logics (PLs) are integrated into a device to further facilitate CNN deployment. In order to deploy the CNN model on FPGA, logic synthesis, placement, and routing need to be performed by taking a design described by a hardware description language (HDL) as an input. Logic synthesis converts an HDL of design into an optimized gate-level netlist. After logic synthesis, the netlist consists of heterogeneous components, such as lookup tables (LUTs), flip-flops (FFs), digital signal processors (DSPs), random access memories (RAMs), and I/O pads. Placement maps all heterogeneous components onto the FPGA and optimizes some metrics, such as wirelength. Finally, routing determines the precise wiring layout needed to connect the placed components.

Placement plays an important role since the location mapping of components highly affects the final timing performance. In FPGA design, the fixed positions of logic resources, as shown in Fig. 1(a), constrain the mapping process, making it a highly complex combinatorial optimization problem. Solving this problem is exactly NP-hard due to the exponential number of possible mappings required to meet timing and other design constraints.

Typical FPGA placement algorithms can be categorized into simulated annealing and analytical methods. The simulated annealing-based FPGA placement algorithms might lead to long placement runtime when the input netlist is large [4]. In contrast, analytical-based methods can achieve high scalability and quality [5]–[11]. However, these analytical placement methods often result in suboptimal placements, leading to detours routing, especially when applied to CNN accelerators with wider bit widths.

†Equal contribution. *Corresponding author.



(a) (b)

Fig. 1 (a) Xilinx UltraScale+ FPGA layout; (b) CNN architecture.

An intuitive method is using the datapath information in the placement stage to alleviate detours [12]–[18]. Different from ASIC, FPGA has a column-wise heterogeneous resource distribution, which brings a challenge to datapath-driven placement. Previously, Koch *et al* cast the placement of all components on regular datapath as a 0-1 integer linear program (ILP) problem [19], [20]. However, this method has an unacceptable running time when placing all heterogeneous datapath components of the CNN accelerator on the modern FPGA. To reduce running time, Kong *et al* propose to place processing engine (PE) arrays instead of all datapath components by the ILP formulation. Despite this effort, the empirically hierarchical design partitioning causes timing degradation, compared with the FPGA commercial tool [21]. Moreover, while these approaches incorporate regularity into the placement process, they fail to account for datapath-specific information, which is crucial for optimizing CNN accelerators.

In the CNN accelerator, the datapath primarily consists of regular processing units (PU) responsible for computations, as shown in Fig. 1(b). Each PU is composed of multiple PE arrays, which are primarily implemented using DSPs [22], [23]. Therefore, leveraging the DSP interconnection topology as a datapath provides a natural avenue for integrating it into FPGA placement. In [24], graph clustering forms graph cliques, minimizing the width and height of the smallest rectangle and bounding all DSPs. However, this approach struggles to scale for large designs, such as CNN accelerators. Similarly, in [25], empirical rules are proposed for DSP placement, such as assigning PEs to DSP columns within a region. However, these rules are not formalized into an automated algorithm. As computational demands increase and design scales expand, traditional computational architectures are insufficient. To overcome these challenges, advanced architecture, such as systolic array, have gained attention, especially in the CNN accelerator domain. The state-of-the-art systolic array placement method, R-SAD [26], introduces a dedicated analytical algorithm to exploit systolic array regularity fully, achieving significant wirelength reduction. However, its specialized nature limits its applicability to CNN accelerators with more diverse architectures.

In this paper, we present a datapath-driven DSP placement framework designed to support various FPGA-based CNN accelerator architectures. To enhance timing performance, the framework focuses on datapath DSP extraction and datapath-driven DSP placement. The

key contributions are summarized as follows:

- We introduce DSPlacer, a framework that iteratively enhances datapath DSP placement by cooperating with state-of-the-art placers.
- We propose a novel placement methodology incorporating graph convolution networks (GCNs) for datapath DSP identification, datapath graph construction, min-cost-flow DSP assignment, and ILP-based cascade constraint legalization. This approach cascades datapath DSPs to achieve a compact layout while preserving direct datapath connections between the processing system and programmable logic.
- We evaluate DSPlacer on multiple academic benchmarks, comparing it with AMD Xilinx Vivado 2020.2 and AMF-Placer 2.0. Experimental results show that DSPlacer improves Worst Negative Slack (WNS) by 32% and 65%, respectively, demonstrating its effectiveness and superiority.

II. Preliminaries

A. FPGA Architecture

This paper focuses on the prominent Xilinx UltraScale+ architecture [27] for heterogeneous FPGAs, having a column-wise heterogeneous resource distribution. As shown in Fig. 1(a), the configurable logic blocks (CLBs), digital signal processors (DSPs), and block random-access-memories (RAMs) and IO components are variably arranged across the FPGA fabric. Besides, PS, such as the CPU, is fixed and placed at the bottom left of the device.

B. Problem Formulation

Our DSPlacer takes a pre-implementation netlist and DSP specifications of a target FPGA as inputs. A pre-implementation netlist defines the connections among various logic components, while the DSP specifications indicate the available DSP locations on the FPGA. Using our datapath DSP placement results as constraints, off-the-shelf FPGA PnR tools are used to perform other component placement and routing. We aim to maximize timing performance while ensuring all component placements meet design rules. Our problem formulation is formally defined as follows:

Problem 1 (DSP placement for FPGA-based CNN accelerator)**.** *Given a pre-implementation netlist of the CNN accelerator and DSP specifications, place all datapath DSPs in legal locations to maximize timing performance after placing all other components and routing all nets.*

C. Overview

To address Problem 1, we propose DSPlacer, a DSP placement framework tailored for FPGA-based CNN accelerators. Our DSPlacer overall flow is shown in Fig. 2 and comprises two main stages: datapath DSP extraction and datapath DSP placement, with our key contributions highlighted in purple and green.

We take a pre-implementation netlist of the CNN accelerator and DSP specifications as inputs. Firstly, we initialize the placement of all components using the off-the-shelf FPGA placement tool to generate an initial placement solution. Meanwhile, in the datapath DSP extraction stage, a pre-implementation netlist is represented as a graph, and a GCN are used to distinguish datapath DSPs from all DSP components by embedding features for each node. Then, a graph construction procedure is developed to transfer a pre-implementation netlist to a datapath DSP graph, which only involves DSP components with their potential datapath. In the datapath-driven DSP placement stage, datapath DSPs and other components are placed alternatively and iteratively. In particular, the placement of datapath

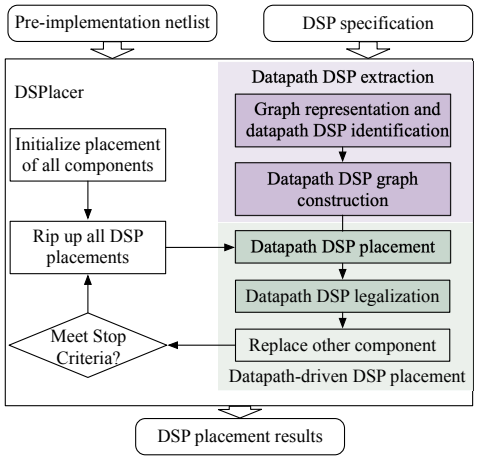
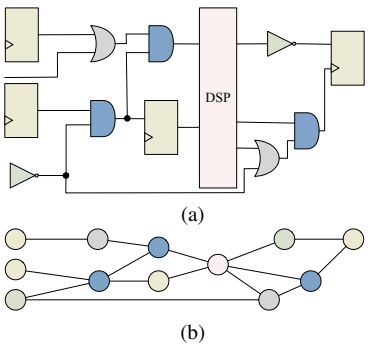


Fig. 2 Our DSPlacer overall flow.



Architecture	
Input Layer	[x,32]
Hidden Layer	[32,32]
MLP Layer	[32,32]
Output Layer	[32,2]
Activation	ReLU
Classification	Softmax

Training Setting	
Optimizer	Adam
Learning Rate	0.001
Dropout	0.35

(c)

Fig. 3 (a) Netlist; (b) Graph representation; (c) GCN configurations.

DSPs is first formulated as a 0-1 quadratic programming problem. To reduce computational complexity, this formulation is further refined into an MCF problem, where the unimodular property of the MCF model guarantees integer solutions. Following this, to strictly enforce cascade constraints, DSPs are legalized both inter-column and intra-column using ILP. Once the DSP placement is finalized, DSPlacer outputs the DSP placement results. Using our output DSP placement results as constraints, the off-the-shelf FPGA PnR tool iteratively places other components and performs routing to generate the final layout. According to this flow as shown in Fig. 2, our DSPlacer can account for datapath-specific information, resulting in improved timing performance for FPGA-based CNN accelerators.

III. Datapath DSP Extraction

This section details our datapath DSP extraction process, which consists of two main steps: (1) graph representation and datapath DSP identification and (2) datapath DSP graph construction. Given a pre-implementation netlist, a well-trained GCN model is adopted to identify datapath DSP, and a DSP graph guides the assignment step.

A. Graph representation and datapath DSP identification

A pre-implementation netlist can be represented as a graph $\mathbb{G} = (\mathcal{V}, \mathcal{E})$, as shown in Figs. 3(a) and 3(b). Let the node set $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ represents n components in the netlist, and the edge set $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ represent m connections. Typically, there are heterogeneous components in a pre-implementation netlist.

To identify the datapath DSP in a netlist, we formulate this task as a node classification problem handled by the GCN framework. Starting

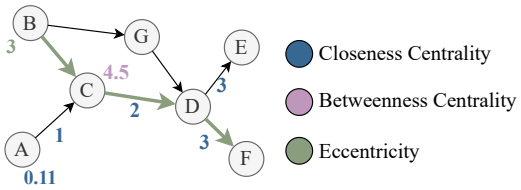


Fig. 4 1. Closeness centrality of A is the reciprocal of the sum of 1, 2, 3, and 3. 2. Betweenness centrality of C is calculated as $1+1+1+0.5+0.5+0.5=4.5$. 3. Eccentricity of B equals the distance from B to F.

from the pre-implementation netlist, we conduct network analysis to extract node features, including centrality metrics and connectivity properties. The dataset is then split into training and test sets, and the trained model is employed to identify datapath DSP nodes in unseen netlists. Specifically, the model consists of two graph convolutional layers, each with 32 hidden units, followed by three fully connected layers with softmax activation. Dropout regularization is applied to mitigate overfitting. The detailed configuration of the GNN model and training setup is presented in Fig. 3(c). To address the class imbalance in the dataset, a weighted loss function is utilized, assigning higher penalties to minority class misclassifications based on class ratios.

Node features play a critical role in classification accuracy. In PADE [28], datapath regularity is exploited by extracting graph automorphism features to classify datapaths. However, while this method identifies local regularities, it struggles to capture global graph properties, which are critical for understanding datapath connectivity and overall functionality. To address this limitation, our approach assigns each node a feature vector $\boldsymbol{x}$ that captures both local and global graph properties. Specifically, the feature vector includes: (a) closeness centrality, (b) feedback loops, (c) eccentricity, (d) indegree, (e) outdegree, (f) betweenness centrality, and (g) the average shortest path distances to other DSP nodes. While most features apply to all nodes, the average shortest path distance is unique to DSP nodes, representing the mean distance to other DSPs. Indegree and outdegree quantify the number of incoming and outgoing neighbors, while feedback loops commonly represent control path feedback. By incorporating global and local metrics, our approach offers a broader evaluation of graph properties. Definitions and Fig. 4 for betweenness centrality, eccentricity, and closeness centrality are provided.

Definition 1 (Betweenness Centrality)**.** *The betweenness centrality of a node is the sum of the fraction of all pairs of shortest paths that pass through this node.*

It can be calculated as follows.

$$\mathrm{C_B}(v) = \sum_{u_i, u_j \in \mathcal{V}} \frac{\sigma(u_i, u_j) \| v}{\sigma(u_i, u_j)}, \tag{1}$$

where $\sigma(u_i, u_j)$ is the number of shortest paths between nodes u_i and u_j. $\sigma(u_i, u_j) \| v$ is the number of those paths passing through the node v, with $v \neq u_i$ and $v \neq u_j$.

Definition 2 (Closeness Centrality)**.** *The closeness centrality of a node is the reciprocal of the sum of the shortest path distances from this node to all other nodes in the network.*

It can be calculated as follows.

$$\mathrm{C_C}(v) = \frac{1}{\sum_{u \neq v} d(v, u)}, \tag{2}$$

where $d(v, u)$ is the shortest path distance from node v to node u, with $v \neq u$.

Definition 3 (Eccentricity)**.** *The eccentricity of a node is the maximum shortest path distance from this node to any other node in the network.*

It can be calculated as follows.

$$\mathrm{ecc}(v) = \max_{u \in \mathcal{V}} d(v, u), \tag{3}$$

where $d(v, u)$ is the shortest path distance from the node v to the node u, with $v \neq u$.

Betweenness centrality measures how frequently a node acts as an intermediary along the shortest path between any two other nodes in the network, highlighting its role in information flow. As observed in [29], control logic DSPs, which direct signals to manage datapath blocks, typically have higher betweenness centrality than datapath DSPs. Closeness centrality, inversely proportional to the average distance to all other nodes, is generally higher for control path DSPs due to the extensive signals they receive. In contrast, eccentricity tends to have values for datapath DSPs that are more distributed towards either the higher or lower extremes of the overall range, reflecting the distributed placement for localized computations.

B. Datapath DSP graph construction

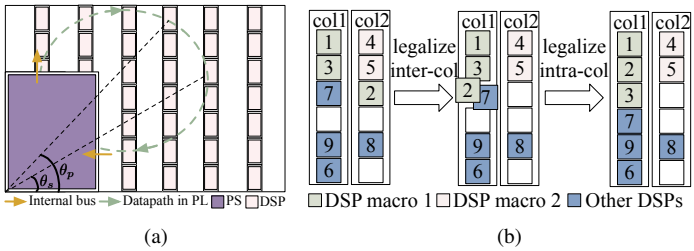
To construct the DSP graph, we perform a search algorithm directly on the netlist to capture the datapath of DSPs and compute their average shortest path distances. Common search algorithms such as Depth-First Search (DFS) and Breadth-First Search (BFS) face limitations in this context: DFS may fail to find the shortest paths, while BFS has high space complexity, making it impractical for large-scale netlists. To address these issues, we adopt an Iterative Deepening Depth-First Search (IDDFS) method, which combines the space efficiency of DFS with the ability to identify shortest paths. This algorithm ensures efficient traversal and supports the accurate construction of the DSP graph.

The construction procedure begins with a netlist traversal to extract all DSP nodes. IDDFS is then applied to each DSP node to compute the shortest paths to other DSPs, recording the paths, the cell type, and the number of cells along each path. This information is then used to construct a DSP graph, retaining only DSP nodes and their connectivity. The resulting graph effectively captures the topology and shortest path distances among DSPs, embedding critical dataflow information for downstream tasks.

However, after constructing an initial DSP graph, a refinement step is necessary. Initial DSP graphs include both datapath DSPs and others, such as control path DSPs. Incorporating control path DSPs into the DSP graph placement can result in a less compact datapath layout, potentially degrading the improvements in timing performance. Our observations reveal that control path DSPs are typically associated with more storage elements, such as flip-flops and RAMs, which facilitate signal holding and allow for greater placement flexibility and less compactness. In contrast, datapath DSPs, characterized by fewer storage elements, require tighter placement to maintain performance. Based on this insight, DSPlacer retains only the datapath DSPs identified by the GCN and removes the others from the DSP graph. This ensures that datapath DSP placement is prioritized for a more compact and efficient layout. After the datapath DSPs are placed, the placement of control path DSPs is handled by standard PnR tools, ensuring a balanced and effective placement strategy.

IV. Datapath-driven DSP placement

In this section, we introduce a datapath-driven DSP placement strategy. Our approach primarily encompasses (1) datapath DSP placement



(a) (b)

Fig. 5 (a) DSP datapath and (b) Datapath DSP legalization.

and (2) datapath DSP legalization. Utilizing the datapath DSP graph, an optimization technique is employed to allocate DSP locations. Then, a subsequent legalization process is applied to ensure strict compliance with the cascade constraints.

A. Datapath DSP placement

In an FPGA device, we assume that there are M DSP locations and each with coordinate $(p_{x,j}, p_{y,j})$, where $j \in \{1, 2, \cdots, M\}$ represents the index of each DSP location. Unlike ASIC placement, DSP placement is an assignment problem. In this context, all DSP components in the design netlist must be mapped to specific DSP locations within the device to optimize timing performance. We introduce binary variables $x_{i,j} \in \{0, 1\}$ to represent the mapping of each DSP component to a location. Specifically, if $x_{i,j} = 1$, the i-th DSP component is mapped to the j-th DSP location in the device, where $i \in \{1, 2, \cdots, N\}$, and $j \in \{1, 2, \cdots, M\}$. N is the number of datapath DSP components in the netlist identified in Section III and M is the number of available DSP locations in the device. $N \leq M$ ensures that all DSP components can be placed within the device. Therefore, we intuitively identify two constraints, as follows.

$$\sum_{j=1}^{M} x_{i,j} = 1, \qquad \sum_{i=1}^{N} x_{i,j} \leq 1, \tag{4}$$

where the first constraint ensures that each DSP component must be mapped to a DSP location, and the second states that each DSP location can be occupied by at most one DSP component.

In addition to constraint (4), DSP macros must also be considered. DSP components within the same macro must be assigned to adjacent positions within the same column. Assume that the DSP location list is sorted in ascending order of coordinates, such that adjacent locations within the same column have consecutive indices. This enables us to express the cascade constraint as follows.

$$x_{c_p,j} = x_{c_s,j+1}, \quad \forall c \in C, \tag{5}$$

where C denotes the set of cascaded DSP component pairs within DSP macros, with c_p and c_s as the predecessor and successor indices of each pair, respectively. The constraint (5) ensures that each pair of cascaded DSP components and their corresponding locations are either simultaneously placed or not placed.

In addition to the constraints above, we incorporate datapath information as observed in Fig. 5(a). In modern FPGA devices (e.g., Xilinx ZCU104), the PS is fixed at the bottom-left corner. Data buses transferring from PS to PL are located above the PS, while those from PL to PS are on the right. This layout forms an ideal datapath from the top to the right of the PS. To facilitate smooth routing, we encourage the angle of the predecessor DSP location to be larger than that of the successor for each DSP graph edge. Using the cos function, the soft datapath constraint is formulated as follows.

$$\cos\theta_{e_{Dp}} \leq \cos\theta_{e_{Ds}}, \tag{6}$$

where $\cos\theta_{e_{Dp}} = \sum_{j=1}^{M} x_{e_{Dp},j} p_{x,j} / \sqrt{p_{x,j}^2 + p_{y,j}^2}$ is the cosine angle between the predecessor and horizontal line, and $\cos\theta_{e_{Ds}} = \sum_{j=1}^{M} x_{e_{Ds},j} p_{x,j} / \sqrt{p_{x,j}^2 + p_{y,j}^2}$ is the cosine angle between the successor and horizontal line. However, different from constraints (4) and (5), we take the datapath constraint as a penalty term in the objective to encourage datapath DSP placement to satisfy our desired datapath.

Under the constraints defined in (4) (5) and penalty term in (6), our objective in datapath DSP placement is to minimize the distance between any two connected datapath DSP components and between a datapath DSP component and any fixed-location component. Thus, our mathematical formulation is defined as follows.

$$\begin{aligned} \min_{x_{i,j}} & \sum_{e \in \mathcal{E}} (\boldsymbol{x}_{e_p} - \boldsymbol{x}_{e_s})^\top (\boldsymbol{p}_x \boldsymbol{p}_x^\top + \boldsymbol{p}_y \boldsymbol{p}_y^\top)(\boldsymbol{x}_{e_p} - \boldsymbol{x}_{e_s}) \\ & + \lambda \sum_{e_D \in \mathcal{E}_D} (\cos\theta_{e_{Dp}} - \cos\theta_{e_{Ds}}) \\ & + \eta \sum_{c \in C} \sum_{j=1}^{M-1} (x_{c_p,j} - x_{c_s,j+1})^2, \\ \text{s.t.} \quad & (4), x_{i,j} \in \{0, 1\}. \end{aligned} \tag{7}$$

where $\mathcal{E}$ denotes the edge set in the netlist graph and $\mathcal{E}_D$ denotes the edge set in the datapath DSP graph. $\boldsymbol{x}_{e_p}$ and $\boldsymbol{x}_{e_s}$ represent the assignment variables for any two connected components via an edge in the netlist graph. Note that $\boldsymbol{x}_{e_p}$ or $\boldsymbol{x}_{e_s}$ is constant if the corresponding component is not a datapath DSP. These variables will be determined by the off-the-shelf FPGA placement tool. $\boldsymbol{p}_x$ ($\boldsymbol{p}_y$) denotes the x (y) coordinate value vector for DSP locations. $\theta_{e_{Dp}}$ ($\theta_{e_{Ds}}$) represents the angle between the horizontal line and the predecessor (successor) of the datapath DSP for the edge e in the datapath DSP graph. $x_{i,j}$ represents the scalars in $\boldsymbol{x}_{e_i}$ and $\boldsymbol{x}_{e_j}$. λ is a hyperparameter that controls the trade-off between distance and datapath effort. $\eta \sum_{c \in C} \sum_{j=1}^{M-1} (x_{c_p,j} - x_{c_s,j+1})^2$ indicates that constraint (5) has been relaxed to penalty, where η is the penalty factor. We rewrite Formulation (7) as follows.

$$\begin{aligned} \min_{x_{i,j}} & \sum_{i=1}^{N} \sum_{j=1}^{M} \sum_{p=1}^{N} \sum_{q=1}^{M} a_{i,j,p,q} x_{i,j} x_{p,q} + \lambda \sum_{i=1}^{N} \sum_{j=1}^{M} c_j x_{i,j}, \\ \text{s.t.} \quad & (4), x_{i,j} \in \{0, 1\}. \end{aligned} \tag{8}$$

where $a_{i,j,p,q}$ and c_j are quadratic and linear coefficients, respectively.

An ILP solver can solve Formulation (8) by replacing each quadratic term with a binary variable and adding artificial constraints. However, as the design scale increases, the variables grow significantly, leading to prohibitive runtimes. Inspired by [30], we adopt the following heuristic to linearize term $x_{i,j} \cdot x_{p,q}$ and solve iteratively.

$$a_{i,j,p,q} x_{i,j} x_{p,q} \approx \frac{1}{2} a_{i,j,p,q} (x_{i,j}^{'} x_{p,q} + x_{i,j} x_{p,q}^{'}), \tag{9}$$

where $x_{i,j}^{'}$ and $x_{p,q}^{'}$ are the values of $x_{i,j}$ and $x_{p,q}$ from the previous iteration. Using the linearization technique in Equation (9), the objective function in Formulation (8) is a weighted sum of all the $x_{i,j}$, which can be solved through a min-cost network flow model [30]. The weighted sum of $x_{i,j}$ variables can be viewed as several assignments from DSP components to DSP locations, with constraint (4) integrated into the flow model by specifying edge capacities.

B. Datapath DSP legalization.

In Section IV-A, the cascade constraint (5) is introduced and relaxed into a penalty term within the MCF formulation as a soft constrain. However, while constraint (5) can be satisfied by finding the optimal solution to formulation (7), the linearization method does not guarantee optimality, and violations of the constraint may therefore occur. To

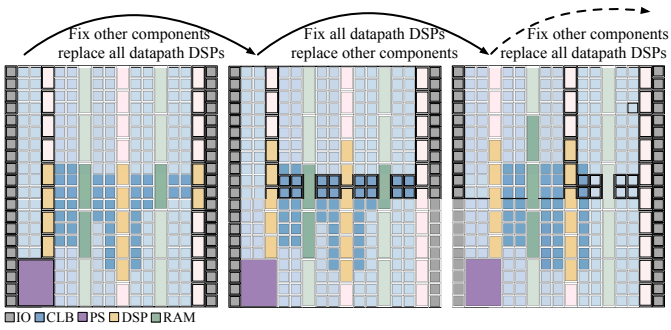


Fig. 6 Incremental Datapath DSP Placement.

address these violations, we introduce a legalization process as a post-processing stage to refine the results obtained from the MCF model. The goal is to enforce cascade constraints while minimizing wire length overhead and preserving the datapath, as guided by the penalty term in (6). To do that, DSP components are adjusted to remain close to their original positions. As shown in Fig. 5(b), the legalization process involves two sequential steps: inter-column legalization to adjust placements across columns, followed by intra-column legalization to refine placements within each column.

For inter-column legalization, we minimize the horizontal displacement by solving the following ILP problem.

$$\min_{t_{i,j}} \sum_{i=1}^{N} \sum_{j=1}^{N_{col}} D_{col}(i,j) t_{i,j}, \tag{10}$$

$$\text{s.t.} \quad \sum_{j=1}^{N_{col}} t_{i,j} = 1, \quad \sum_{i=1}^{N} t_{i,j} \le M_j, \tag{10a}$$

$$t_{c_p,j} = t_{c_s,j}, \quad \forall c \in C, \tag{10b}$$

$$t_{i,j} \in \{0,1\},$$

where $t_{i,j}$ is a binary variable indicating the column to which the DSP component is assigned. N_{col} denotes the number of columns on the device. $D_{col}(i,j)$ denotes the horizontal distance between the newly assigned column and the original column if the i-th DSP component is assigned to the j-th column. $t_{c_p,j}$ ($t_{c_s,j}$) denotes the variable for the predecessor (successor) of the cascaded DSP component pairs, and M_j denotes the number of DSP locations in the j-th column. Constraints (10a) (10b) ensure the solution satisfies constraints (4) (5) at the column level. Since the number of DSP columns is much smaller than the number of DSP locations, the ILP solver remains efficient.

For intra-column legalization, we process each column in parallel and minimize the vertical displacement. For the j-th column, assume that the indices of the DSP components in the column are consistent with the order after sorting by vertical locations. DSP components within a macro are sorted by the average vertical location of the corresponding macro, while other DSP components are sorted by their individual vertical locations. We then solve the formulation below:

$$\min_{r_i} \sum_{i=1}^{N_j} |r_i - R_{col}(i)|, \tag{11}$$

$$\text{s.t.} \quad x_i \in \{1,2,...,M_j\},$$

$$r_{i+1} - r_i = 1, \forall (i,i+1) \in C_j, \tag{11a}$$

$$r_{i+1} - r_i \ge 1, \forall (i,i+1) \notin C_j, \tag{11b}$$

where r_i is an integer variable that indicates the row index of the location to which the i-th DSP component in the column will be assigned. N_j denotes the number of DSP components in the j-th column, $R_{col}(i)$ denotes the original row index of the i-th DSP component in the j-th column and C_j represents cascaded DSP component pairs within the j-th column. Constraint (11a) ensures that the successor is placed immediately after the predecessor in the cascaded pairs, and constraint (11b) allows space between non-cascaded DSP components while preventing overlap.

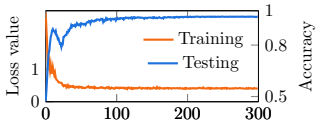
In the datapath DSP placement stage, as shown in Fig. 6, our placement is performed incrementally, alternating between placing datapath DSPs and other components in each iteration. During each iteration, we first fix the positions of the other components and place all the datapath DSPs, then fix the positions of the datapath DSPs and place the remaining components. This method incorporates datapath information while alleviating detours caused by the datapath-driven approach.

TABLE I Benchmarks detail.

Design	#LUT	#LUTRAM	#FF	#BRAM	#DSP	DSP%	freq.(MHz)
iSmartDNN	53503	2919	55767	122	197	11%	130.0
SkyNet	43146	2748	51410	192	346	20%	150.0
SkrSkr-1	35743	3611	53887	196	642	37%	195.0
SkrSkr-2	70558	3815	64007	196	1180	68%	175.0
SkrSkr-3	70382	3791	67257	196	1431	83%	175.0

	SVM [28]	GCN
iSmartDNN	80%	88.1%
SkyNet	85%	90%
SkrSkr-1	96%	96%
SkrSkr-2	69%	95.5%
SkrSkr-3	78.7%	96.6%

(a)



(b)

Fig. 7 (a) Datapath DSP identification comparison; (b) Training and Testing.

V. Experimental Results

A. Target Device, Benchmark and Environment

In the DSPlacer framework, the datapath DSP extraction and DSP assignment tasks are relatively independent. To facilitate implementation, the datapath DSP extraction was developed in Python, utilizing the NetworkX [31] library for graph construction and the PyTorch [32] Geometric library for the graph learning model. For the DSP assignment, we implemented the MCF in C++ and used the Lemon solver [33] for assignment tasks. The cascade legalization after MCF was solved using Gurobi [34]. For placement, routing, and timing analysis of other components, we employed AMF-Placer 2.0 [11], [35] and Xilinx Vivado 2020.2. Since Vivado lacks a built-in wirelength calculation function, RapidWright [36] was used to compute the wirelength. All experiments were conducted on a Linux machine with 10 cores, targeting the Xilinx Zynq UltraScale+ MPSoC ZCU104 FPGA. It is worth mentioning that AMF-Placer 2.0, as an academic open-source placer, originally only supported the Xilinx VCU108 platform. Significant effort was dedicated to adapting it for ZCU104. The benchmarks include several designs from the DAC System Design Contest: iSmartDNN [37], SkyNet [38], and SkrSkr [39].

B. GCN model training and performance

Our classification methodology leverages global graph properties to capture structural importance and node connectivity within DSPs, forming a robust basis for classification. In contrast, PADE [28] only employs automorphism-based features to extract datapath regularity. To evaluate the model, we adopt a leave-one-out strategy across five benchmarks: four benchmarks are used for training, and the resulting

TABLE II Experiment Result.

Benchmark	Vivado				AMF				DSPlacer			
	WNS (ns)	TNS (ns)	HPWL (um)	Runtime (s)	WNS (ns)	TNS (ns)	HPWL (um)	Runtime (s)	WNS	TNS	HPWL (um)	Runtime (s)
iSmartDNN	-0.131	-0.391	2965232	771	-0.830	-649.995	5152227	1552	**0.151**	**0**	3107487	865
SkyNet	-0.164	-7.765	3482155	868	-0.154	-3.557	3633692	970	**0.189**	**0**	3878462	940
SkrSkr-1	-0.587	**-97.799**	2488788	775	-0.883	-4043.358	3384970	691	**-0.164**	-109.042	4473110	701
SkrSkr-2	-0.597	-826.739	3700116	1091	-1.865	-16672.268	8809141	4486	**0.009**	**0**	4489697	2462
SkrSkr-3	-0.216	-21.417	4034489	1232	-0.786	-804.950	6104589	3329	**0.007**	**0**	4912782	1755
Normalize	1.325×	1.042×	0.550×	0.485×	1.658×	1.103×	1.446×	2.145×	**1.000**×	**1.000**×	1.000×	1.000×

model is tested on the remaining benchmark. This process is repeated for all benchmarks to ensure comprehensive evaluation. Our GCN-based approach achieves an average accuracy of 96%, significantly outperforming PADE's SVM-based model, which achieves around 81% accuracy on average, as shown in Fig. 7(a). Additionally, an accuracy curve is presented in Fig. 7(b). These results underscore the advantages of global centrality features over local automorphism-based methods, showcasing superior classification performance and adaptability for node classification in netlists.

C. Placement setting

To evaluate the performance of DSPlacer across varying DSP counts, we extended the SkrSkr benchmark suite by incorporating a broader spectrum of DSP configurations, resulting in three distinct benchmarks: SkrSkr-1, SkrSkr-2, and SkrSkr-3. Our experiments utilize five benchmarks with DSP counts ranging from 197 to 1431, as detailed in TABLE I. To demonstrate the advantage of DSPlacer, we first use Vivado for placement while progressively increasing the clock frequency for each benchmark until a negative WNS is observed. At the same frequency, DSPlacer is then employed for placement. If DSPlacer avoids negative WNS under the same conditions, its advantages are validated. Furthermore, the number of iterations of DSPlacer's internal MCF algorithm is set to 50. The hyperparameter λ, which governs the trade-off between minimizing distance and preserving datapath integrity, is set to 100 based on the experiment.

D. Placement performance comparison

To evaluate the timing improvements achieved by DSPlacer, we report post-route PPA metrics, including routed wirelength, setup WNS, TNS, and total runtime, as summarized in TABLE II. While AMF is tailored for the VCU108 platform, its performance, and adaptability to other platforms, such as ZCU104, is limited. For WNS and TNS, DSPlacer achieves improvements of 65% and 10%, respectively, with reduced wirelength and runtime. Compared to Vivado, DSPlacer improves setup WNS by an average of 32.5% and up to 60% in the benchmark case SkrSkr-2, at the cost of an additional half runtime and HPWL. These results demonstrate that DSPlacer delivers superior WNS and TNS performance compared to Vivado and AMF-Placer while offering greater universality and robustness than AMF-Placer.

E. Analysis, Profiling, and Visualization

The timing improvement achieved by DSPlacer derives from satisfying two critical datapath placement requirements: (1) cascading datapath DSPs to create a more compact layout and (2) preserving the datapath information between PS and PL. These optimizations significantly reduce total WNS. However, the trade-off for the compactness is a slightly increased routing time due to a medium congestion level.

Fig. 8 presents the runtime breakdown of DSPlacer on iSmartDNN and SkyNet. The prototype placement solution and other component placements dominate the total runtime (90.61% for iSmartDNN and 88.31% for SkyNet), whereas datapath DSP extraction and datapath-driven DSP placement account for only about 2% of the total runtime.

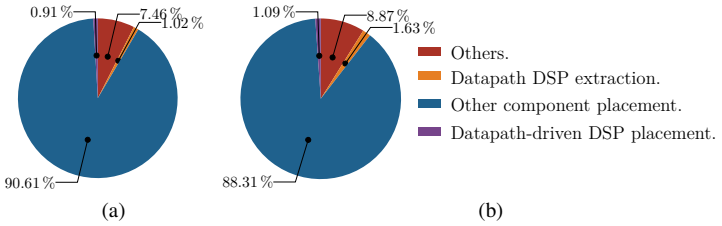


Fig. 8 Runtime profiling: (a) iSmartDNN and (b) SkyNet.

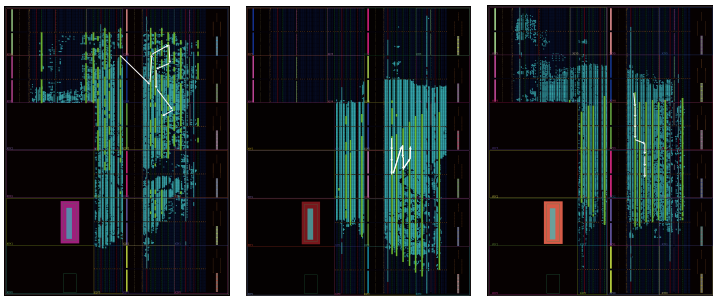


Fig. 9 Datapath visualizations of the SkrSkr-1 placement layout within different design tools: (a) Vivado, (b) AMF, and (c) DSPlacer.

Fig. 9 visualizes the post-route layouts of SkrSkr using different placement tools. Compared to Vivado, DSPlacer produces a more compact and regular datapath layout. While AMF achieves a compact layout similar to DSPlacer, it fails to maintain the datapath information between PS and PL, resulting in a disordered datapath.

VI. Conclusion

In this paper, we propose DSPlacer, a DSP placement framework for FPGA-based CNN accelerators, designed to enhance timing performance. DSPlacer combines GCN techniques with optimization methods to address the challenges of DSP placement. Specifically, the framework extracts datapath DSPs and their shortest path information to enable precise identification of critical DSP nodes. The placement optimization process is formulated as a 0-1 integer programming problem, further simplified into a min-cost-flow model to improve computational efficiency. By integrating datapath-specific information into the placement stage, DSPlacer complied with cascading constraints while ensuring timing closure. Experimental results show a 32% improvement in WNS compared to Vivado and a 65% improvement over AMF. These results highlight the framework's ability to achieve substantial timing performance improvements.

Acknowledgement

This work is supported in part by the National Key Research and Development Program of China (No. 2023YFB4402900), the National Natural Science Foundation of China (No. 62304197) and Pangomicro.

REFERENCES

[1] X. Wei, Y. Liang, and J. Cong, "Overcoming data transfer bottlenecks in FPGA-based DNN accelerators via layer conscious memory management." in *Proc. DAC*, 2019, pp. 125–1.

[2] C. Hao, X. Zhang, Y. Li, S. Huang, J. Xiong, K. Rupnow, W.-m. Hwu, and D. Chen, "FPGA/DNN co-design: An efficient design methodology for 1ot intelligence on the edge," in *Proc. DAC*. IEEE, 2019, pp. 1–6.

[3] Y. Zhang, X. Zhang, P. Xu, Y. Zhao, C. Hao, D. Chen, and Y. Lin, "AutoAI2C: An automated hardware generator for DNN acceleration on both FPGA and ASIC," *IEEE TCAD*, vol. 43, no. 10, pp. 3143–3156, 2024.

[4] K. E. Murray, O. Petelin, S. Zhong, J. M. Wang, M. Eldafrawy, J.-P. Legault, E. Sha, A. G. Graham, J. Wu, M. J. Walker *et al.*, "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM TRETS*, vol. 13, no. 2, pp. 1–55, 2020.

[5] C.-W. Pui, G. Chen, W.-K. Chow, K.-C. Lam, J. Kuang, P. Tu, H. Zhang, E. F. Young, and B. Yu, "RippleFPGA: A routability-driven placement for large-scale heterogeneous FPGAs," in *Proc. ICCAD*. IEEE, 2016, pp. 1–8.

[6] R. Pattison, Z. Abuowaimer, S. Areibi, G. Gréwal, and A. Vannelli, "GPlace: A congestion-aware placement tool for ultrascale FPGAs," in *Proc. ICCAD*. IEEE, 2016, pp. 1–7.

[7] Y.-C. Kuo, C.-C. Huang, S.-C. Chen, C.-H. Chiang, Y.-W. Chang, and S.-Y. Kuo, "Clock-aware placement for large-scale heterogeneous FPGAs," in *Proc. ICCAD*. IEEE, 2017, pp. 519–526.

[8] W. Li, S. Dhar, and D. Z. Pan, "UTPlaceF: A routability-driven FPGA placer with physical and congestion aware packing," *IEEE TCAD*, vol. 37, no. 4, pp. 869–882, 2018.

[9] W. Li, Y. Lin, M. Li, S. Dhar, and D. Z. Pan, "UTPlaceF 2.0: A high-performance clock-aware FPGA placement engine," *ACM TODAES*, vol. 23, no. 4, pp. 1–23, 2018.

[10] T. Liang, G. Chen, J. Zhao, S. Sinha, and W. Zhang, "AMF-Placer: High-performance analytical mixed-size placer for FPGA," in *Proc. ICCAD*, 2021, pp. 1–9.

[11] ——, "AMF-Placer 2.0: Open-source timing-driven analytical mixed-size placer for large-scale heterogeneous FPGA," *IEEE TCAD*, vol. 43, no. 9, pp. 2769–2782, 2024.

[12] T. T. Ye and G. De Micheli, "Data path placement with regularity," in *Proc. ICCAD*. IEEE, 2000, pp. 264–270.

[13] S. I. Ward, M.-C. Kim, N. Viswanathan, Z. Li, C. J. Alpert, E. E. Swartzlander, and D. Z. Pan, "Structure-aware placement techniques for designs with datapaths," *IEEE TCAD*, vol. 32, no. 2, pp. 228–241, 2013.

[14] S. Chou, M.-K. Hsu, and Y.-W. Chang, "Structure-aware placement for datapath-intensive circuit designs," in *Proc. DAC*, 2012, pp. 762–767.

[15] Z. He, P. Liao, S. Liu, Y. Ma, Y. Lin, and B. Yu, "Physical synthesis for advanced neural network processors," in *Proc. ASPDAC*, 2021, pp. 833–840.

[16] J.-M. Lin, W.-F. Huang, Y.-C. Chen, Y.-T. Wang, and P.-W. Wang, "DAPA: A dataflow-aware analytical placement algorithm for modern mixed-size circuit designs," in *Proc. ICCAD*. IEEE, 2021, pp. 1–8.

[17] D. Fang, B. Zhang, H. Hu, W. Li, B. Yuan, and J. Hu, "Global placement exploiting soft 2D regularity," in *Proc. ISPD*, 2022, pp. 203–210.

[18] A. B. Kahng and Z. Wang, "DG-RePlAce: A dataflow-driven GPU-accelerated analytical global placement framework for machine learning accelerators," *arXiv preprint arXiv:2404.13049*, 2024.

[19] A. Koch, "Module compaction in FPGA-based regular datapaths," in *Proc. DAC*, 1996, pp. 471–476.

[20] ——, "Structured design implementation: a strategy for implementing regular datapaths on FPGAs," in *Proc. FPGA*, 1996, pp. 151–157.

[21] H. Kong, L. Feng, C. Deng, B. Yuan, and J. Hu, "How much does regularity help FPGA placement?" in *Proc. FPT*, 2020, pp. 76–84.

[22] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *Proc. ICCAD*, 2016, pp. 1–8.

[23] Y. Ma, Y. Cao, S. Vrudhula, and J.-s. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proc. FPGA*, 2017, pp. 45–54.

[24] S. Gowda, A. Parsons, R. Jarnot, and D. Werthimer, "Automated placement for parallelized FPGA FFTs," in *Proc. FCCM*, 2011, pp. 206–209.

[25] J. Zhang, W. Zhang, G. Luo, X. Wei, Y. Liang, and J. Cong, "Frequency improvement of systolic array-based CNNs on FPGAs," in *Proc. ISCAS*, 2019, pp. 1–4.

[26] H. Hu, D. Fang, W. Li, B. Yuan, and J. Hu, "Systolic array placement on FPGAs," in *Proc. ICCAD*. IEEE, Oct. 2023, p. 1–9.

[27] "AMD Zynq UltraScale+ MPSoCs: Heterogeneous multiprocessing platform for broad range of embedded applications."

[28] S. Ward, D. Ding, and D. Z. Pan, "PADE: a high-performance placer with automatic datapath extraction and evaluation through high dimensional data learning," in *Proc. DAC*, 2012, pp. 756–761.

[29] S. D. Chowdhury, K. Yang, and P. Nuzzo, "ReIGNN: State register identification using graph neural networks for circuit reverse engineering," in *Proc. ICCAD*. IEEE, 2021, p. 1–9.

[30] B. Yu, D. Liu, S. Chowdhury, and D. Z. Pan, "TILA: Timing-driven incremental layer assignment," in *Proc. ICCAD*, 2015, pp. 110–117.

[31] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[32] *Pytorch*, https://pytorch.org/.

[33] *LEMON*, https://lemon.cs.elte.hu/trac/lemon.

[34] *gurobi*, https://www.gurobi.com/.

[35] *AMF-Placer 2.0*, https://github.com/zslwyuan/AMF-Placer.

[36] *rapidwright*, https://www.rapidwright.io/.

[37] *iSmartDNN*, https://github.com/onioncc/iSmartDNN.

[38] *SkyNet*, https://github.com/ZhuangzhuangWu/SkyNet.

[39] *SkrSkr*, https://github.com/jiangwx/SkrSkr.

GTN-Path: Efficient Path Timing Prediction through Waveform Propagation with Graph Transformer

Lihao Liu[1], Beisi Lu[1], Yunhui Li[1], Li Shang[1,2] and Fan Yang[1*]
[1]State Key Lab of Integrated Chips and Systems, School of Microelectronics, Fudan University, Shanghai, China
[2]China and Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, Shanghai, China

***Abstract*—As technology nodes shrink, static timing analysis (STA) must balance accuracy and efficiency to ensure circuit functionality. Graph-based analysis (GBA) is fast but pessimistic, while path-based analysis (PBA) offers higher accuracy with an expensive runtime cost. However, GBA and PBA rely on lookup table (LUT)-based standard cell libraries, introducing accuracy losses compared to accurate SPICE simulations at advanced technology nodes. This work presents GTN-Path, an efficient post-layout path timing prediction method based on waveform propagation and graph transformer network (GTN). GTN-Path captures structural information to accurately predict waveforms by modeling standard cells and interconnects as graphs. Compared to HSPICE simulations, GTN-Path predicts waveforms with 2.98% error and delay with 2.96% error, achieving a speedup of 3510×. Additionally, compared with the sign-off STA tool, the GTN-Path achieves a speedup of 12×.**

***Index Terms*—static timing analysis, graph transformer network, waveform prediction, machine learning.**

I. Introduction

As technology nodes continue to scale down, chip complexity and operating speeds consistently increase [1]. Static timing analysis (STA) remains essential in the design flow, ensuring circuits meet functional and timing requirements [2], [3]. STA commonly employs two methods: graph-based analysis (GBA) and path-based analysis (PBA) [4]. GBA tends to be pessimistic, as it assumes worst-case conditions during propagation. In contrast, PBA evaluates delays based on the specific timing arcs of each cell, offering improved accuracy at the cost of increased runtime. At advanced technology nodes, pessimistic STA will lead to the overdesign of circuits that degrades the performance [5].

Lookup table (LUT)-based timing libraries play a crucial role in modern STA, primarily comprising the nonlinear delay model (NLDM) and current source models (CSMs). NLDM stores only a single delay value, making it inadequate for accurately capturing the complex waveforms of cells at advanced technology nodes. Thus, CSMs that store waveform information are widely adopted in the industry. However, STA using PBA with CSMs is highly time-consuming [6] and can still exhibit discrepancies exceeding 5% compared to SPICE simulations [7]. The discrepancies between STA and SPICE simulations arise from several factors:

- LUT-based timing libraries in STA are derived from limited simulations. For cells operating under conditions that do not exactly match those simulations, interpolation or extrapolation is applied [8], leading to accuracy losses.
- The impact of interconnects on timing analysis becomes increasingly significant [9], particularly for high-fanout nets. Estimation methods for interconnect delay may introduce additional errors in calculating cell delays compared to SPICE simulations.

As the design gets closer to tape-out, more accurate timing analysis is required to guide timing optimization [10]. As a result, SPICE simulations of waveforms at the post-layout stage are applied to path timing analysis of circuits in the industry to meet the accuracy requirements [11], [12]. However, transistor-level SPICE simulations introduce significant runtime overhead.

In addition to accuracy limitations compared with SPICE waveform simulations, STA at advanced nodes also faces challenges that waveform-based timing libraries require substantial computational resources to construct and can consume ten/hundred GB of storage [13]. Furthermore, extracting interconnect parasitic parameters is also a time-intensive process.

Recently, numerous machine learning (ML)-based methods have been proposed to improve timing prediction. Some research focuses on pre-routing timing prediction. For example, in [14], a regression-based method is used to predict net-based delay before routing. Cao et al. [15] employ transformers to predict path delay, while Wang et al. [16] model netlists as graphs and placed cells using a convolutional neural network (CNN) for timing prediction. Additionally, Guo et al. [17] propose a graph convolutional network (GCN)-based method for pre-routing slack prediction. Other efforts target post-layout timing prediction. Ye et al. [18] propose to predict timing based on graph attention network (GAT) and GBA. The aforementioned ML-based methods primarily focus on cell/path delay prediction. However, accurately predicting post-layout waveforms with SPICE-level precision is essential to meet industrial requirements.

To address these challenges, this work presents GTN-Path, a novel post-layout timing path waveform prediction method that utilizes waveform propagation and a graph transformer network (GTN). Our approach aims to achieve SPICE-level accuracy in predicting global path output waveforms by training on local data from standard cells and interconnects. Inspired

This research is supported by National Natural Science Foundation of China (NSFC) research projects 92373207 and 62090025.
*Corresponding author: Fan Yang (yangfan@fudan.edu.cn).

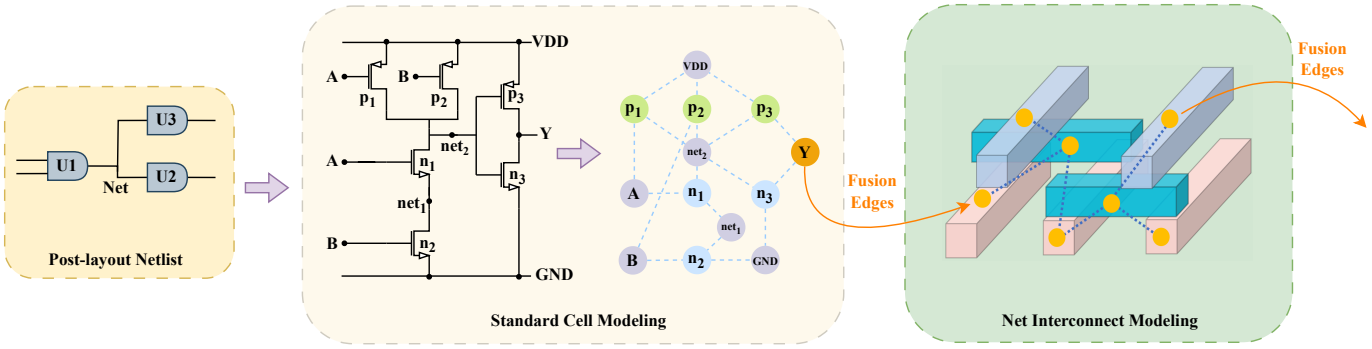


Fig. 1: Graph construction of standard cells and interconnects.

by the natural representation of transistor-level structures of standard cells and net interconnects as graphs, we develop a graph modeling technique to capture detailed information about cells and interconnects. Given that global information along the timing path also influences the timing behavior of cells, a transformer-based model is well-suited for such scenarios. Recognizing that different components within cells and interconnects contribute variably to the final waveforms, we assign different types and features to nodes and features to enhance representational capability. The proposed graph transformer-based model efficiently learns the local relationships between cells and interconnects and the overall graph structure. The contributions of this work can be summarized as follows:

- A post-layout waveform prediction flow for timing paths with SPICE-level accuracy that eliminates the need for constructing timing libraries or extracting interconnect parasitics.
- A graph representation approach to model both the transistor-level structures of standard cells and detailed net interconnects, coupled with a GTN-based model that efficiently captures the structural relationships between them. The model is trained on local waveform data from standard cells and interconnects and predicts the overall path output using waveform propagation.
- Compared to HSPICE, the predicted waveforms of timing paths achieve an average relative error of 2.98% in output voltage waveforms and 2.96% in calculated delay on unseen testcases, with a speedup of 3510×. Additionally, compared to the sign-off STA tool, GTN-Path delivers a speedup of 12×. The training dataset is constructed from standard cell data sampled from only 30% of all paths in real industrial circuits.

The remainder of this article is organized as follows: Section II provides background on STA and GTNs. Section III details the proposed GTN-based path timing prediction method. Section IV presents the experimental results, and Section V concludes the article.

II. Background

This section introduces the basics for relevant background. Firstly, the basics of STA are described. Then the definition of GTN is discussed.

A. Static Timing Analysis

STA is crucial in modern circuit design, verifying functionality and timing. It primarily includes two methods: GBA and PBA. In GBA, the worst-case transition time of all cell input pins is used to calculate delays, often resulting in a pessimistic calculation. In the PBA, specific timing arcs of cells are analyzed, and the actual delay is propagated through the timing path to calculate the final delay, which reduces the pessimism [6], [19].

At advanced technology nodes, single delay values cannot accurately represent the complex waveforms of cells and paths. To address this, CSMs are used to calculate the timing behavior of cells based on waveforms. SPICE simulations offer the highest accuracy but incur significant computational costs. Additionally, both SPICE waveform simulations and sign-off STA rely on pre-generated timing libraries and extracted interconnect parasitics. While accurate, these methods become increasingly time-consuming as the number of timing paths grows.

B. Graph Transformer Networks

A graph is a structure used to represent objects and their relationships, and a graph neural network (GNN) is an ML method designed to learn from such graph structures, making it a powerful tool for electronic design automation (EDA) tasks [20]–[22]. Traditional GNNs learn graph embeddings through message passing and information updating. The GTN enhances this approach by incorporating the attention mechanism from transformers [23], [24]:

$$\alpha_{ij} = softmax_j\left(\frac{Qh_{v_i} \cdot Kh_{v_j}}{\sqrt{d}}\right), \quad (1)$$

where α_{ij} represents the attention score between nodes v_i and v_j, h_{v_i} is the node embedding, Q and K are learnable parameters, and d is the scaling factor. By taking these attention scores into account, the GTN computes the final embeddings as follows:

$$h_{v_i}^{l+1} = \phi\left(W^l AGGR_{v_j \in N(v_i)}(\alpha_{ij}^l V^l h_{v_j}^l) + W_0^l h_{v_i}^l\right), \quad (2)$$

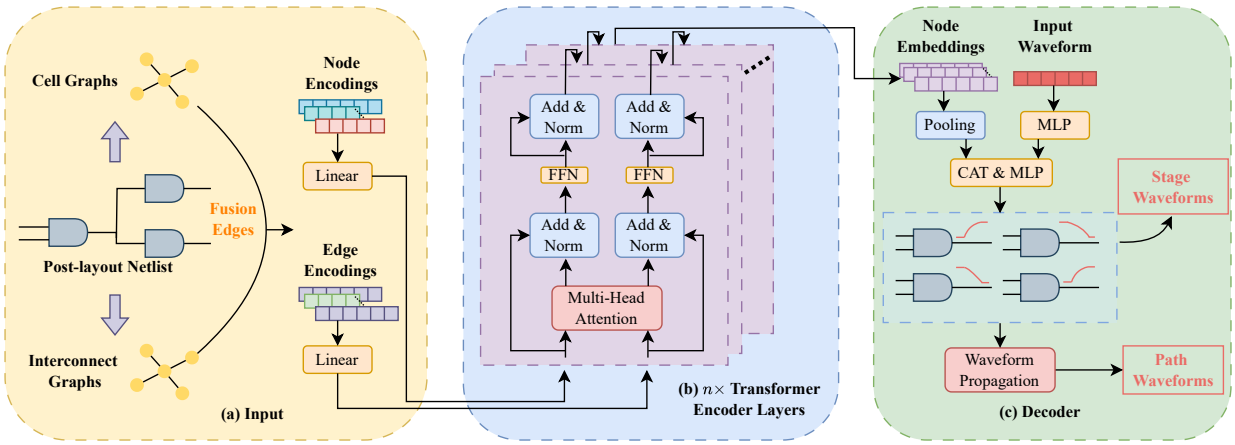


Fig. 2: The framework of the proposed GTN-based path waveform prediction method. (a) Model input with distinct node and edge types. (b) Graph transformer-based encoder layers for feature extraction. (c) Decoder for predicting final path waveforms through waveform propagation.

where $h_{v_i}^l$ denotes the node embedding at layer l, $AGGR$ is the aggregation method, and W, V are learnable parameters. Like transformers, positional encodings are added to the graph to capture global, graph-level information [25].

III. GTN-based Path Timing Prediction

This section introduces the proposed GTN-based path waveform prediction method. Firstly, we present the details of the graph construction process. Next, the architecture of the model is discussed. Finally, the waveform propagation mechanism and model training methodology are explained.

A. Graph Modeling of Standard Cells and Interconnects

At advanced technology nodes, interconnects play a significant role in cell delay and cannot be neglected. The transistor-level structures of standard cells and interconnects can be naturally represented as graphs, as illustrated in Fig. 1. We define a graph as $G = (V, E)$, where $V = \{v_1, v_2, \cdots, v_n\}$ is the set of nodes and $E = \{e_{ij} | v_i, v_j \in V\}$ is the set of edges. Based on the post-layout netlists, we construct graphs for each cell in the timing path, including the driving standard cells, their corresponding interconnects, and the sink standard cells, then connect them by fusion edges.

1) Graph Representation of Standard Cells: For transistor-level standard cell modeling, nets and transistors are represented as nodes in the graph, with edges constructed based on their connection relationships. Given that different components have varying impacts on the timing behavior of the cell, we assign different types of nodes, which can be summarized as follows:

- **Nets:** Features include the net type and the node degree. Net types are categorized as IO pins along the path, other IO pins, VDD, GND, and other nets.
- **Output pin of the cell along the path:** For the output pin of the cell in the timing path, a unique node type is assigned. Node features include the node's degree and the cell's fanout.
- **PMOS and NMOS:** Two types of transistors are modeled by two types of nodes, with features including channel width, length, and number of fins of transistors.

By encoding different types and features of nodes, the detailed structures of the components are effectively captured, enabling the GTN model to learn the relationships between cell components and predict their timing behavior. To further enhance the model's ability to extract information from the global structure of standard cells, the 3 smallest non-trivial Laplacian eigenvectors are incorporated into the node features.

For edge features, one-hot vectors are used to represent different connection types between nodes, such as nets connected to the drain, source, or gate of a transistor.

2) Graph Representation of Interconnects: Given the circuit layout, the multi-layer interconnects of the net between two cells in the timing path can be extracted. The interconnects are segmented whenever there is a change in direction, with each segment forming a node in the graph. Features of nodes are the length, width, and height of the segment. Edges are created based on the physical connectivity of segments, with edge features representing the distances between the center of the connected nodes.

Once the graphs for cells and interconnects at each stage in the timing path are constructed, fusion edges are introduced to connect them, as illustrated in Fig. 1. A fusion edge represents a new edge type, with features that include the distances between the pins of the cell and the center of the connected interconnect. At each stage, fusion edges link driving cells to their corresponding interconnects and also connect interconnects to sink cells.

B. Graph Transformer-Based Model

The constructed graph for each cell in the path contains detailed information and the relationships between standard cells and interconnects. Based on this graph, the proposed GTN-based model generates high-level graph embeddings. The framework of the GTN-based path prediction method is

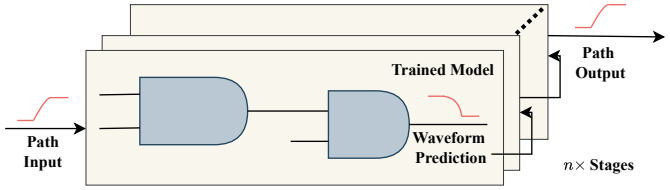


Fig. 3: Illustration of waveform propagation.

TABLE I: Statistics of The Testcases

Design	#Nets	#Cells	#Paths	#FFs
Ckt #1	20335	20325	10789	6632
Ckt #2	42724	42182	5353	7676
Ckt #3	150484	150099	37656	11696

illustrated in Fig. 2. For each cell in the timing path, node and edge encodings are initially passed through a linear transformation. These embeddings are fed into multiple transformer encoder layers to generate high-level graph representations. The model employs a multi-head attention mechanism, where attention scores are calculated between nodes, reflecting the relative importance of one node to others. Edge features are also incorporated into the attention score calculations, as shown in Eq. (3).

$$\alpha_{e,ij}^{k,l} = softmax_j\left(E^{k,l}h_{e_{ij}}^{l} \cdot \frac{Q^{k,l}h_{v_i}^{l} \cdot K^{k,l}h_{v_j}^{l}}{\sqrt{d_k}}\right), \quad (3)$$

$$h_{e_{ij}}^{l+1} = \phi\left(CAT(\alpha_{e,ij}^{k,l}, k = 1, \cdots, H)\right), \quad (4)$$

where $E^{k,l}, Q^{k,l}, K^{k,l}$ are learnable parameters, $h_{e_{ij}}^{l}, h_{v_i}^{l}$ are edge and node embeddings of layer l. Node updates for each attention head follow a similar process as described in Eq. (2), and the embeddings from all attention heads are concatenated to produce the final node embeddings. Edge updates are performed according to Eq. (4).

After the multi-head attention mechanism, residual connections are incorporated into the model to enhance the training of deep architectures. The model produces high-level graph embeddings following the message passing and updating process. These embeddings are then utilized to predict the waveforms of cells and timing paths.

C. Waveform Propagation and Model Training

GTN-Path predicts the waveforms of standard cells in the timing path using the generated graph embeddings and computes the waveforms of the entire path through waveform propagation, as illustrated in Fig. 2(c). The input waveforms of standard cells are first processed through an MLP. The pooled node embeddings from the graph are then concatenated with the waveform embeddings to produce the predicted waveforms.

The model predicts voltage waveforms at 13 specific time points, corresponding to voltage levels at specific ratios of VDD: 0.02, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, and 0.98. As depicted in Fig. 3, waveform propagation is used to predict the final waveform of the entire timing path. For each standard cell in the path, the previous cell's predicted output waveform serves as the next cell's input waveform. Starting from the initial input waveform, the final output waveform of the timing path is iteratively predicted.

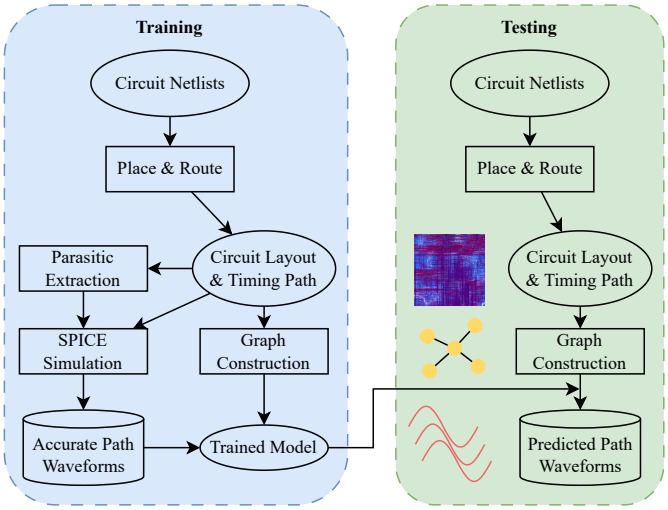
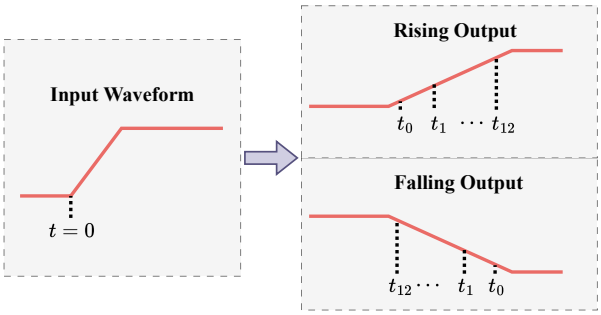


Fig. 4: Training and testing flow of the GTN-Path.



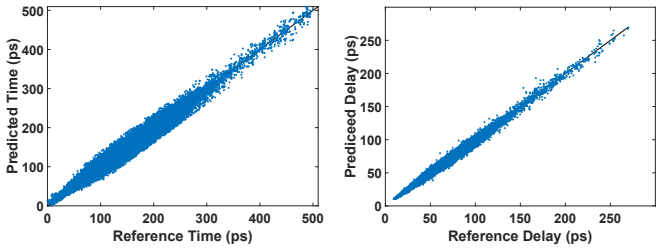
Ratio of the VDD												
t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}
0.02	**0.05**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**0.95**	**0.98**

Fig. 5: Illustration of the model's capability to automatically predict rising and falling edges of waveforms.

Our framework adopts two transformer layers, and all MLPs have one hidden layer of 128 neurons. To accommodate both rising and falling edges in waveforms, the predicted time points are arranged in increasing order for rising waveforms and decreasing order for falling waveforms. This setup allows the model to automatically handle rising and falling conditions of the output waveforms based on the given input waveforms, as shown in Fig. 5. We adopt the mean square error (MSE) as the loss function during model training. The golden values are chosen as the simulation results from HSPICE. The learning rate is set to 0.001, and the batch size is set to 4096 based on the capability of the GPU.

TABLE II: Comparison of Predicted Waveforms and HSPICE Simulations

Testcases			Ckt #1		Ckt #2		Ckt #3		Avg	
			Cell	Path	Cell	Path	Cell	Path	Cell	Path
GTN-Path	Waveform Error	$Mean$ (%)	4.47	3.60	4.46	2.93	3.14	2.40	4.02	**2.98**
		Std (%)	6.20	2.77	4.04	2.39	5.37	1.94	5.20	**2.37**
	Delay Error	$Mean$ (%)	3.37	3.49	3.79	3.00	2.50	2.39	3.22	**2.96**
		Std (%)	3.35	2.70	3.19	2.31	2.14	1.92	2.89	**2.31**
	Time (s)		**4.37**		**2.30**		**15.60**		**7.42**	
STA	Time (s)		46.93		71.75		149.85		89.51	
	Speedup (x)		**10.74**		**31.20**		**9.61**		**12.06**	
HSPICE	Time (s)		19577.57		11015.14		47556.48		26049.73	
	Speedup (x)		**4479.99**		**4789.19**		**3048.49**		**3510.75**	



(a) (b)

Fig. 6: Comparison between predicted and reference HSPICE results: (a) Predicted waveform time points for each standard cell. (b) Computed delay for each standard cell.

IV. Experiments

This section evaluates the proposed method by comparing the predicted cell and path waveforms against HSPICE results. Additionally, a comparison with other ML techniques techniques is presented to demonstrate the effectiveness of the approach. The results show superior efficiency compared with the SPICE simulation as well as sign-off STA tools, with a SPICE-level prediction accuracy.

A. Experiment Setup

GTN-Path is trained and evaluated on a Linux server with an Nvidia Tesla V100 GPU and an Intel Xeon Silver 8358 CPU operating at 2.60 GHz. The implementation was performed in Python, using the PyTorch framework and the PyTorch-geometric toolkit [26]. We build the dataset for training and testing based on the ASAP 7nm FinFET PDK [27].

The overall training and testing scheme is summarized in Fig. 4. The dataset consists of three industrial designs, with detailed information summarized in TABLE I. For the training dataset, standard cells and their corresponding interconnects are randomly selected from 30% of the extracted timing paths in the testcases, where 10% of the standard cells are reserved for validation. The remaining standard cells from 70% of the timing paths are used as the unseen testing dataset. To generate the dataset labels, we first synthesize the designs using Synopsys Design Compiler. The designs are then placed and routed using Cadence Innovus, followed by parasitic extraction with Synopsys StarRC. Finally, HSPICE simulations are conducted to generate the waveforms for the timing paths and the standard cells within those paths.

To evaluate the accuracy of the trained model, we compare the predicted values against the golden reference results obtained from HSPICE simulations. The mean absolute relative error (MARE) is employed to quantify the discrepancy between the predicted and reference values. The MARE is calculated as follows:

$$MARE = \frac{1}{N}\sum_{i=1}^{N}(|1 - \frac{y_o}{y}|), \tag{5}$$

where N is the number of predicted values, y_o means the predicted value and y means the reference value.

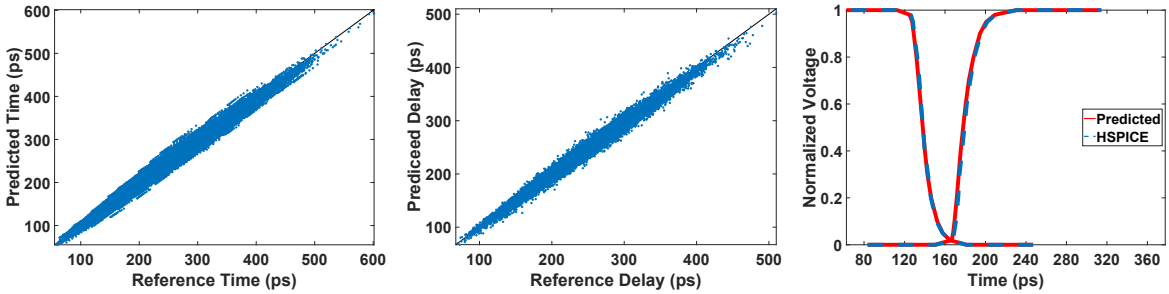
B. Evaluation of Predicted Waveforms

1) Evaluation of Cell Waveforms Prediction: After training the model as described in Section III-C, we evaluate the accuracy of the predicted waveforms for each standard cell. For each testing cell in the 70% unseen data, we first construct the graph using the method illustrated in Fig. 1 and then utilize the model outlined in Fig. 2 to predict the waveforms of standard cells. The predicted waveforms include 13 time points at which the voltages reach specific ratios of VDD, as discussed in Section III-C.

The comparison between the predicted waveforms and the reference values is summarized in TABLE II. $Mean$ is the MARE between predicted and reference waveforms, Std is the standard deviation, and $Cell$ and $Path$ columns refer to the error in standard cell and path waveform prediction, respectively. The results show that for testing standard cells from completely unseen paths, GTN-Path achieves a MARE of 4.02% and 3.22% for waveform prediction and delay calculation results, respectively. Additionally, Fig. 6 presents the predicted waveforms and calculated delays of standard cells from GTN-Path compared to the reference HSPICE values. The figure demonstrates that the predicted values align closely with the HSPICE simulation results.

2) Evaluation of Path Waveform Propagation: We evaluate the accuracy and efficiency of GTN-Path using the waveform propagation method discussed in Section III-C. For each testing timing path on the 70% unseen testing paths, waveform propagation, as illustrated in Fig. 3, is performed to predict the output waveforms and HSPICE simulations are conducted for comparison. The detailed statistics of the comparison are summarized in TABLE II.

The results show that the prediction has only 2.98% MARE for waveforms and 2.96% for calculated delay. Fig. 7a and Fig.



(a) (b) (c)

Fig. 7: Comparison between predicted and reference HSPICE results: (a) Predicted waveform time points for each path. (b) Computed delay for each path. (c) Predicted waveforms with a 3% error compared with HSPICE simulation.

TABLE III: Comparison of Predicted Waveforms with Other ML Techniques

Testcases	# Paths	GCN				GAT				GTN-Path w/o				GTN-Path			
		Waveform Error (%)		Delay Error (%)		Waveform Error (%)		Delay Error (%)		Waveform Error (%)		Delay Error (%)		Waveform Error (%)		Delay Error (%)	
		Mean	*Std*	*Mean*	*Std*	*Mean*	*Std*	*Mean*	*Std*	*Mean*	*Std*	*Mean*	*Std*	*Mean*	*Std*	*Mean*	*Std*
Ckt #1	7552	5.13	4.18	5.13	4.16	4.62	2.91	4.61	2.89	6.38	5.09	6.34	5.04	**3.60**	**2.77**	**3.49**	**2.70**
Ckt #2	3747	11.89	8.45	10.29	7.41	4.78	4.05	4.73	3.88	5.87	5.50	4.21	3.82	**2.93**	**2.39**	**3.00**	**2.31**
Ckt #3	26359	6.32	5.00	6.09	4.53	3.52	2.78	3.52	2.77	3.78	2.95	3.82	2.97	**2.40**	**1.94**	**2.39**	**1.92**
Avg	12552	7.78	5.88	7.17	5.37	4.31	3.25	4.29	3.18	5.34	4.51	4.79	3.94	**2.98**	**2.37**	**2.96**	**2.31**
Ratio	\	2.61	2.48	2.42	2.32	1.45	1.37	1.45	1.38	1.79	1.90	1.62	1.71	**1**	**1**	**1**	**1**

7b compare the predicted time points against the reference values, while Fig. 7c presents timing path waveforms with a MARE of 3% compared with HSPICE simulations. These figures highlight the strong consistency between GTN-Path predictions and HSPICE results.

The runtime for sign-off STA is measured using PrimeTime in SI mode with the CSMs and PBA to perform waveform propagation for the testing paths. The runtime for HSPICE is recorded as the simulation time for the testing paths, while GTN-Path's runtime is calculated as the time taken to perform waveform propagation across all testing paths. The comparison shows that GTN-Path achieves an impressive average speedup of 3510× compared to HSPICE simulations and an average speedup of 12× compared to the commercial sign-off STA tool. Notably, the model training requires only 30% of the paths in the testcases and can be completed within 2 hours.

These results demonstrate that GTN-Path achieves SPICE-level accuracy with a MARE below 3% for predicting timing path waveforms, offering exceptional efficiency and eliminating the need for timing library construction or interconnect parasitic extraction.

C. Comparison with Other ML Techniques

GCN and GAT are two primary ML methods for timing prediction with good results [16]–[18]. However, most existing approaches focus primarily on predicting pre-routing delays and still require timing libraries. At advanced technology nodes, it is essential to consider interconnect effects and perform post-layout SPICE-level waveform simulations.

To demonstrate the effectiveness of GTN-Path, we implemented GCN- and GAT-based methods for predicting post-layout path waveforms and compared them with our approach. The training and testing schemes were consistent for all methods, as discussed in Section IV-A. The results are summarized in TABLE III, which shows that our GTN-Path method achieves the highest accuracy in predicting path waveforms and delays. Compared to GCN and GAT-based methods, GTN-Path leverages the transformer structure with positional encoding, effectively capturing global and local relationships between cells and interconnects.

We further assessed the impact of excluding detailed interconnect modeling by representing the pre-routing net as a single graph node with features based on [14], as shown in the *GTN-Path w/o* column of TABLE III. The results show that by modeling the detailed interconnects, the accuracy of GTN-Path improves by 79% and 62% for predicting path waveform and delay, respectively. This shows the importance of modeling the detailed interconnects at advanced nodes for SPICE-level accuracy waveform prediction.

V. Conclusion

This paper presents GTN-Path, an efficient graph transformer network (GTN)-based approach for predicting post-layout timing path waveforms. GTN-Path represents transistor-level structures of standard cells and interconnects as graphs and connect them by fusion edges, assigning various node and edge types with features to enhance their representational power. Leveraging the constructed graph, the model generates high-level graph embeddings and predicts timing path waveforms through waveform propagation. Compared to HSPICE simulations, GTN-Path achieves an average relative error of 2.98% in path output voltage waveforms and 2.96% in the calculated delay while providing an average speedup of 3510×. Additionally, compared with the sign-off STA tool, the GTN-Path achieves a speedup of 12×.

REFERENCES

[1] Linyu Zhu, Yue Gu, and Xinfei Guo. Rc-gnn: Fast and accurate signoff wire delay estimation with customized graph neural networks. In *2023 IEEE 5th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 1–5. IEEE, 2023.

[2] Kevin Kai-Chun Chang, Chun-Yao Chiang, Pei-Yu Lee, and Iris Hui-Ru Jiang. Timing macro modeling with graph neural networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1219–1224, 2022.

[3] Zizheng Guo, Tsung-Wei Huang, Zhou Jin, Cheng Zhuo, Yibo Lin, Runsheng Wang, and Ru Huang. Heterogeneous static timing analysis with advanced delay calculator. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[4] Zuodong Zhang, Zizheng Guo, Yibo Lin, Runsheng Wang, and Ru Huang. Eventtimer: Fast and accurate event-based dynamic timing analysis. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 945–950. IEEE, 2022.

[5] Sneh Saurabh, Hitarth Shah, and Shivendra Singh. Timing closure problem: Review of challenges at advanced process nodes and solutions. *IETE Technical Review*, 2019.

[6] Yasin Zamani and Tsung-Wei Huang. A high-performance heterogeneous critical path analysis framework. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2021.

[7] Tariq El Motassadeq. Ccs vs nldm comparison based on a complete automated correlation flow between primetime and hspice. In *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, pages 1–5. IEEE, 2011.

[8] Qin Tang, Amir Zjajo, Michel Berkelaar, and Nick van der Meijs. Rde-based transistor-level gate simulation for statistical static timing analysis. In *Proceedings of the 47th Design Automation Conference*, pages 787–792, 2010.

[9] Olivier Coudert. Timing and design closure in physical design flows. In *Proceedings International Symposium on Quality Electronic Design*, pages 511–516. IEEE, 2002.

[10] Hsien-Han Cheng, Iris Hui-Ru Jiang, and Oscar Ou. Fast and accurate wire timing estimation on tree and non-tree net structures. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[11] Shahin Nazarian, Hanif Fatemi, and Massoud Pedram. Accurate timing and noise analysis of combinational and sequential logic cells using current source modeling. *IEEE transactions on very large scale integration (VLSI) systems*, 19(1):92–103, 2010.

[12] Tobias Thiel. Have i really met timing?-validating primetime timing reports with spice. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 3, pages 114–119. IEEE, 2004.

[13] Sneh Saurabh and Priyanka Mittal. A practical methodology to compress technology libraries using recursive polynomial representation. In *2018 31st International Conference on VLSI Design and 2018 17th International Conference on Embedded Systems (VLSID)*, pages 301–306. IEEE, 2018.

[14] Xu He, Zhiyong Fu, Yao Wang, Chang Liu, and Yang Guo. Accurate timing prediction at placement stage with look-ahead rc network. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1213–1218, 2022.

[15] Peng Cao, Guoqing He, and Tai Yang. Tf-predictor: Transformer-based prerouting path delay prediction framework. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(7):2227–2237, 2022.

[16] Ziyi Wang, Siting Liu, Yuan Pu, Song Chen, Tsung-Yi Ho, and Bei Yu. Restructure-tolerant timing prediction via multimodal fusion. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2023.

[17] Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z Pan, and Yibo Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pages 1207–1212, 2022.

[18] Yuyang Ye, Tinghuan Chen, Yifei Gao, Hao Yan, Bei Yu, and Longxing Shi. Graph-learning-driven path-based timing analysis results predictor from graph-based timing analysis. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, pages 547–552, 2023.

[19] Ao Han, Zhenyu Zhao, Chaochao Feng, and Shuzheng Zhang. Stage-based path delay prediction with customized machine learning technique. In *Proceedings of the 2021 5th International Conference on Electronic Information Technology and Computer Engineering*, pages 926–933, 2021.

[20] Pengju Chen, Dan Niu, Zhou Jin, Changyin Sun, Qi Li, and Hao Yan. TSA-TICER: A two-stage TICER acceleration framework for model order reduction. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[21] Haoxing Ren, George F Kokai, Walker J Turner, and Ting-Sheng Ku. Paragraph: Layout parasitics and device parameter prediction using graph neural networks. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2020.

[22] Yunfan Zuo, Yuyang Ye, Hongchao Zhang, Tinghuan Chen, Hao Yan, and Longxing Shi. A graph-learning-driven prediction method for combined electromigration and thermomigration stress on multi-segment interconnects. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE, 2024.

[23] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[24] Vijay Prakash Dwivedi and Xavier Bresson. A generalization of transformer networks to graphs. *arXiv preprint arXiv:2012.09699*, 2020.

[25] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021.

[26] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[27] Lawrence T Clark, Vinay Vashishtha, Lucian Shifren, Aditya Gujja, Saurabh Sinha, Brian Cline, Chandarasekaran Ramamurthy, and Greg Yeric. Asap7: A 7-nm finfet predictive process design kit. *Microelectronics Journal*, 53:105–115, 2016.

APSQ: Additive Partial Sum Quantization with Algorithm-Hardware Co-Design

Yonghao Tan*, Pingcheng Dong*, Yongkun Wu, Yu Liu, Xuejiao Liu, Peng Luo,
Shih-Yang Liu, Xijie Huang, Dong Zhang, Luhong Liang, Kwang-Ting Cheng†
The Hong Kong University of Science and Technology, AI Chip Center for Emerging Smart Systems (ACCESS)

***Abstract*—DNN accelerators, significantly advanced by model compression and specialized dataflow techniques, have marked considerable progress. However, the frequent access of high-precision partial sums (PSUMs) leads to excessive memory demands in architectures utilizing input/weight stationary dataflows. Traditional compression strategies have typically overlooked PSUM quantization, which may account for 69% of power consumption. This study introduces a novel Additive Partial Sum Quantization (APSQ) method, seamlessly integrating PSUM accumulation into the quantization framework. A grouping strategy that combines APSQ with PSUM quantization enhanced by a reconfigurable architecture is further proposed. The APSQ performs nearly lossless on NLP and CV tasks across BERT, Segformer, and EfficientViT models while compressing PSUMs to INT8. This leads to a notable reduction in energy costs by 28-87%. Extended experiments on LLaMA2-7B demonstrate the potential of APSQ for large language models. Code is available at *https://github.com/Yonghao-Tan/APSQ*.**

***Index Terms*—Hardware-aware quantization, partial sum quantization, dataflow, DNN, Transformer**

I. Introduction

Recent advancements in deep neural networks (DNNs) have driven substantial progress in computer vision (CV) and natural language processing (NLP) tasks, including image classification [1], semantic segmentation [2], [3], object detection [4], and question answering [5]. Furthermore, the rise of generative large language models (LLMs) [6] has profoundly influenced various applications. Central to this advancement is the Transformer architecture [7], renowned for its self-attention mechanism that effectively captures long-range dependencies. However, Transformers are notorious for their intensive computation and memory requirements. For instance, semantic segmentation for self-driving vehicles involves processing over 20,000 tokens, while BERT models must handle tokens with substantial hidden dimensions, such as 4,096, resulting in considerable memory overhead. These challenges significantly impact hardware deployment and acceleration. To address these obstacles, the algorithm community has developed model compression techniques, including quantization [8]–[11], pruning [12], and distillation [13].

To further accelerate DNNs at the hardware level, prior works have focused on designing dedicated accelerators utilizing diverse dataflows [14]–[18]. They can be categorized into Input Stationary (IS), Weight Stationary (WS), and Output Stationary (OS). The IS reduces data movement by keeping input tiles stationary [15] to achieve minimum access to the input buffer. The WS tries to minimize weight movement from external DRAM due to insufficient on-chip weight buffer [16]. The OS achieves optimization by reusing partial sum (PSUM) in the output registers [14], enhancing computational efficiency in scenarios with extensive accesses of PSUMs. In addition, Chen et al. [15] proposed Row Stationary (RS) dataflow, which aims to minimize the energy consumption of data movement for convolutional neural networks (CNNs). However, since Transformers rarely employ standard convolution, the RS dataflow has not been widely adopted. The effectiveness of these dataflows is contingent upon several critical aspects, including the layer configuration, degree of parallelism, and on-chip SRAM size. Addressing this, Tu et al. [16] proposes a reconfigurable architecture specifically designed to leverage the unique characteristics of these diverse dataflows optimally. Besides, recent Transformer accelerators [17], [18] employ Score/Key stationary dataflows to accelerate the attention, which is aligned with the dataflows introduced above.

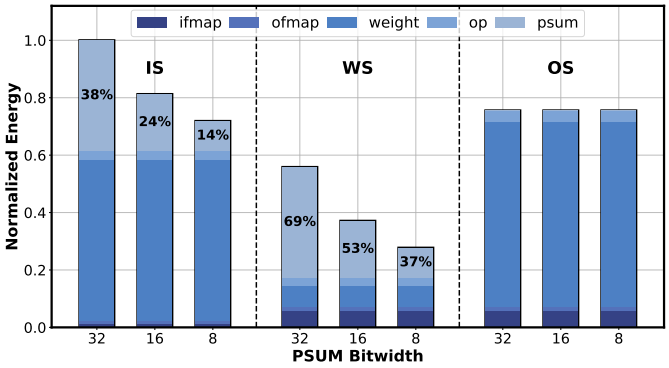


Fig. 1: Energy breakdown of IS, WS, and OS dataflows for the BERT-Base model with 128 input token-length.

The key distinction among the aforementioned dataflows lies in the fact that the IS and WS frequently store and fetch PSUMs from SRAM/DRAM for accumulation, while OS updates PSUMs directly within low-cost registers. We perform an energy consumption analysis following the methods adopted in [16] as shown in Fig. 1. It can be seen that PSUMs constitute a large portion of the energy for IS and WS compared with input feature map (ifmap), output feature map (ofmap), weight, and multiply-add operations (op), especially with larger PSUM bit-width, up to 69% of

*Both authors contributed equally to this research.
†Corresponding author.

total power consumption. Since PSUMs are obtained by matrix accumulation, they need to be stored in higher precision (e.g., INT16/32) than the quantized weights/activation that are stored in INT8 or lower. Recent studies [19], [20] have proposed PSUM quantization (PSQ) technique to reduce analog-to-digital converter (ADC) overheads of high-precision PSUMs in ReRAM-based accelerator. Nevertheless, the PSUMs after ADC are dequantized to their original bit-width and stored in on-chip SRAM, where the memory access overhead of PSUMs fails to be alleviated.

In this study, we introduce a novel additive PSUM quantization strategy denoted as APSQ, which, to the best of our knowledge, is the first to address challenges in IS and WS encountered with large PSUMs. The APSQ enables each quantizer to consider the cumulative prior outcomes rather than just focusing on the current PSUM. In addition, we propose a grouping strategy to improve accuracy with a reconfigurable architecture to support various grouping configurations. The main contributions of this paper are summarized as follows:

- We reveal the critical role of high-precision PSUM access and storage in both IS and WS dataflows, refining an analytical framework with PSUM precision awareness.
- A novel method termed APSQ is designed by introducing the accumulation effect into the quantizer, thereby achieving INT8 PSUM storage and accesses.
- We propose a grouping strategy integrating PSUM quantization and APSQ within a reconfigurable architecture, efficiently accommodating various group sizes with minimal additional hardware resources.
- The INT8 APSQ minimizes accuracy loss to 0.16% for BERT-Base, 0.61% and 0.83% for Segformer-B0 and EfficientViT-B1, while the energy costs are saved by 28-87% for the IS and WS. We further apply APSQ for LLaMA2-7B and achieved only 0.59% accuracy drop while providing up to 31.7× energy savings, highlighting its significant potential for LLMs.

II. Preliminary

A. Energy Efficiency Analysis

In the realm of DNN accelerators, a key factor is energy efficiency in terms of operations/watt, significantly influenced by precision, layer size, dataflow, and architecture. This study adopts the methodologies from [15], [16] to examine energy efficiency in a typical DNN accelerator system shown in Fig. 2. This system includes a multiply-accumulate (MAC) array, on-chip SRAM, off-chip DRAM, and a top controller. The MAC array is organized based on P_o, P_{ci}, and P_{co}, which define parallelism in the output feature map, input channels, and output channels, respectively. The on-chip SRAM comprises a 128KB input/output buffer and a 64KB weight buffer, while the off-chip DRAM capacity is assumed sufficient, and the top controller manages module configuration. Following [16], the total energy cost E_{total} is:

$$E_{total} = N_d \cdot E_{dram} + N_s \cdot E_{sram} + N_m \cdot E_{mac} \quad (1)$$

where the E_{dram}, E_{sram}, and E_{mac} represent the energy cost for each access to DRAM and SRAM, as well as for a single MAC operation, respectively. The N_d, N_s, and N_m correspond to the total number of accesses to DRAM and SRAM, and the overall count of MAC operations utilized, with energy cost values taken from [21]. Since the number of MACs for a specific DNN is constant, the energy efficiency largely depends on $E_{dram/sram}$ as proved by [16].

In the analytical framework proposed by [15], [16], weights, activations, and partial sums (PSUMs) are all stored in INT16 format. However, in certain integer-only quantized DNN accelerators, PSUMs may require higher precision, such as INT32, to maintain accuracy [22], [23]. Consider a neural network where both weights and activations are quantized to INT8 (W8A8). During a multiply-accumulate (MAC) operation, the product of the quantized weight and activation results in an INT16 value. When these products are accumulated along the input channel dimension, denoted as C_i, the accumulation depth plays a crucial role in determining the required bit width to prevent overflow. Specifically, to ensure numerical stability, the PSUM should be stored using $16 + \log_2(C_i)$ bits. For instance, in the Feed-Forward Network (FFN) of BERT-Large [5], where $C_i = 4096$ in the MLP layer, this results in a required PSUM precision of up to 28 bits to accommodate the full accumulation range without overflow. Given that memory hierarchy designs are typically byte-based, PSUM is generally stored with 32-bit precision.

To accommodate this variation, we revise the framework by introducing a precision factor as below:

$$N_{d/s} = Si \cdot N^i_{d/s} + S_w \cdot N^w_{d/s} + \beta \cdot S_o \cdot N^p_{d/s} + S_o \cdot N^o_{d/s} \quad (2)$$

the total number of access to DRAM/SRAM for ifmap, weight, PSUM, and ofmap is defined as $N^i_{d/s}$, $N^w_{d/s}$, $N^p_{d/s}$ and $N^o_{d/s}$, with S_i, S_w and S_o indicating the size of ifmap, weights, and ofmap. The β is the ratio of PSUM precision to that of feature map/weight. For instance, β would be 4 if PSUM is in INT32 for an INT8 DNN. In this work, we merely focus on INT8 DNN with WS or IS, since the OS does not struggle with the precision of PSUM.

1) Input Stationary: The IS-based accelerator keeps the input tiles stationary within the MAC array's registers, facilitating their reuse by weights to update the PSUMs in the output buffer. Similarly, the $N_s^{i/w/p/o}$ is summarized in equation (3):

$$N^w_s = \begin{cases} 1 + \left\lceil \frac{H_i}{P_{ih}} \right\rceil \left\lceil \frac{W_i}{P_{iw}} \right\rceil, & S_w < B_w \\ 2 \left\lceil \frac{H_i}{P_{ih}} \right\rceil \left\lceil \frac{W_i}{P_{iw}} \right\rceil, & S_w \geq B_w \end{cases}, N^i_s = 2$$
$$N^p_s = \begin{cases} 2\left(\left\lceil \frac{C_i}{P_{ci}} \right\rceil - 1 \right), & \frac{C_o}{P_{co}} \cdot \tilde{S}_p < B_o \\ 4\left(\left\lceil \frac{C_i}{P_{ci}} \right\rceil - 1 \right), & \frac{C_o}{P_{co}} \cdot \tilde{S}_p \geq B_o \end{cases}, N^o_s = 2 \quad (3)$$

the weight buffer size is denoted by B_w, and the height and width of the enlarged ifmap are H_i and W_i, respectively. Besides, the ifmap tile is shaped by P_{ih} and P_{iw} following $P_i = P_{ih} \times P_{iw}$. Notably, the PSUM of IS is smaller than that of WS since the ifmap tile reused by weights is

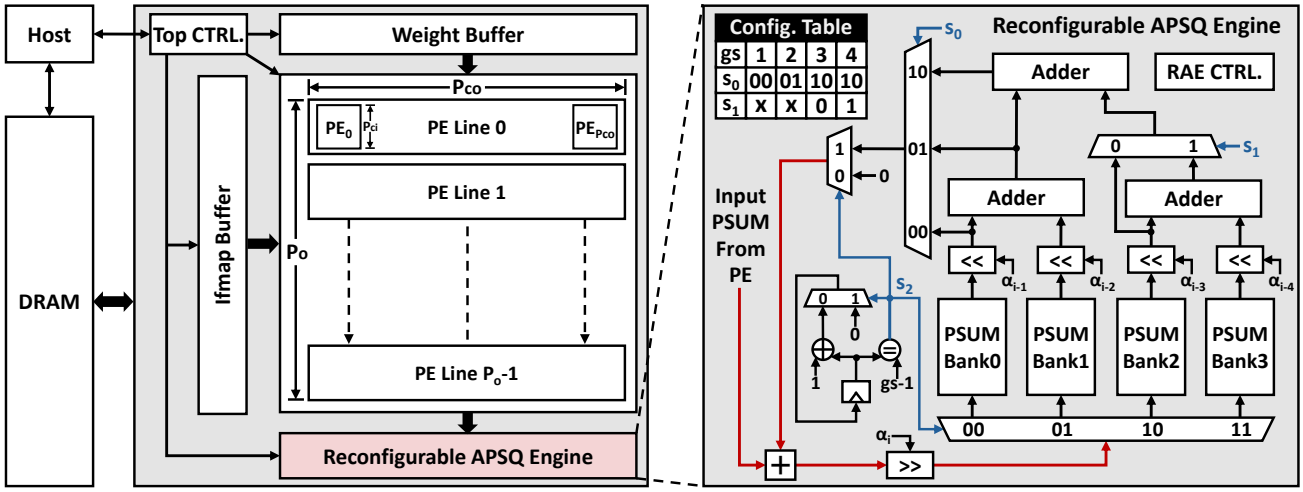


Fig. 2: The analytical DNN accelerator and the Reconfigurable APSQ Engine (RAE) in this work.

generally small. Consequently, we scale the $\tilde{S}_p$ by C_o/P_{co}. The $N_p^{i/w/p/o}$ for DRAM is derived as:

$$N_d^w = \begin{cases} 1, & S_w < B_w \\ \left\lceil \frac{H_i}{P_{ih}} \right\rceil \left\lceil \frac{W_i}{P_{iw}} \right\rceil, & S_w \geq B_w \end{cases}, N_d^i = 1$$
$$N_d^p = \begin{cases} 0, & \frac{C_o}{P_{co}} \cdot \tilde{S}_p < B_o \\ 2\left(\left\lceil \frac{C_i}{P_{ci}} \right\rceil - 1\right), & \frac{C_o}{P_{co}} \cdot \tilde{S}_p \geq B_o \end{cases}, N_s^o = 1 \quad (4)$$

2) Weight Stationary: In contrast, the WS-based accelerator loads ifmap tiles to update the PSUMs via reusing $P_{ci} \times P_{co}$ kernel weights, and the ofmap tile is generated after finishing PSUM accumulation. According to this, the components of N_s can be derived as:

$$N_s^i = \begin{cases} 1 + \left\lceil \frac{C_o}{P_{co}} \right\rceil, & \tilde{S}_i < B_i \\ 2\left\lceil \frac{C_o}{P_{co}} \right\rceil, & \tilde{S}_i \geq B_i \end{cases}, N_s^w = 2$$
$$N_s^p = \begin{cases} 2\left(\left\lceil \frac{C_i}{P_{ci}} \right\rceil - 1\right), & \frac{H_o \cdot W_o}{P_o} \cdot \tilde{S}_p < B_o \\ 4\left(\left\lceil \frac{C_i}{P_{ci}} \right\rceil - 1\right), & \frac{H_o \cdot W_o}{P_o} \cdot \tilde{S}_p \geq B_o \end{cases}, N_s^o = 2 \quad (5)$$

where C_i and C_o represent the dimensions of input and output channels, the B_i and B_o indicate the capacities of ifmap and ofmap in bytes. The ofmap's height and width are denoted as H_o and W_o. Besides, the input tile size $\tilde{S}i$ is enlarged based on output tiles, kernels, and strides, as explained in [16]. The $\tilde{S}_p$ is the size of tiled PSUM calculated as $\beta \times P_o \times P_{co}$, where P_{oh} and P_{ow} are the height and width of the output/PSUM tile, constrained by $P_o = P_{oh} \times P_{ow}$. If exceeding the B_i or B_o, extra read to both the SRAM and DRAM occur, the $N_d^{i/w/p/o}$ is defined as:

$$N_d^i = \begin{cases} 1, & \tilde{S}_i < B_i \\ \left\lceil \frac{C_o}{P_{co}} \right\rceil, & \tilde{S}_i \geq B_i \end{cases}, N_d^w = 1$$
$$N_d^p = \begin{cases} 0, & \frac{H_o \cdot W_o}{P_o} \cdot \tilde{S}_p < B_o \\ 2\left(\left\lceil \frac{C_i}{P_{ci}} \right\rceil - 1\right), & \frac{H_o \cdot W_o}{P_o} \cdot \tilde{S}_p \geq B_o \end{cases}, N_d^o = 1 \quad (6)$$

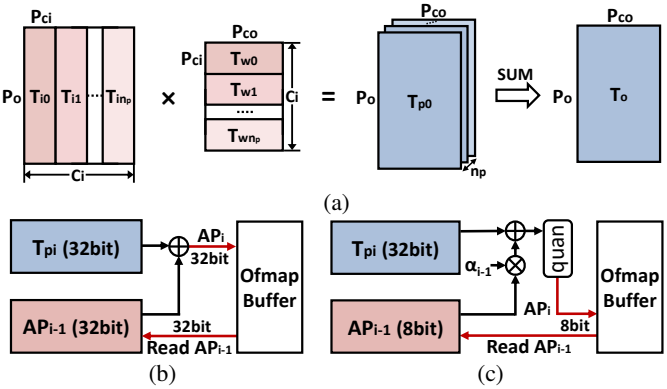


Fig. 3: (a) An example of pointwise convolution with tile-based computation scheme, (b) typical partial sum accumulation, (c) proposed INT8 additive partial sum quantization ($gs = 1$).

B. Quantization

Quantization has emerged as an effective strategy to alleviate computational and memory overheads, particularly advantageous for deployments on hardware accelerators, and can be expressed as:

$$\tilde{x} = \alpha \cdot Q_k(x) = \alpha \cdot \left\lfloor \text{Clip}\left(\frac{x}{\alpha}, Q_n, Q_p\right) \right\rceil \quad (7)$$

where $Q_k(\cdot)$ represents the k-bit quantization function that compresses the high-precision input data to low-bit INT-k with a scaling factor α. Specifically, the $Q_k(x)$ constrains $\frac{x}{\alpha}$ within the range $[-2^{k-1}, 2^{k-1} - 1]$ or $[0, 2^k - 1]$ for the signed and unsigned quantization, bounded by Q_n and Q_p. The dequantized $\tilde{x}$ can be retrieved through re-scaling $Q_k(x)$ by α, which is determined either using the min-max technique [9] or the learnable alternative [10]. In this work, we choose the learnable method LSQ [10] to quantize both weights and activations/PSUMs, among which the scaling factor α of PSUMs are forced to power-of-two format by learning $2^{\lfloor log_2^{\alpha} \rceil}$ via straight through estimator (STE) [24], [25]. In this way, the multiplication in the re-scaling could be replaced by the hardware-efficient shift operation.

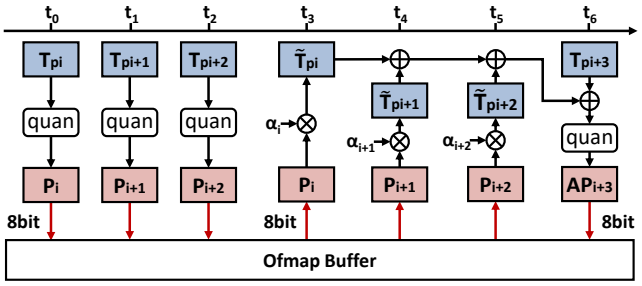


Fig. 4: Workflow of additive partial sum quantization with $gs = 3$.

III. Method

A. Additive Partial Sum Quantization

As highlighted in Section II-A, tile-based computation (TBC) is a primary method in DNN accelerators due to computational limitations. This technique frequently involves additive interactions with PSUMs to produce the final output tile. Defining T_i, T_w, T_p, and T_o as the tiles for ifmap, weight, PSUM, and ofmap severally, a typical PSUM accumulation example with TBC is illustrated in Fig. 3b. The ofmap tile T_o is derived from the accumulation of multiple PSUMs. Thus, the TBC could be formulated as follows:

$$T_o = \sum_{i=0}^{n_p-1} T_{pi}, \;\; n_p = \left\lceil \frac{C_i}{P_{ci}} \right\rceil \tag{8}$$

where n_p signifies the number of PSUM tile T_{pi} for $0 \leq i \leq n_p - 1$. Although TBC ensures computational efficiency, the necessity for high precision (INT32) in PSUMs can still introduce heavy energy overheads, as the numeric results of PSUM accumulation are generally large in Transformers, which is proved in Section IV. Besides, the typical PSUM accumulation could be formulated as follows:

$$AP_j = \sum_{i=0}^{j} T_{pi}, \;\; T_o = AP_j + \sum_{i=j+1}^{n_p-1} T_{pi} \tag{9}$$

where each AP_j represents the additive PSUM after the jth PSUM tile T_{pj}. Since the numeric results of PSUM accumulation are generally large in Transformers, they are required to be stored in INT32, which is proved in Section II-A. To achieve low-bit memory accesses across all APs, we propose an additive PSUM quantization approach termed APSQ, detailed as follows:

$$AP_i = Q_k^i(T_{pi} + \alpha_{i-1} \cdot AP_{i-1}), \;\; \text{for}\, 1 \leq i < n_p. \tag{10}$$

In the APSQ, each AP_i is recursively determined by applying Q_k^i to the sum of T_{pi} and the preceding additive PSUM AP_{i-1} which is depicted in Fig. 3c. The initial additive PSUM AP_0 is set as $Q_k^0(T_{p0})$. The output tile T_o is derived upon completing the quantization of the final PSUM. Consequently, T_o can be succinctly expressed as AP_{n_p-1}. In this way, each APs could be accessed by low-bit precision while the quantizer considers both the current high-precision PSUM and its previous APs.

Algorithm 1 Grouping Strategy

Input: Total number of PSUM tiles n_p, PSUM tile set $T_p = [T_{p0}, \ldots, T_{pn_p-1}]$, scaling factor set $\alpha = [\alpha_0, \alpha_1, \ldots, \alpha_{n_p-1}]$ and the group size gs.
Output: Output tile T_o.

α is stored in the register list, $n_p > 0$ and $AP^*_{i<0} = \mathbf{0}$
$T_o = \alpha_0 \times Q_k^0(T_{p0})$ ▷ Initialize T_o
for $i = 0$ to $n_p - 1$ with step gs **do**
  Read gs previous PSUMs from buffer
  $AP_{i-1} = \sum_{i=i-gs}^{i-1} AP_i^* \times \alpha_i$ ▷ Accumulation
  $AP_i^* = Q_k^i(AP_{i-1} + T_{pi})$ ▷ Perform APSQ
  Write AP_i^* to buffer
  for $j = i + 1$ to $\min(i + gs - 1, n_p - 1)$ **do**
    if $j < n_p - 1$ **then**
      $AP_j^* = Q_k^j(T_{pj})$ ▷ Perform PSUM quantization
      Write AP_j^* to buffer
    else ▷ Final output tile T_o
      Read $n_p - i + 1$ previous PSUMs from buffer
      $T_o = \alpha_{n_p-1} \times Q_k^{n_p-1}(\sum_{l=i}^{n_p-2} AP_l^* \times \alpha_l + T_{pn_p-1})$
    end if
  end for
end for

B. Grouping Strategy

However, the accuracy may suffer since each PSUM interacts with multiple quantizers, leading to heavy approximation errors due to repeated rounding during quantization in APSQ. To mitigate this issue, reducing the quantization frequency is essential, as it can minimize the additional rounding error. Building on this insight, we proposed a grouping strategy that combines APSQ with typical quantization detailed in Algorithm 1. Specifically, the PSUM tile set is partitioned into groups of size gs, where APSQ is applied only once per group, targeting the accumulation of the current input PSUM tile and the previous gs dequantized PSUM tiles. An workflow example of the grouping strategy is depicted in Fig. 4. Each group of gs quantized PSUMs is stored in the Ofmap Buffer using INT8 precision per PSUM and is sequentially read from the buffer for accumulation, with dequantization performed at INT8 precision per PSUM read. Notably, the grouping strategy maintains the same total memory read and write operations for APSQ with both $gs = 1$ and $gs > 1$.

C. Reconfigurable APSQ Architecture

While the grouping strategy offers opportunities to enhance accuracy, an excessively large gs may introduce additional memory overhead. Moreover, different models exhibit varying optimal values for gs, as demonstrated in Table I. To support APSQ with various gs, we designed a Reconfigurable APSQ Engine (RAE) as depicted in Fig. 2. The RAE features a PSUM Buffer with four SRAM banks, shifter-based quantization/dequantization modules, adders, and a dedicated RAE controller. RAE's work mode is governed by static encodings s_0 and s_1, in conjunction with a dynamic encoding s_2. Static

TABLE I: Accuracy comparison between Baseline and APSQ methods across various models and tasks.

Model	Task	Baseline	gs=1	gs=2	gs=3	gs=4
BERT-Base	QNLI	91.32	90.26	90.77	**91.12**	91.03
	MNLI	84.08	82.27	83.12	83.43	**83.54**
	RTE	74.73	74.01	74.01	73.29	**75.81**
	STS-B	87.89	86.94	87.31	87.60	**87.61**
	MRPC	87.99	87.25	87.01	**87.75**	87.01
	CoLA	53.40	50.84	51.27	**52.59**	52.36
Segformer-B0	ADE20K	36.72	35.83	**36.11**	35.97	35.85
EfficientViT-B1		39.48	37.45	**38.65**	38.41	38.47

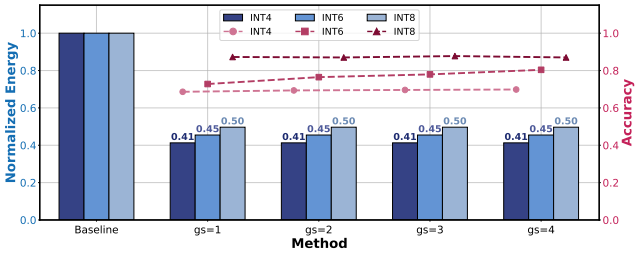


Fig. 5: Normalized energy and accuracy across varied gs settings for MRPC under WS dataflow on BERT-Base.

encodings s_0 and s_1 are derived from a predefined table based on the group size gs, configuring the RAE multiplexers to select appropriate PSUM tiles P_i for different gs. The dynamic encoding s_2 determines whether an APSQ or a PSUM quantization operation is performed.

For instance, when $gs = 1$, s_0 is set to 00, s_1 is irrelevant, and s_2 remains 1. In this mode, each PSUM P_i generated by the PE array prompts the RAE to retrieve the previous INT8 PSUM P_{i-1} from PSUM Bank0, dequantize and accumulate it with P_i. The result is quantized via a shift operation and written back to Bank0. When $gs = 4$, the configuration adjusts: s_0 becomes 10, s_1 is set to 1, and s_2 dynamically toggles. With s_2 set to 0, incoming PSUMs are quantized and stored in the corresponding PSUM Banks without APSQ. After four rounds of PSUM quantization, s_2 switches to 1, triggering the simultaneous retrieval of four PSUMs, P_{i-4} to P_{i-1}, from all four PSUM Banks. These are processed through a two-stage adder pipeline, accumulated with P_i, quantized, and then stored in PSUM Bank3. This flexible architecture allows the RAE to efficiently accommodate various group sizes, effectively replacing conventional PSUM accumulation and storage methods in IS/WS-based DNN accelerators.

IV. Experimental Results

A. Experiment Setup

In this section, we evaluate the model accuracy and hardware performance of the proposed APSQ method across both NLP and CV domains. For the NLP experiments, we adopt the BERT-Base model with an input token length of 128 [5] and benchmark it using the GLUE [26] dataset, which includes tasks such as QNLI, MNLI, RTE, STS-B, MRPC, and CoLA. In the CV domain, we focus on semantic segmentation using the ADE20K [27] dataset, comprising over 20,000 images with 150 distinct classes. We experiment with two models at a 512×512 input resolution: Segformer-B0 [3], which employs a Transformer architecture with vanilla attention, and EfficientViT-B1 [2], a lightweight model that integrates convolution with linear attention. All baseline models in our experiments are quantized to W8A8, and APSQ is incorporated into the quantization-aware training (QAT) process, guided by a full-precision teacher model for knowledge distillation. We employ task-specific metrics from the GLUE benchmark for evaluating NLP tasks and the standard mean intersection over union (mIoU) metric to assess segmentation performance. Furthermore, we apply the proposed techniques to a large language model LLaMA2-7B for zero-shot commonsense reasoning evaluation.

The configuration of the analytical DNN accelerator, as shown in Fig. 2, begins with the allocation of memory buffers: 256KB each for the input feature map (ifmap) and output feature map (ofmap), and 128KB for the weight buffer, while the off-chip DRAM is DDR3. In the MAC array setup, the parallelisms are organized as $P_o = 16$, $P_{ci} = 8$, and $P_{co} = 8$ for BERT-Base, Segformer-B0, and EfficientViT-B1. For large language model (LLM) evaluation, the parallelism settings are adjusted to $P_o = 1$, $P_{ci} = 32$, and $P_{co} = 32$, since the input to the LLM at the decoding stage is a vector, enabling P_o to be set to 1.

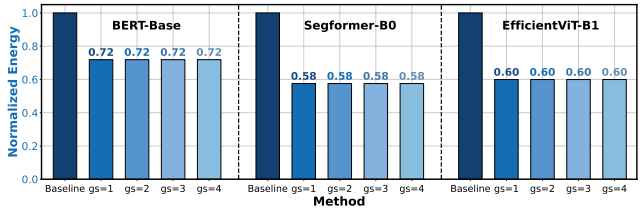
B. Accuracy Analysis

As discussed in Sections III-A and III-B, applying APSQ to all PSUMs (equivalent to $gs = 1$ grouping) significantly degrades performance due to excessive quantization on each PSUM. To validate this, we evaluated accuracy across different gs values in Table I under full INT8 quantization. Results show that $gs = 1$ causes notable accuracy drops, indicating APSQ's insufficiency alone. Although increasing gs generally helps restore accuracy, the improvements are not strictly monotonic. The performance variation across different models and tasks implies that the intrinsic characteristics of each model affect the effectiveness of APSQ, leading to diverse outcomes for different gs values. These observations underscore the necessity of the RAE, which is designed to adapt to varying gs requirements and optimize accuracy accordingly.

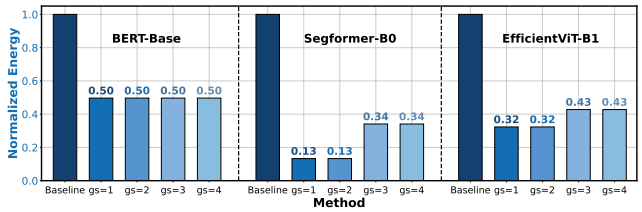
Additionally, as shown in Fig. 5, lower PSUM precision consistently reduces energy consumption. However, energy savings weaken below INT8 precision, indicating that reducing PSUM precision further yields minimal benefits and incurs substantial accuracy loss. Hence, adopting INT8 precision for APSQ is technically optimal. Compared to the BERT-Base baseline, the average accuracy metrics determined from the best results of each individual task decrease by only 0.16%, demonstrating a minimal trade-off. For the semantic segmentation task using Segformer-B0 and EfficientViT-B1, the mIoU decreases by just 0.61% and 0.83% with $gs = 2$, indicating the INT8 setup with a suitable task-specific gs could strike a balance between accuracy and energy savings.

TABLE II: Hardware synthesis resource consumptions

	Area (μm^2)
Baseline DNN Accelerator	1873408
RAE	86410
DNN Accelerator w/ RAE	1933674



(a)



(b)

Fig. 6: Normalized energy across varied gs settings, and models under (a) IS dataflow, (b) WS dataflow.

C. Hardware Performance Analysis

In continuation of the prior analyses on accuracy, we evaluate the energy costs associated with different gs values, following the methods detailed in Section II-A. As depicted in Fig. 6, APSQ achieves normalized energy consumption reductions of 28%, 42%, and 40% for BERT-Base, Segformer-B0, and EfficientViT-B1, respectively, in IS dataflow. The gs parameter has minimal influence on energy savings for IS, given the typically small PSUM tile size. However, its effect becomes more significant in WS dataflow, where large PSUMs must be buffered. For WS, varying gs values lead to a uniform 50% energy reduction in BERT-Base, owing to its short token length. In contrast, the impact varies for semantic segmentation models: Segformer-B0 maintains an 87% energy saving at $gs = 1$ and $gs = 2$, but this benefit declines to 66% at $gs = 3$ and $gs = 4$. Similarly, EfficientViT-B1 achieves a 68% energy reduction for $gs = 1$ and $gs = 2$, decreasing to 57% at higher gs values. This decline occurs as larger PSUM sizes surpass buffer capacity at higher gs, particularly with high-resolution input, resulting in additional DRAM operations.

Consequently, when the PSUM is fully stored on-chip, energy consumption primarily depends on the PSUM precision. However, when the buffer size becomes insufficient, as observed in Segformer-B0 and EfficientViT-B1 under WS, gs significantly impacts energy consumption. This trade-off between accuracy and energy efficiency emphasizes the necessity of a reconfigurable architecture that can dynamically adjust to various gs settings, underscoring the essential role of the RAE. We implement a DNN accelerator with and without RAE using Verilog HDL and obtain the hardware resources via Synopsys Design Compiler at 28-nm technology with a frequency constraint of 250 MHz. The accelerator without RAE served as the baseline. As presented in Table II, the accelerator with RAE incurred only a 3.21% increase in the synthesized area while achieving a 28-87% reduction in energy consumption as discussed before.

D. Experiment on Large Language Model

TABLE III: Accuracy of Baseline and APSQ across seven Zero-shot Common Sense Reasoning tasks on LLaMA2-7B.

Method	**BoolQ**	**PIQA**	**HellaS.**	**WinoG.**	**Arc-e**	**Arc-c**	**OBQA**
Baseline	77.80	79.22	76.64	69.69	75.25	47.10	43.40
gs=1	75.26	76.82	72.99	65.75	71.38	42.58	38.60
gs=2	75.93	77.09	74.94	67.48	73.86	46.42	42.00
gs=3	76.45	**78.84**	75.43	**68.43**	73.40	47.18	41.80
gs=4	**76.82**	78.45	**76.01**	67.96	**74.75**	**47.35**	**42.80**

TABLE IV: Normalized energy across varied gs settings under IS and WS on LLaMA2-7B.

Dataflow	**Baseline**	**gs=1**	**gs=2**	**gs=3**	**gs=4**
IS	1.02×	1×	1×	1×	1×
WS	31.7×	1×	1×	8.42×	8.42×

Nowadays, large language models (LLMs) represent a significant advancement in the field of AI. The autoregressive nature of LLMs leads to only one token being inferred at a time during the generation phase. To simulate this scenario while keeping the total number of MAC operations unchanged, we adjust the parallelisms to $P_o = 1$, $P_{ci} = 32$, and $P_{co} = 32$. We implement APSQ within the RoLoRA framework [28] and apply W8A8 quantization for LLaMA2-7B as the baseline. Experiments are conducted on seven Zero-shot Common Sense Reasoning (ZCSR) tasks [29]. As shown in Table III, APSQ results in an average accuracy reduction of only 0.59% when compared to the baseline, evaluated using the best accuracy for each individual task. Despite this small accuracy trade-off, it achieves substantial energy savings of 1.02-$31.7\times$ with 4096 sequence length considering both prefilling and decoding stages, as shown in Table IV, demonstrating APSQ's significant potential in LLM applications. The minimal enhancement of APSQ on IS is primarily due to the feature map being a vector, which is considerably smaller than weight. Further explorations will be conducted in our future work.

V. Conclusion

This work highlights the significant energy overhead from high-precision PSUM accesses in DNN accelerators, often overlooked in prior research. To address this, we first refine an PSUM-precision-aware analytical framework to evaluate the energy consumption of DNN accelerators. Then, we propose a novel APSQ method enhanced by a grouping strategy and a reconfigurable architecture. Experimental results demonstrate the intricate relationship between group size and PSUM quantization overhead. The INT8 APSQ implementation achieves minimal accuracy loss on the GLUE and ADE20K datasets while significantly reducing energy costs. Further experiments also reveal APSQ's potential for efficient acceleration of LLMs. These findings provide a strategic framework for optimizing DNN accelerators that harness APSQ.

REFERENCES

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[2] H. Cai, J. Li, M. Hu, C. Gan, and S. Han, "Efficientvit: Lightweight multi-scale attention for high-resolution dense prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 17 302–17 313.

[3] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 077–12 090, 2021.

[4] Y. Quan, D. Zhang, L. Zhang, and J. Tang, "Centralized feature pyramid for object detection," *IEEE Transactions on Image Processing*, 2023.

[5] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[6] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[9] S. Kim, A. Gholami, Z. Yao, M. W. Mahoney, and K. Keutzer, "I-bert: Integer-only bert quantization," in *International conference on machine learning*. PMLR, 2021, pp. 5506–5518.

[10] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," in *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[11] S.-Y. Liu, Z. Liu, and K.-T. Cheng, "Oscillation-free quantization for low-bit vision transformers," *arXiv preprint arXiv:2302.02210*, 2023.

[12] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, and J. Sun, "Metapruning: Meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3296–3305.

[13] C. Shu, Y. Liu, J. Gao, Z. Yan, and C. Shen, "Channel-wise knowledge distillation for dense prediction," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 5311–5320.

[14] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 269–284, 2014.

[15] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[16] F. Tu, S. Yin, P. Ouyang, S. Tang, L. Liu, and S. Wei, "Deep convolutional neural network architecture with reconfigurable computation patterns," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 8, pp. 2220–2233, 2017.

[17] H. You, Z. Sun, H. Shi, Z. Yu, Y. Zhao, Y. Zhang, C. Li, B. Li, and Y. Lin, "Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design," in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2023, pp. 273–286.

[18] L. Lu, Y. Jin, H. Bi, Z. Luo, P. Li, T. Wang, and Y. Liang, "Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 977–991.

[19] A. Azamat, F. Asim, J. Kim, and J. Lee, "Partial sum quantization for reducing adc size in reram-based neural network accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[20] J. Bai, W. Xue, Y. Fan, S. Sun, and W. Kang, "Partial sum quantization for computing-in-memory based neural network accelerator," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2023.

[21] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*. IEEE, 2014, pp. 10–14.

[22] S. K. Lee, P. N. Whatmough, M. Donato, G. G. Ko, D. Brooks, and G.-Y. Wei, "Smiv: A 16-nm 25-mm² soc for iot with arm cortex-a53, efpga, and coherent accelerators," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 2, pp. 639–650, 2022.

[23] S. Ryu, H. Kim, W. Yi, E. Kim, Y. Kim, T. Kim, and J.-J. Kim, "Bitblade: Energy-efficient variable bit-precision hardware accelerator for quantized neural networks," *IEEE Journal of Solid-State Circuits*, vol. 57, no. 6, pp. 1924–1935, 2022.

[24] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," *arXiv preprint arXiv:1308.3432*, 2013.

[25] P. Dong, Y. Tan, D. Zhang, T. Ni, X. Liu, Y. Liu, P. Luo, L. Liang, S.-Y. Liu, X. Huang *et al.*, "Genetic quantization-aware approximation for non-linear operations in transformers," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[26] A. Wang, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[27] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 633–641.

[28] X. Huang, Z. Liu, S.-Y. Liu, and K.-T. Cheng, "Rolora: Fine-tuning rotated outlier-free llms for effective weight-activation quantization," *arXiv preprint arXiv:2407.08044*, 2024.

[29] L. Gao, J. Tow, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, K. McDonell, N. Muennighoff *et al.*, "A framework for few-shot language model evaluation," *Version v0. 0.1. Sept*, vol. 10, pp. 8–9, 2021.

ReChisel: Effective Automatic Chisel Code Generation by LLM with Reflection

Juxin Niu[1], Xiangfeng Liu[1,2], Dan Niu[3], Xi Wang[4,5*], Zhe Jiang[4,5], Nan Guan[1*]
[1] Department of Computer Science, City University of Hong Kong, Hong Kong SAR
[2] School of Computer Science and Engineering, Northeastern University, China
[3] School of Automation, Southeast University, China
[4] National Center of Technology Innovation for EDA, China [5] School of Integrated Circuits, Southeast University, China
Email: juxin.niu@my.cityu.edu.hk, xiangfengliu.cn@gmail.com, 101011786@seu.edu.cn, xi.wang@seu.edu.cn,
zhejiang.arch@gmail.com, nanguan@cityu.edu.hk
*Corresponding authors.

***Abstract*—Coding with hardware description languages (HDLs) such as Verilog is a time-intensive and laborious task. With the rapid advancement of large language models (LLMs), there is increasing interest in applying LLMs to assist with HDL coding. Recent efforts have demonstrated the potential of LLMs in translating natural language to traditional HDL Verilog. Chisel, a next-generation HDL based on Scala, introduces higher-level abstractions, facilitating more concise, maintainable, and scalable hardware designs. However, the potential of using LLMs for Chisel code generation remains largely unexplored. This work proposes ReChisel, an LLM-based agentic system designed to enhance the effectiveness of Chisel code generation. ReChisel incorporates a reflection mechanism to iteratively refine the quality of generated code using feedback from compilation and simulation processes, and introduces an escape mechanism to break free from non-progress loops. Experiments demonstrate that ReChisel significantly improves the success rate of Chisel code generation, achieving performance comparable to state-of-the-art LLM-based agentic systems for Verilog code generation.**

I. Introduction

In hardware design, coding with hardware description languages (HDLs) like Verilog is often a time-consuming and labor-intensive task. Large language models (LLMs) have demonstrated promising capabilities in assisting the coding processing in not only software but also hardware design. Recently, LLMs have been employed for translating natural language to HDL code [11], [25], assisting with debugging [27], [33], and optimizing code implementation to improve key chip metrics such as performance, power, and area (PPA) [34]. These efforts primarily focus on traditional HDLs like Verilog.

Chisel [2] is a next-generation HDL embedded in Scala that facilitates advanced circuit design. Unlike Verilog, which primarily targets the Register Transfer Level (RTL) and often necessitates detailed low-level specifications, Chisel offers higher-level abstractions through object-oriented and functional programming features. This approach results in more concise, maintainable, and scalable designs [10]. Due to its compatibility with agile design methodologies, Chisel is gaining popularity for complex hardware design [13], including processors (e.g., XiangShan [31], BOOM [36], and RocketChip [1]), accelerators (e.g., MAGMA-Si [6] and Gemmini [8]), among many others.

However, the potential of using LLMs for Chisel code generation remains largely unexplored. As a relatively new language, Chisel has significantly less publicly available code online—for instance, the amount of Chisel code on GitHub is only about 1/20th that of Verilog—which limits the training data available for LLMs to generate Chisel code effectively. To better understand these limitations, we conducted experiments to evaluate the baseline performance of mainstream LLMs in automatically generating Chisel code in a zero-shot setting. The experiment results show that the baseline capabilities of these LLMs are indeed limited.

In this paper, we propose ReChisel, an LLM-based agentic system designed to improve the effectiveness of using LLM for Chisel code generation. ReChisel incorporates a reflection mechanism, utilizing an iterative process to refine the quality of generated code. The key insight is leveraging feedback from compilation and simulation processes to guide the LLM in correcting errors. Additionally, we observed that during the reflection process, the LLM may enter a non-progress loop. To overcome this, we developed an escape mechanism that enables the LLM to break free from these loops and resume making useful attempts. We evaluate the performance of ReChisel on three benchmarks, VerilogEval's Spec-to-RTL [21], AutoChip's HDLBits [9] and RTLLM [17], with five mainstream LLMs (various versions of GPT-4 and Claude 3.5). The experimental results show that our method significantly improves the success rate of generating Chisel code. Additionally, we compare the performance of LLMs in generating Chisel versus Verilog code. Although the baseline capabilities of LLMs are significantly weaker for Chisel code generation than for Verilog, the performance of ReChisel reaches a level comparable to the state-of-the-art LLM-based agentic systems for Verilog code generation. This not only validates the effectiveness of our proposed method but also highlights the potential of Chisel in LLM-assisted hardware design. ReChisel is open-sourced at https://github.com/niujuxin/rechisel.

II. Background & Related Work

A. LLM-assisted Hardware Design

In hardware design, coding with hardware description languages (HDLs) like Verilog is a time-consuming and labor-intensive task. Recent advancements in LLMs have demonstrated transformative potential in diverse domains, sparking increasing interest in leveraging LLMs to streamline and improve the hardware design process.

Some research [4] aims at automating the end-to-end hardware design process using LLMs. ChatCPU [28] introduces the first agile hardware design and verification platform, and develops a 6-stage pipelined RISC-V CPU prototype with the platform. Other studies [3], [20], [25], [30] focus on enhancing the effectiveness of HDL code generation through LLMs. RTLCoder [16] presents an agent-based framework for automatically generating labeled datasets, which are then used to fine-tune a lightweight model that achieves performance comparable to GPT models. RTLRewriter [34] leverages LLMs for code optimization to enhance key chip metrics such as power, performance, and area (PPA). Some work [24] emphasizes assessing the performance of LLMs in hardware design tasks. RTLLM [17], [19] provides an open-source RTL generation benchmark consisting of 50 design problems that span a diverse and broad

TABLE I: Comparison of LLM baseline capabilities in generating Chisel and Verilog code.

Model	Pass@1 (%)		Pass@5 (%)		Pass@10 (%)	
	CHS	VRL	CHS	VRL	CHS	VRL
GPT-4 Turbo	45.54	67.61	61.97	77.46	66.20	81.22
GPT-4o	45.07	69.48	65.26	75.59	70.89	77.46
GPT-4o mini	11.27	59.15	28.64	69.48	36.62	72.30
Claude 3.5 Sonnet	33.33	77.93	52.58	82.16	59.62	84.04
Claude 3.5 Haiku	26.29	75.59	54.46	83.57	58.69	84.04

* CHS and VRL represent Chisel and Verilog, respectively.

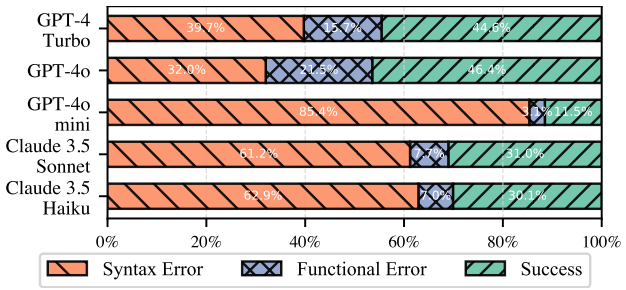


Fig. 1: Proportion of different error types in generated Chisel code.

spectrum of complexities. VerilogEval [15], [21] provides benchmarks for two tasks: code completion and translating specifications to RTL. MG-Verilog dataset [35] divides module implementations into multiple abstraction levels, enabling the evaluation of LLM capabilities across varying degrees of abstraction. These works focus on Verilog, leaving the effectiveness of using LLMs for Chisel code generation an open question yet to be explored.

B. Chisel: Next-Generation HDL

Chisel [2] is a next-generation HDL embedded in Scala, leveraging Scala's advanced syntax and features to enable high-level circuit design. Unlike traditional HDLs like Verilog, which focus on register-transfer level (RTL) and require detailed low-level specifies, Chisel introduces modern programming paradigms such as object-oriented design (OOD) and functional programming. This allows for modular, reusable, and parameterized hardware design [13]. Chisel is gaining popularity for complex hardware design [13], including processors (e.g., XiangShan [31], BOOM [36], and RocketChip [1]), accelerators (e.g., MAGMA-Si [6] and Gemmini [8]), among many others. ChatChisel [18] leverages LLMs to generate Chisel code, designing and implementing a RISC-V CPU with a five-stage pipeline and branch prediction. This work shows the advantages of using LLMs with Chisel in complex hardware design scenarios, highlighting Chisel's potential of using LLM to assist in hardware generation.

Chisel is not directly used for hardware implementation; instead, it serves as a high-level language that must first be *compiled* into Verilog [22]. The compilation process begins with the execution of Chisel code, using the Chisel library to construct a hardware design graph. This design is then serialized into a platform-independent intermediate representation known as FIRRTL [12], [14]. The FIRRTL representation is optimized and ultimately transformed into Verilog code through the FIRRTL compiler.

C. Reflection

Reflection [23], [29], [32] enables LLMs to self-analyze their reasoning processes and actions, dynamically adapting to enhance reliability, adaptability, and effectiveness. In the context of code generation, reflection allows LLMs to automatically identify syntax errors or logical flaws in generated code. AutoChip [26] leverages reflection to improve the performance of Verilog code generation. VerilogCoder [11] introduced an autonomous agent capable of performing graph planning, and integrates an AST-based waveform tracing tool to provide feedback for Verilog code debugging.

III. Motivation

Chisel's high-level abstractions and modern programming principles make it well-suited for agile hardware design and increasingly popular for RTL design. However, it remains unclear whether Chisel is also a viable choice for automatic RTL code generation using LLMs. Should we use LLMs to directly generate Verilog code from a natural language specification, or should we generate Chisel code and then compile it into Verilog? As a first step towards answering this question, this paper explores the capability of LLMs to generate *correct* Chisel code.

Compared to Verilog, Chisel presents both potential advantages and disadvantages for LLM-based automatic code generation. On the downside, Chisel is relatively new and has significantly less publicly available code for model training—approximately only $1/20^{\text{th}}$ the volume of Verilog code on GitHub [1]. On the upside, Chisel's similarity to software languages and its incorporation of modern programming concepts enables LLMs to focus on higher-level logic in complex scenarios rather than low-level specifications. Additionally, the strong capabilities of LLMs in handling software code like Java could potentially transfer to Chisel code generation. Given these factors, it remains uncertain whether Chisel can be a competitive choice for automatic RTL code generation using LLMs.

As a preliminary step toward understanding LLMs' performance in Chisel code generation, we assessed mainstream LLMs' *baseline* capabilities in generating Chisel code, and compared it with the more widely studied Verilog to identify potential performance gaps. The evaluation was performed under *zero-shot* conditions, providing the LLMs with only the specification without additional context. Each model was tasked with producing *complete* module implementations in either Verilog or Chisel in a *single* pass. We evaluate using the widely adopted Pass@k [5] metric, which measures the probability of achieving at least one correct result among the top k generated attempts. Results in Table I show that the ability of LLMs in HDL code generation to use Chisel is significantly less effective than Verilog. We further analyze the proportions of different types of errors that occur when LLMs generate Chisel code. As illustrated in Fig. 1, a significant proportion of errors arise during the compilation phase, which indicates that these LLMs still struggle with ensuring the syntactic correctness of Chisel code. This huge performance gap highlights the necessity of exploring new methods to enhance LLMs' ability to generate Chisel code.

IV. ReChisel Design

A. Overview

We introduce ReChisel, an LLM-based agentic system designed to enhance the effectiveness of Chisel code generation. Fig. 2 illustrates the workflow of ReChisel. Given a specification and a testbench, the goal is to generate correct Chisel code that passes all tests.

The workflow begins with the *Generator* creating Chisel code based on the specification (**Step ❶**). The *Compiler* then translates this code into Verilog (**Step ❷**), followed by functional testing of the Verilog code using the *Simulator* (**Step ❸**). If both compilation and

[1]As of March 2025, our GitHub advanced search revealed 1.60 million lines of publicly available Verilog/SystemVerilog code and 80.1 thousand lines of Scala code importing Chisel extensions.

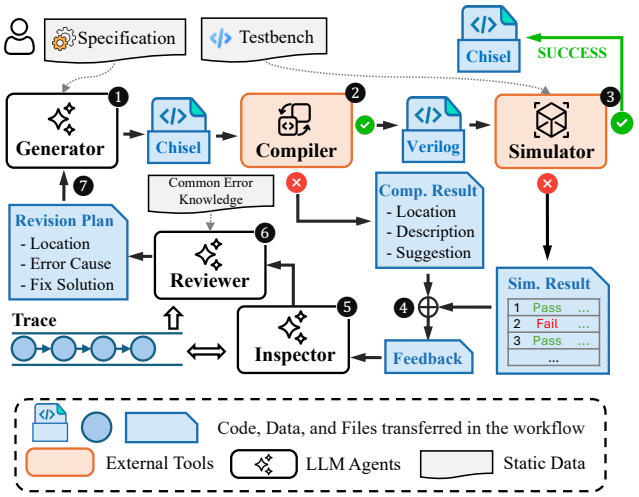


Fig. 2: ReChisel workflow: ReChisel is an LLM-based agentic system designed to enhance the effectiveness of Chisel code generation.

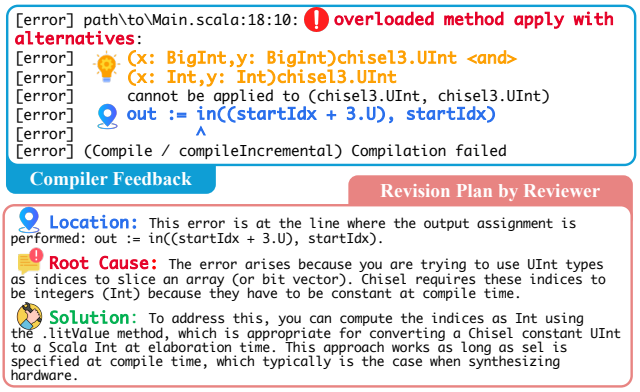


Fig. 3: Example of compiler feedback and revision plan.

simulation succeed, the Chisel code is considered correct, concluding the workflow, and the user receives the validated code. However, if errors occur, ReChisel employs a reflection mechanism to iteratively generate code until it succeeds. In each iteration, results are collected and organized to provide feedback to the *Inspector* (**Step ❹**). The Inspector maintains a trace that includes all previous iterations. Upon receiving the feedback, the Inspector updates the trace (**Step ❺**). The *Reviewer* then analyzes the trace to formulate a revision plan (**Step ❻**), which is passed back to the *Generator* as guidance for error correction. The Generator applies the necessary code adjustments according to the plan to produce a new version of the Chisel code (**Step ❼**). The updated code re-enters the workflow, initiating the next iteration. The Generator, Reviewer, and Inspector are LLM agents whose roles in the workflow are defined by the provided system prompts, while the Compiler and the Simulator are external tools.

A *maximum number of iterations* is set to limit the number of attempts the LLM can make. If the number of iterations exceeds this limit, the workflow terminates, and the case is considered a failure. The number of iterations is a key factor affecting the effectiveness of ReChisel. In general, a higher number of iterations typically leads to better performance. However, the performance gains from increasing the number of iterations is limited by the inherent capabilities of the LLM. In Section V, we provide a detailed analysis of how the number of iterations impacts the effectiveness of ReChisel.

To enhance the reflection process, we propose different feedback strategies for different types of results. We also adopt in-context learning methods for more effective reviews. These approaches are detailed in Section IV-B. Additionally, we observed that the LLMs may fall into a *non-progress loop* during reflection. To address this issue, we propose an *escape* mechanism implemented within the Inspector. The detailed discussion is given in Section IV-C.

B. Error Feedback & Reflection Mechanism

Errors in testing LLM-generated Chisel code fall into two types: *syntax errors* reported in compilation and *functional errors* discovered in simulation. We propose different feedback strategies for each type.

Syntax Error. During the compilation process, the compiler conducts a series of checks to ensure that the code is syntactically correct and can be successfully translated. When errors are detected, the compiler reports the specific location of each issue, provides a detailed explanation of the error and, in some cases, suggests possible fixes. This information is organized into an error list and serves as feedback. Fig. 3 illustrates an example of compiler feedback.

We found that the types of syntax errors tend to be densely distributed. Specifically, the most common errors generated by the LLM include mixing Scala and Chisel syntax, handling signal types, and managing clock domains. In Table II, we show several examples of common errors made by the LLM along with the corresponding feedback provided by the compiler. Beyond syntax errors, we found that the compiler performs static code analysis during compilation and reports logical errors that can cause functional errors in simulations. For instance, the compiler can detect uninitialized or partially initialized signals (B3), preventing unintended latch generation in hardware. Besides, it can identify combinational loops (C2) to avoid unpredictable signal oscillations. For simplicity, we also refer to these logical errors detected at compile time as syntax errors in the future discussion. For these common errors, we employ in-context learning [7] to further enhance the effectiveness of reviews. We pre-organized the causes and corresponding fix guidance for each error and included this information within the prompts provided to the LLM. This approach enables the LLM to deliver feedback more quickly and accurately when encountering such errors.

Functional Error. After compiling the Chisel code into Verilog, the Verilog code is passed to the simulator as the Device Under Test (DUT). In addition to the DUT, the simulation process includes a reference module and a series of functional points. Each functional point consists of input stimuli and the expected output. During simulation, input stimuli are applied to both the DUT and the reference module, and their outputs are compared. If the outputs match, the test passes; otherwise, it fails. For all failed test points, we extract the input signals, the expected output signals, and the actual obtained signals to create an error list, which is used as feedback.

Feedback for the two aforementioned error types is provided to the LLM alongside the Chisel code for reflection. The LLM generates a revision plan that, for each error, includes the error location, a cause analysis, and specific solutions. Fig. 3 also illustrates a revision plan based on the compiler feedback shown in the figure. The revision plan guides the LLM in making adjustments to the Chisel code.

C. Non-progress Loop & Escape Mechanism

During the iterative process of code optimization through reflection, LLMs may fall into a *non-progress loop*, i.e., the LLM repeatedly cycles through the same set of errors without resolving them, and is unable to break free from the error cycle. We use Fig. 4 to illustrate this problem. The code error lies within the `switch` block where not all possible cases of `y` are handled, leading to

TABLE II: Common syntax errors in LLM-generated Chisel code and corresponding compiler feedback.

Description	Incorrect	Corrected	Compiler Feedback
A. Structural Errors			
1. Misspelling, unmatched parentheses, etc.	`val signal=Wire(...)` `signal:=0.U`	`val signal=Wire(...)` `signal:=0.U`	`Value signal is not a member. Did you mean signal?`
2. Mixed usage of Chisel and Scala syntax.	`out := in.asInstanceOf[SInt]`	`out := in.asSInt`	`class chisel3.UInt cannot be cast to class chisel3.SInt.`
3. Incorrect invocations of functions or methods.	`val r=Seq.fill(5)(...)` `r(0, 2)`	`val r=Seq.fill(5)(...)` `r(2)`	`Too many arguments. Found 2, expected 1 for method apply:(i:Int)`
B. Signal Definition, Usage and Typing Errors			
1. Incorrect definition of clock or reset signals.	`val rst = IO(Input(Reset()))`	`val rst = IO(Input(Bool()))`	`A port rst with abstract reset type was unable to be inferred by InferResets.`
2. Failure to encapsulate interface signals within `IO()`.	`val clk=Input(Clock())`	`val clk=IO(Input(Clock))`	`Clock must be hardware, not a bare Chisel type. Perhaps you forgot to wrap it in Wire(_) or IO(_)?`
3. Wire signal not (fully) initialized.	`when (io.in) {w:=0.U}` `// ?`	`when (io.in) {w:=0.U}` `otherwise {w:=1.U}`	`Reference w not fully initialized.`

4. Bundle connection mismatch.	`val a = OneBdl(...)`
	`val b = AnotherBdl(...)`
	`a := b`
		`// assignment not allowed`	`Connection between sink (... OneBdl) and source (... AnotherBdl) failed: .cSource Record missing field (c).`

5. Signal type mismatch.	`val cnt=oks.reduce(_ +& _)`	`val cnt=oks.map(_.asUInt).reduce(_ +& _)`	`found: chisel3.Bool required: chisel3.UInt`
6. Unsupported signal type conversion or casting.	`invertedClk := (~ clk.asUInt).asClock`	`// unsupported`	`Value asClock is not a member of chisel3.UInt.`
7. Out-of-bounds access on array-type signal.	`vector(-1) := 0.U`	`vector(0) := 0.U`	`-1 is out of bounds. (min 0, max 3)`
C. Miscellaneous Errors			
1. Incorrect Clock Domain * (in multi-clock design)	`val out=RegNext(in)`	`val out=withClock(clk) { RegNext(in)}`	`No implicit clock.`
2. Combinational Loop	`val a=Wire(...)` `a := a + 1.U`	`// remove loop`	`Detected combinational cycle in a FIRRTL module. Sample path: {a ← a_T_1 ← ··· ← a_T ← ··· ← a}.`

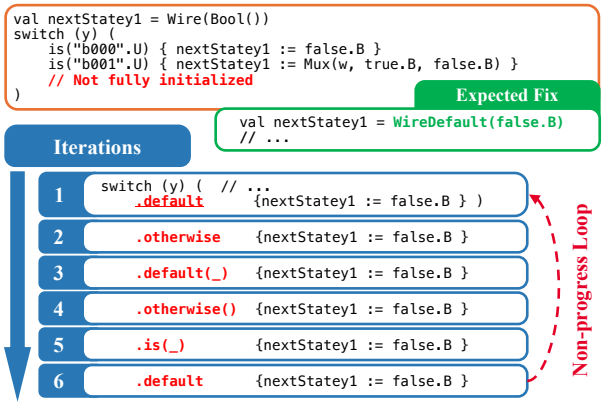


Fig. 4: Example of non-progress loop. The LLM is expected to provide default values when defining the signal to ensure fully initialization. However, it repeatedly attempts to add default statements within the `switch` block.

`nextStatey1` being uninitialized when `y` takes unspecified values. The correct approach to fix this is to use `WireDefault` when defining the signal to provide a default value. However, in the first reflection iteration, the LLM tried to add a default case to the `switch` block incorrectly, as Chisel's `switch` does not support any default case syntax. Subsequently, in each iteration, the LLM fell into the trap of repeatedly attempting to add a default case to the `switch` block, resulting in a non-progress loop.

The non-progress loops happen mainly due to the greedy strategy during reflection and the limitations of the LLM's context. In the example in Fig. 4, after the first iteration fails, the compiler issues an

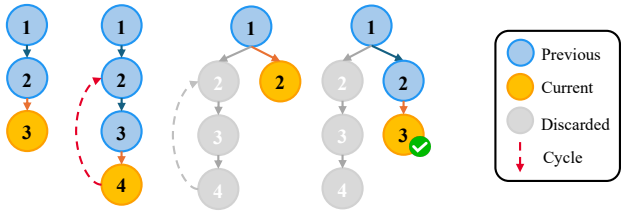


Fig. 5: Illustration of escape mechanism.

error message indicating unsupported syntax in the `switch` block. Affected by this error, the reviewer focuses solely on correcting issues inside the `switch` block in subsequent attempts, ignoring the true location of the error which can only be captured from a global perspective. Additionally, the LLM has a limited context length, which means that as the number of iterations increases, early error experiences are completely forgotten, leading the LLM to return to its previous state after multiple attempts.

To address the non-progress loop issue, we propose an *escape* mechanism that monitors the LLM's iterative process for potential cycles. During each iteration, the trace and the current feedback are provided to the Inspector for analysis to detect any cycles. Specifically, the feedback is compared with each entry in the trace. If the current feedback contains an error at the same location as a previous entry's error, and the LLM determines that the causes of these errors are identical, then all iterations between these two points form a non-progress loop. In such cases, the LLM discards all iterations involved in the loop. The Reviewer then regenerates a revision plan starting from the step immediately preceding the loop. With the inherent diversity, the LLM is expected to break out of the

TABLE III: ReChisel performance

Metric	LLMs	Success Rate (%) $n=$ 0	1	5	10
Pass@1	GPT-4 Turbo	**45.54**	52.11	67.61	73.24
	GPT-4o	**45.07**	**56.81**	73.24	77.46
	GPT-4o mini	11.27	16.43	31.46	40.38
	Claude 3.5 Sonnet	33.33	**63.38**	**80.28**	**84.98**
	Claude 3.5 Haiku	26.29	56.34	**79.81**	**84.51**
Pass@5	GPT-4 Turbo	**61.97**	68.54	80.28	83.10
	GPT-4o	**65.26**	75.59	83.10	85.45
	GPT-4o mini	28.64	37.56	54.93	62.91
	Claude 3.5 Sonnet	52.58	**77.46**	**91.08**	**92.49**
	Claude 3.5 Haiku	52.11	**76.53**	**90.14**	**91.08**
Pass@10	GPT-4 Turbo	**66.20**	72.77	84.04	85.92
	GPT-4o	**70.89**	79.81	85.92	88.73
	GPT-4o mini	36.62	45.54	61.03	67.61
	Claude 3.5 Sonnet	59.62	**83.10**	**92.02**	**93.43**
	Claude 3.5 Haiku	58.69	**82.63**	**91.55**	**92.96**

* n represents the maximum allowed number of iterations.
** The top-1 and top-2 entries in each metric and n group is highlighted using dark and light green shading, respectively.

loop and potentially find the correct fix method. Fig. 5 illustrates this escape mechanism.

V. Evaluation

This section is organized as follows: Section V-A provides the experimental setup. Section V-B evaluates the performance of ReChisel. Section V-C compares ReChisel with AutoChip, an LLM-based automatic Verilog code generation system. Finally, Section V-D presents a case study to demonstrate the workflow of ReChisel.

A. Experimental Setup

Benchmark. The evaluation is conducted at the module level, where each test case comprises a module specification that includes a functional description and definitions of I/O signals. Our benchmark specifications are sourced from the VerilogEval's Spec-to-RTL [21], AutoChip's HDLBits [9], and RTLLM [17] dataset. While most test cases align with our evaluation criteria, some are invalid or incompatible. In particular, issues arise when: 1) the case requires using certain Verilog features such as the parameterization[2], which are not supported by Chisel. As a result, it is impossible to write Chisel code that compiles into Verilog code incorporating these features; 2) the correctness of generated code cannot be verified due to missing or incorrect reference code; or 3) the case is designed specifically for Verilog code debugging or completion and cannot be adapted for Chisel code generation tasks. After filtering out these cases, we finalized a set of 216 valid test cases.

Model. Five mainstream LLMs are selected for evaluation: GPT-4 Turbo (version: 2024-04-09), GPT-4o (version: 2024-08-06), GPT-4o mini (version: 2024-07-18), and Claude 3.5 Sonnet (version: 2024-10-22), Claude 3.5 Haiku (version: 2024-10-22).

Evaluation Parameters. LLMs are probabilistic, meaning the same prompt can generate different outputs in multiple attempts. Therefore, each case is tested ten times, and the Pass@k [5] metric is calculated. The LLM is evaluated in its default configuration without manual adjustments to parameters such as temperature and top-p. The maximum number of iterations is capped at 10.

B. Evaluation on ReChisel

Table I demonstrates the performance improvements achieved by ReChisel. The results indicate that ReChisel significantly enhances the performance of LLMs in generating Chisel code, with improvements ranging from 10% to 50%. The four LLMs—GPT-4 Turbo, GPT-4o, Claude 3.5 Sonnet, and Claude 3.5 Haiku—exhibit very similar peak performances, while GPT-4o mini's performance is notably inferior. Among these, Claude 3.5 Sonnet achieved the highest final success rates, with 84.98% under Pass@1 and 93.43% under Pass@10. Notably, although Claude 3.5 Haiku is much more lightweight than Sonnet, its performance is nearly on par with that of Sonnet, and it even surpasses GPT-4o and GPT-4 Turbo.

Fig. 6 further illustrates how the success rates of LLMs change as the number of iterations increases. As shown in the figure, the two Claude LLMs demonstrate much stronger reflection capabilities. Taking Sonnet as an example, under the Pass@1 metric, its baseline accuracy ($n = 0$) is only 33%, significantly lower than that of GPT-4 Turbo or GPT-4o. However, after a few iterations, its performance improves substantially, eventually surpassing GPT-4 Turbo and GPT-4o at $n = 10$. After about four iterations, all LLMs except GPT-4o mini can reach a performance plateau. In contrast, GPT-4o mini's performance improves more slowly as the number of iterations increases. After reaching the plateau, about 10% of cases still cannot be generated correctly, highlighting the inherent limitations of LLMs.

We also examine how the number of syntax and functional errors changes with increasing iterations. In Fig. 7, we present the proportions of syntax and functional errors for GPT-4o under the Pass@1 metric across each iteration. Compared to the baseline, ReChisel reduces both syntax and functional errors. However, we observed that in some cases (e.g., from the 9th to the 10th iteration in the figure), the proportion of syntax errors increased. This suggests that when the LLM fixes functional errors, it may inadvertently reintroduce syntax errors.

TABLE IV: Performance comparison of ReChisel and AutoChip.

Metric	LLMs	Success Rate (%) ReChisel	AutoChip
Pass@1	GPT-4 Turbo	73.24	**79.81**
	GPT-4o	77.46	**78.40**
	Claude 3.5 Sonnet	84.98	**91.08**
Pass@5	GPT-4 Turbo	83.10	**87.79**
	GPT-4o	**85.45**	84.51
	Claude 3.5 Sonnet	92.49	**96.71**
Pass@10	GPT-4 Turbo	85.92	**89.20**
	GPT-4o	**88.73**	87.79
	Claude 3.5 Sonnet	93.43	**97.65**

C. Performance Comparison of ReChisel and AutoChip

To further investigate the questions raised in Section III, we compare ReChisel with AutoChip [26]. AutoChip is an LLM-based agentic system designed for automatic *direct* Verilog code generation. Both systems are evaluated using the same benchmarks and parameters given in Section V-A. We select OpenAI's GPT-4o, GPT-4 Turbo, and Claude 3.5's Sonnet models for comparison. For both systems, the maximum number of iterations is set to 10. The results presented in Table III demonstrate that ReChisel enhances the LLM's Chisel code generation performance to a level almost comparable with Verilog generation. Specifically, for the GPT-4o model, ReChisel outperforms AutoChip in both Pass@5 and Pass@10 metrics. These results demonstrate the great potential of using Chisel for hardware code generation.

D. Case Study

Fig. 8 illustrates the ReChisel's reflection workflow through a case study. This case corresponds to the `Vector5` test case from AutoChip's HDLBits benchmark, utilizing the GPT-4o model. The figure

[2]Chisel employs parameterized module generators as a replacement for the `#(parameter ...)` mechanism in Verilog.

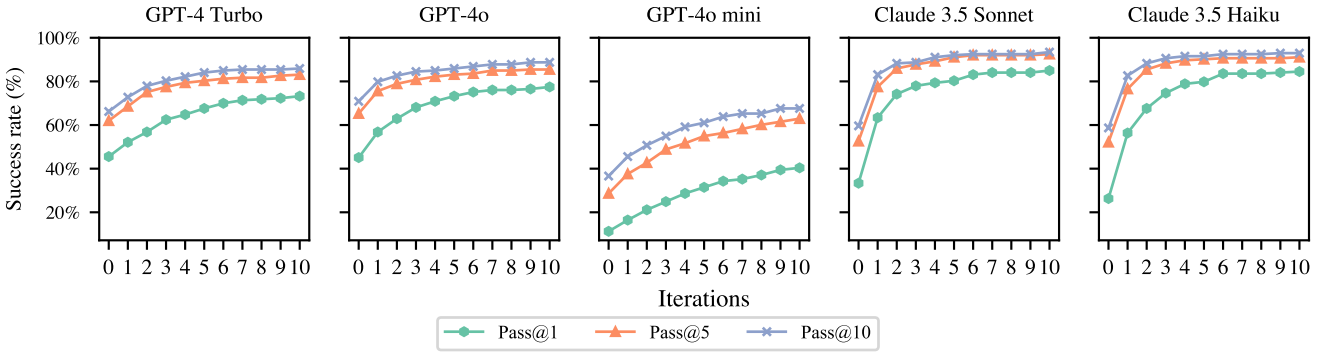


Fig. 6: Relationship between success rate and number of iterations for different LLMs.

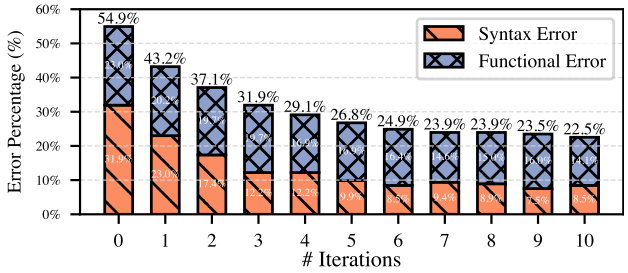


Fig. 7: Proportion of syntax and functional errors across iterations. Example of GPT-4o under Pass@1 metric.

illustrates the specifications and demonstrates how an LLM iteratively refines its output to progressively generate syntactically correct and functionally accurate Chisel code through reflection. The process was completed in three iterations. During the first two iterations, the code encountered syntax errors. In the first iteration, the LLM introduced intermediate variables to address multiple assignment issues, but an incorrect data type choice caused another error. In the second iteration, the LLM used boolean vectors that support bitwise operations, successfully generating syntactically correct code. In the third iteration, the syntactically correct code contained a functional error. The LLM then corrected the loop logic, ultimately producing a functionally correct implementation that passed all tests.

VI. Conclusion

In this work, we introduce ReChisel, an LLM-based agentic system designed to enhance the effectiveness of Chisel code generation. ReChisel incorporates a reflection mechanism to iteratively refine the quality of generated code using feedback from compilation and simulation processes, and introduces an escape mechanism to break free from non-progress loops. Experiments across three benchmarks and five mainstream LLMs demonstrate that ReChisel significantly improves the success rate of Chisel code generation, achieving performance comparable to state-of-the-art LLM-based agentic systems for Verilog code generation. This work highlights the potential of Chisel in LLM-assisted hardware design.

VII. Acknowledgement

This work is supported by the National Key Research and Development Program (Grant No. 2024YFB4405600), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grants No. BK20243042, BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005).

```
Ports: - input a, b, c, d, e,
       - output [24:0] out
Given five 1-bit signals (a, b, c, d, and e), compute all 25 pairwise one-bit comparisons in the 25-bit output vector. The output should be 1 if the two bits being compared are equal.
```

```
1| val out = IO(Output(UInt(25.W)))

3| val inputs = VecInit(a, b, c, d, e)
4| var idx = 0
5| for(i <- 0 until 5) {
6|   for(j <- i until 5) {
7|       when(inputs(i) === inputs(j)) { out(24.U - idx.U) := 1.U }
8|       .otherwise { out(24.U - idx.U) := 0.U }
9|       idx += 1
10| }}
```

Iter 0

Syntax Error

```
Multiple conflicting assignments to the same bits of the output vector out within a loop violate Chisel's enforcement of single static assignments to IO ports for ensuring hardware determinism.
```

```
1| val out = IO(Output(UInt(25.W)))
2|
3| val tempOut = Wire(Uint(25.W))
4| val inputs = VecInit(a, b, c, d, e)
5| var idx = 0
6| for(i <- 0 until 5) {
7|   for(j <- i until 5) {
8|       when(inputs(i) === inputs(j)) {tempOut(24.U - idx.U) := 1.U }
9|       .otherwise {tempOut(24.U - idx.U) := 0.U }
10|      idx += 1
11| }}
12| out := tempOut
```

Iter 1

Syntax Error

```
Individual bits of a UInt are read-only in Chisel; use a Vec of Bool for bit-level manipulation and convert it to UInt with asUInt after assignments.
```

```
1| val out = IO(Output(UInt(25.W)))
2|
3| val tempOut = Wire(Vec(25, Bool()))
4| val inputs = VecInit(a, b, c, d, e)
5| for (bit <- tempOut) { bit := false.B }
6| var idx = 0
7| for(i <- 0 until 5) {
8|   for(j <- i until 5) {
9|       tempOut(24 - idx) := inputs(i) === inputs(j)
10|      idx += 1
11| }}
12| out := tempOut.asUInt
```

Iter 2

Functional Error

```
Incorrect indexing and pairing logic in the nested loops lead to mismatched comparisons and improper mapping in the output vector.
```

```
   // ......
7| for(i <- 0 until 5) {
8|   for(j <- 0 until 5) {
   // ......
```

Iter 3, Success

Fig. 8: Case study on reflection. Green underlined codes represent modifications based on feedback from the previous iteration, while red codes indicate syntax or functional errors in the current iteration. The content within the error boxes summarizes the corresponding reviewer plans.

References

[1] K. Asanovic, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, vol. 4, pp. 6–2, 2016.

[2] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, "Chisel: constructing hardware in a scala embedded language," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 1216–1225.

[3] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*. IEEE, 2023, pp. 1–6.

[4] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[5] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[6] M.-S. Contributors, "Magma-si: A matrix accelerator generator," https://github.com/merledu/magma-si, 2024, accessed: 2024-11-15.

[7] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu *et al.*, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[8] H. Genc, A. Haj-Ali, V. Iyer, A. Amid, H. Mao, J. Wright, C. Schmidt, J. Zhao, A. Ou, M. Banister *et al.*, "Gemmini: An agile systolic array generator enabling systematic evaluations of deep-learning architectures," *arXiv preprint arXiv:1911.09925*, vol. 3, no. 25, pp. 15–17, 2019.

[9] HDLBits, "Problem sets - hdlbits," https://hdlbits.01xz.net/wiki/Problem_sets, accessed: 2024-11-11.

[10] F. Heilmann, C. Brugger, and N. Wehn, "White paper-investigate the high-level hdl chisel," *Kaiserslautern, Germany*, 2013.

[11] C.-T. Ho, H. Ren, and B. Khailany, "Verilogcoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (ast)-based waveform tracing tool," *arXiv preprint arXiv:2408.08927*, 2024.

[12] A. Izraelevitz, J. Koenig, P. Li, R. Lin, A. Wang, A. Magyar, D. Kim, C. Schmidt, C. Markley, J. Lawson, and J. Bachrach, "Reusability is firrtl ground: Hardware construction languages, compiler frameworks, and transformations," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2017, pp. 209–216.

[13] P. Lennon and R. Gahan, "A comparative study of chisel for fpga design," in *2018 29th Irish Signals and Systems Conference (ISSC)*. IEEE, 2018, pp. 1–6.

[14] P. S. Li, A. M. Izraelevitz, and J. Bachrach, "Specification for the firrtl language," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-9, Feb 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html

[15] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[16] S. Liu, W. Fang, Y. Lu, J. Wang, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[17] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "Openllm-rtl: Open dataset and benchmark for llm-aided design rtl generation(invited)," in *Proceedings of 2024 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. ACM, 2024.

[18] T. Liu, Q. Tian, J. Ye, L. Fu, S. Su, J. Li, G.-W. Wan, L. Zhang, S.-Z. Wong, X. Wang *et al.*, "Chatchisel: Enabling agile hardware design with large language models," in *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2024, pp. 710–716.

[19] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[20] H. Pearce, B. Tan, and R. Karri, "Dave: Deriving automatically verilog from english," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 2020, pp. 27–32.

[21] N. Pinckney, C. Batten, M. Liu, H. Ren, and B. Khailany, "Revisiting verilogeval: Newer llms, in-context learning, and specification-to-rtl tasks," *arXiv preprint arXiv:2408.11053*, 2024.

[22] M. Schoeberl, *Digital design with chisel*. Kindle Direct Publishing, 2019.

[23] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language agents with verbal reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[24] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[25] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[26] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," in *Proceedings of the 61st Design Automation Conference (DAC)*, San Francisco, CA, 2024.

[27] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language model," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[28] X. Wang, G.-W. Wan, S.-Z. Wong, L. Zhang, T. Liu, Q. Tian, and J. Ye, "Chatcpu: An agile cpu design & verification platform with llm," in *61st ACM/IEEE Design Automation Conference (DAC'24)*, 2024, p. 6.

[29] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824–24 837, 2022.

[30] S.-Z. Wong, G.-W. Wan, D. Liu, and X. Wang, "Vgv: Verilog generation using visual capabilities of multi-modal large language models," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[31] Y. Xu, Z. Yu, D. Tang, G. Chen, L. Chen, L. Gou, Y. Jin, Q. Li, X. Li, Z. Li *et al.*, "Towards developing high performance risc-v processors using agile methodology," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1178–1199.

[32] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao, "React: Synergizing reasoning and acting in language models," *arXiv preprint arXiv:2210.03629*, 2022.

[33] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," *arXiv preprint arXiv:2403.11671*, 2024.

[34] X. Yao, Y. Wang, X. Li, Y. Lian, R. Chen, L. Chen, M. Yuan, H. Xu, and B. Yu, "Rtlrewriter: Methodologies for large models aided rtl code optimization," *arXiv preprint arXiv:2409.11414*, 2024.

[35] Y. Zhang, Z. Yu, Y. Fu, C. Wan, and Y. C. Lin, "Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[36] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," May 2020.

LMM-IR: Large-Scale Netlist-Aware Multimodal Framework for Static IR-Drop Prediction

Kai Ma[1], Zhen Wang[1], Hongquan He[1], Qi Xu[2], Tinghuan Chen[3], Hao Geng[1†]
[1]ShanghaiTech University [2]University of Science and Technology of China
[3]The Chinese University of Hong Kong, Shenzhen

***Abstract*—Static IR drop analysis is a fundamental and critical task in the field of chip design. Nevertheless, this process can be quite time-consuming, potentially requiring several hours. Moreover, addressing IR drop violations frequently demands iterative analysis, thereby causing the computational burden. Therefore, fast and accurate IR drop prediction is vital for reducing the overall time invested in chip design. In this paper, we firstly propose a novel multimodal approach that efficiently processes SPICE files through large-scale netlist transformer (LNT). Our key innovation is representing and processing netlist topology as 3D point cloud representations, enabling efficient handling of netlist with up to hundreds of thousands to millions nodes. All types of data, including netlist files and image data, are encoded into latent space as features and fed into the model for static voltage drop prediction. This enables the integration of data from multiple modalities for complementary predictions. Experimental results demonstrate that our proposed algorithm can achieve the best F1 score and the lowest MAE among the winning teams of the ICCAD 2023 contest and the state-of-the-art algorithms.**

I. Introduction

As AI and high-performance computing (HPC) advance in tandem, new pathways like chiplet [1] bring design challenges and significant power consumption concerns. The analysis and minimization of integrity losses within signal and power distribution networks (PDNs) have become increasingly crucial. Each transistor, whether nMOS or pMOS, requires power, which is distributed across the microprocessor by metal layers [2]. While chip dimensions have remained relatively constant, advancements in fabrication technology have led to thinner, though not shorter, wires. This trend has resulted in a significant increase in resistance per unit length. For instance, between 28 nm and 7 nm nodes, there is nearly a tenfold increase in resistance, with exponential growth expected for smaller geometries. Voltage drops when current passes through a resistor. A decline in voltage can result in a reduction in the speed of a transistor, which may potentially lead to a functional failure if it occurs on a critical path within a design [3]. The potential for performance degradation and functional failures requires a comprehensive approach to IR drop analysis throughout the design cycle. IR drop stands as a pivotal focus for numerous EDA vendors.

To address these challenges, design engineers must repeatedly assess and address IR drop issues across the various design phases, extending from the initial placement to the final sign-off. The workflow of the commercial tool is shown in Fig. 1. However, the use of simulation-based methods to acquire precise IR drop estimates is an extremely time-consuming process [4]. Consequently, IR drop mitigation strategies that rely on frequent simulations usually require a large amount of computational resources and time, especially when performing static voltage drop simulations on a chip-wide scale. To enhance efficiency in this process, there is a crucial demand for an IR drop estimation method that is both fast and accurate. With the rapid advancement of artificial intelligence (AI), an increasing number of machine learning (ML)-based methods have been proposed for predicting static IR drop. PowerNet [5] introduced a cell-by-cell prediction methodology utilizing convolutional neural networks (CNNs). However, it's challenging to find the right cell size, and processing each cell individually is time-consuming. Besides, IREDGe [6] employed the U-Net [7] architecture to analyze

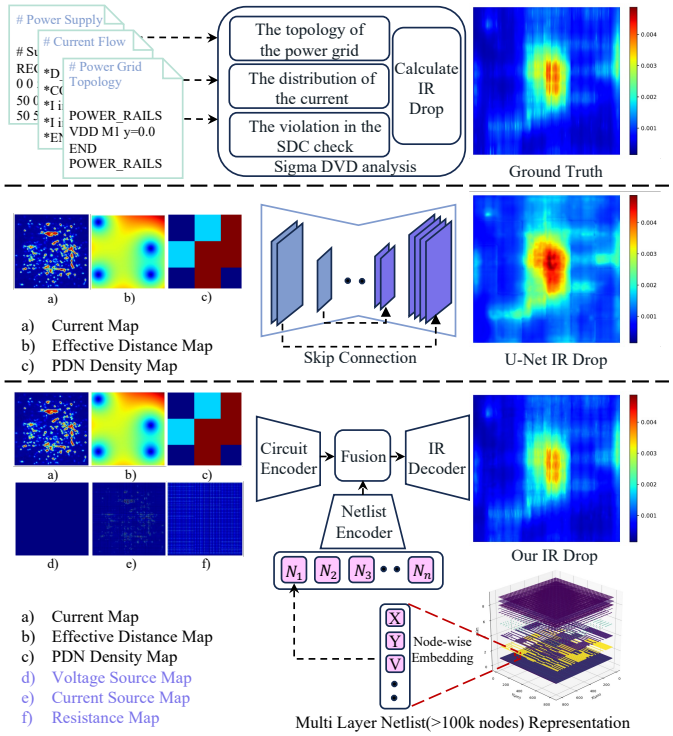


Fig. 1 The workflow of three models for IR drop prediction. From top to bottom: commercial tool, traditional U-Net, and the proposed LMM-IR model.

†Corresponding author.

the IR-drop while the conventional CNNs often fall short in capturing comprehensive global circuit context, leading to potential inaccuracies. Additionally, recent advancements such as IRPnet [8] proposed a shape-adaptive conventional kernel to capture more information from the physical structure. Nevertheless, the methodology employed by IRPnet is constrained by its dependence on single-modality features, which may be insufficient for capturing the intricate characteristics inherent in circuits. To address these fundamental challenges, we propose LMM-IR, a novel multi-modal fusion architecture. Our framework introduces an end-to-end paradigm with GPU acceleration that autonomously optimizes IR drop predictions across the entire chip. The LMM-IR architecture innovatively integrates a large-scale netlist transformer (LNT) to comprehensively represent netlist features, enabling superior capture of three-dimensional spatial relationships among metal layers. By aligning and fusing netlist data with circuit-level information, our approach facilitates mutual enhancement of diverse modalities, providing a more robust and accurate representation for IR-drop prediction. The main differences between our model, called LMM-IR, and previous machine learning-based works are summarized in TABLE I.

Departing from conventional simulation-based techniques and single modality model, we propose a novel large-scale netlist-aware multimodal framework that fundamentally transforms static voltage drop estimation, delivering superior accuracy and computational efficiency. An illustration of our multimodal framework is shown in Fig. 1.

Our contributions are summarized as follows:

- To the best of our knowledge, the first end-to-end framework is proposed which can directly process large-scale netlist data (more than 100K nodes) alongside circuit data, achieving state-of-the-art performance while maintaining competitive inference time. This breakthrough bridges the gap between netlist-level and circuit-level analysis in IR drop prediction.
- A novel large-scale netlist transformer (LNT) is designed for netlist feature extraction. This innovation enables comprehensive modeling of inter-layer connections and significantly enhances the model's ability to understand chip topology.
- For the first time, we propose a novel multi-modal fusion architecture that effectively aligns and integrates netlist and circuit data, facilitating mutual enhancement of different modalities.
- Extensive experiments demonstrate our approach's performance on large and complex circuits, overcoming the limitations of traditional image-based methods. The multimodal approach yields superior results.

TABLE I Comparison among different IR drop models.

Methods	Fully handle Netlist	Multimodal Fusion	Extra Features	Global attention mechanism
1st Place+[9]	✗	✗	✓	✓
2nd Place+[9]	✗	✗	✓	✓
IREDGe [10]	✗	✗	✗	✗
IRPnet [8]	✗	✗	✗	✗
Ours	✓	✓	✓	✓

+ "1st and 2nd Place" refer to the top two winners of 2023 ICCAD contest.

II. Preliminaries

A. Circuits Representation

The on-chip PDN structure [11] can be modeled as a network consisting of *voltage sources*, *current sources*, and *resistors*, where the wires form the resistive network, the power source is a voltage source connected to the wires of the PDN and the current source represents the cell/instance consuming the current. The goal of the static voltage drop simulation is to determine the voltage at the nodes in the PDN to which the instances (current sources) are connected.

The input data is provided in 2 formats:

1). *Circuit-based data:* There are 3 inputs determining the static IR drop, including current map, effective distance to the voltage source, and PDN density map. The circuit-based data is provided as matrices in comma-separated values (CSV) files, with each value in the file representing the current, effective distance, or PDN density in a $1\mu m \times 1\mu m$ region of the chip (noting that the spacing of the PDN nodes at the lowest level will be larger).

2). *SPICE-based data:* For each data point, the PDN model is provided as a SPICE netlist. The SPICE netlist describes the node positions, resistance values between nodes, current source locations and their values, as well as voltage source location and their values.

B. Multimodal Representation Learning

The field of multimodal representation learning [12] has attracted significant interest due to its capacity to integrate diverse data sources and enhance predictive performance in complex systems. This approach is particularly relevant in the field of electronic design automation (EDA), where integrating heterogeneous data types, including circuit-level characteristics and netlist descriptions, is essential for improving the accuracy of tasks such as IR drop prediction. Alternative methodologies, such as CLIP [13] and SLIP [14], generate distinct features for images and text independently, followed by an alignment phase. This efficient process enables large-scale pre-training while remaining robust in the presence of noisy or incomplete data. In electronic design automation (EDA), multimodal learning facilitates the combination of graphical circuit representations with structured netlist data, providing richer feature sets that enhance modeling precision and predictive capability. By integrating data from different modalities within a unified framework, this approach strengthens system analysis, particularly for critical applications such as power integrity and signal integrity verification.

C. Attention Module in IR Drop Prediction

Special attention must be paid to the prediction of hot areas, as their scope of hot areas is too small and their proportion in the whole network is not large. We employ the attention module from the Transformer [15] and modify it to incorporate cross-attention for the multimodality fusion

and self-attention for feature extraction within the netlist embedding. Let Q, K, and V represent the $query$, key, and $value$, respectively. Given an input $X \in \mathbb{R}^{n\times c}$ where n is the count of embedding entities and c is the number of channels per entity. Consider three weights $W_Q, W_K \in \mathbb{R}^{c\times d_\alpha}$ and $W_V \in \mathbb{R}^{c\times d_\beta}$ which are shared and learnable during training. Then, the input X interact with the weights:

$$(Q, K, V) = X \cdot (W_Q, W_K, W_V). \tag{1}$$

Finally, the attention features Y can be obtained by

$$Y = softmax(\frac{Q \cdot K^T}{\sqrt{d_\alpha}})V. \tag{2}$$

In IR drop, the wider range of information around can also have a significant effect. Thus, it is necessary to construct a broader view within the attention mechanism. Feature models operating at the level of coarse-grained spatial grids identify and describe locations and relationships among global-scale structures. However, it is still difficult to reduce false positive (FP) predictions for small objects that show large shape variations. In this paper, we demonstrate that higher accuracy can be achieved by integrating attention gates (AGs) [16] into standard CNN models and utilizing self-attention to capture netlist embedding features. This approach does not require training multiple models or introducing a large number of additional model parameters. Additionally, cross-attention is employed to better fuse different modalities. Compared to multi-stage CNN models, attention modules gradually suppress the feature responses in irrelevant IR region, focusing instead on extracting the pressure drop information.

D. Problem Formulation

Definition 1 (F1)**.** *To evaluate IR drop prediction accuracy, we employ a binary classification approach. Nodes with IR drop values exceeding 90% of the maximum true value are classified as positive samples, while others are negative. Predictions are divided into four groups: True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). The F1 score is calculated as follows:*

$$\mathrm{F1} = \frac{2 \times \mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}. \tag{3}$$

$$\mathrm{Precision} = \frac{TP}{TP + FP}, \quad \mathrm{Recall} = \frac{TP}{TP + FN}. \tag{4}$$

F1 is used to assess the model's performance, particularly in identifying significant IR drop nodes. A higher F1 score indicates better accuracy in predicting critical IR drop.

Definition 2 (Mean absolute error (MAE))**.** *In IR drop analysis, the accuracy of predicted voltage drops compared to actual measurements is often evaluated using the MAE. The metric is also used in the ICCAD 2023 contest.*

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |V_{pred,i} - V_{obs,i}|\,. \tag{5}$$

Lower MAE values indicate better predictive accuracy, as they reflect smaller deviations between predicted and observed voltages. IR-Drop predictor aiming to get lower MAE.

Definition 3 (Turn around time (TAT))**.** *Inference time of the ML model. The goal is to have fast inference.*

Problem 1. *Given netlist files and 3 circuit-based features: current map, PDN density map, and effective distance map, which are detailed in Section II-A, the objective is to train a multimodal-based framework that predicts the static IR drop within an acceptable inference time while achieving the highest F1 score, minimal mean absolute error (MAE).*

III. The Proposed IR Drop Prediction Flow

As illustrated in Fig. 2, we outline the complete architecture of our framework. We first introduce the circuit encoder. Subsequently, we present our large-scale netlist transformer (LNT) along with the direct netlist preprocessing approach. Then, our netlist encoding methodology is presented and offering insights in contrast to traditional netlist representation techniques. Lastly, we describe the fusion of the two modalities, followed by the multimodal decoder.

A. Embedding PDN with Circuit Encoder

For circuit-based thermal data processing, we leverage downsampling encoder structure to effectively extract hierarchical spatial features. In order to capture a receptive field large enough to capture semantic context information, the feature map grid in the standard CNN structure is gradually downsampled. With regard to the PDN structure-level features, this paper incorporates current, effective distance and PDN density map [9]. The effective distance is a metric that evaluates the distance from each location to all voltage sources. It is calculated as the reciprocal of the sum of the inverse Euclidean distances. Furthermore, the PDN density map is generated by extracting the mean PDN spacing within each grid [10]. This paper also introduces voltage and current source plots and resistance maps to provide additional insight into the PDN structure. The voltage and current source plot are derived based on the values at their respective source positions, while the resistance map is generated by distributing the resistance of each resistor across the corresponding overlapping grid cells.

What's more, the samples in the dataset differed significantly in spatial dimension (width × height), with edge lengths ranging from 204 to 930. However, it is necessary to ensure that each batch has a consistent spatial dimension during model training. Therefore, it is crucial to employ spatial adjustment techniques. Considering the structure of the entire network, cropping the data may lead to information loss, which may affect the prediction. Eventually, in this paper, scaling and padding techniques are used to process the data in order to maximize the retention of overall information.

During the model training phase, each batch undergoes a scaling or padding operation. Padding is applied only when the edge length is less than 512 which allows lossless encoding; conversely, scaling is performed if the edge length exceeds 512. These spatial adjustment techniques ensure that the overall information is preserved and the stability of the

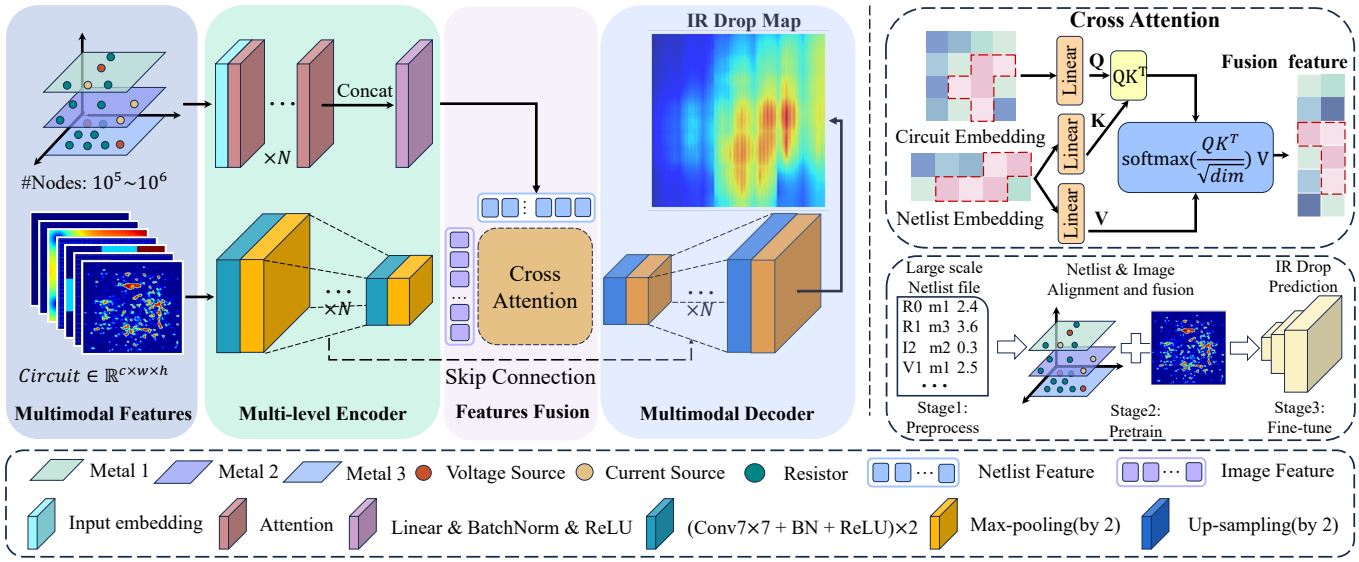


Fig. 2 Our LMM-Net framework.

data is maintained, thus improving the overall performance and robustness of the model. In addition, the corresponding normalization is performed for each channel. During model training, each batch of data is scaled or padded while the corresponding normalization process is performed for each channel. This normalization process helps to remove the inter-channel bias in the data, which in turn enhances the stability and generalization of the model. By mapping the range of values for each channel to similar intervals, normalization avoids the dominant influence of certain channels on model training, thus better preserving the statistical properties of the data. This further improves the model's ability to adapt to the input data and facilitates the model's generalization performance to unseen data.

B. Embedding the netlist into point cloud

Since convolutional neural networks (CNNs) are limited to capturing global information and pixel-based circuit representations often lack precision. To address this, we propose a novel large-scale netlist encoding approach that provides comprehensive global insights. As illustrated in Fig. 3, our approach can encode netlists with over 100k nodes without any information loss. In power delivery networks (PDNs), power delivery extends beyond a single layer, creating complex interactions that impact overall performance. Vias, which serve as vertical interconnects between different layers, play a critical role in enabling current flow across layers. However, this inter-layer connection introduces unique challenges, such as increased IR drops at via positions, which can degrade performance and signal integrity. Given their importance, accurately modeling and capturing the behavior of vias is essential for effective circuit analysis and optimization.

Traditional netlist representations often convert netlists into 2D maps. These methods obtain coordinate pairs (x_1, y_1) and (x_2, y_2) and place an average value between the start and end nodes. For nets spanning different layers, values are typically combined pixel-by-pixel. This approach ignores spatial connections. Unlike prior work, our approach encodes netlists using detailed attributes. These include coordinates (x_1, y_1), (x_2, y_2), values, element types (R, I, V), and key structural details such as originating and destination layers. If the layer1 is same as layer2, the component is a via. This rich and precise encoding captures node-wise connections and provides the model with a robust set of global features.

C. Encoding netlist point cloud with LNT

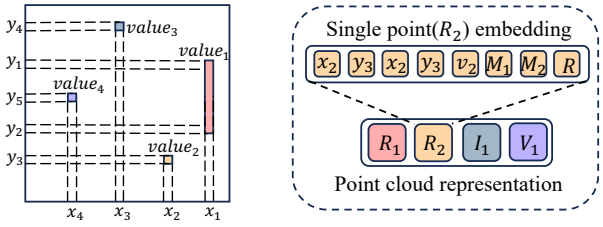
For SPICE netlist processing, we first construct a 3D point cloud representation [17] to capture the topological structure of the SoC. Each point in the cloud encodes critical physical and electrical characteristics including spatial coordinates (x_1, y_1, x_2, y_2) representing the node's position in the multi-layer structure, electrical parameters (node values), node type information, and layer identification. To effectively encode SPICE-based features into a shared feature space with circuit, we leverage the Large-scale Netlist Transformer (LNT) architecture that is inspired by [15]. This approach significantly benefits from transformer-based architectural principles while extending them to 3D points data processing.

Unlike grid-based or image-based representations, point clouds naturally handle the irregular structure of circuit netlists, where components and connections do not follow a regular grid pattern. Point clouds efficiently represent large-scale netlists with millions of nodes without the memory overhead of dense matrix representations, making them particularly suitable for modern complex SoC designs [18]. Point cloud representation enables seamless integration of both spatial information (3D coordinates) and electrical properties (node values, types) into a unified data structure, facilitating more effective feature learning.

The initial processing stage, illustrated in Fig. 3, transforms the circuit netlist into a point cloud representation. Each point in this representation encapsulates three key attributes: spatial

The format of the given netlist

#Type	#Node1	#Node2	#Value
R_1	(M_1, x_1, y_1)	(M_1, x_1, y_2)	$value_1$
R_2	(M_1, x_2, y_3)	(M_2, x_2, y_3)	$value_2$
I_1	(M_1, x_3, y_4)	0	$value_3$
V_1	(M_2, x_4, y_5)	0	$value_4$



Other works netlist representation Our novel netlist embedding

Fig. 3 Illustration of ordinary netlist representation vs. our proposed netlist embedding.

coordinates, electrical values, and component classifications (including resistors, current sources, and voltage sources).

Subsequently, each netlist undergoes transformation into a high-dimensional vector space through a trainable embedding layer. These embeddings are optimized during the training phase to capture essential circuit characteristics and topological relationships inherent in the netlist structure.

D. Multimodality Fusion Model for Static IR Drop

In the encoding phase, the circuit and netlist are encoded into features, respectively. These multi-dimensional features are projected into a higher-dimensional embedding space through a learned transformation. The embedded features are then processed by an attention mechanism that learns to focus on relevant node interactions while suppressing less important ones. Finally, a fully-connected layer transforms the attention-weighted features into the desired output representation.

On the contrary, in the decoding stage, 4 upsampling operations are performed, each with a coefficient of 2, and each time deconvolution is used to recover the spatial dimension. According to the observation, hotspots of static pressure drop are obviously not only affected by local information. Finally, the prediction result of the static pressure drop graph is obtained by 1×1 convolution operation.

Our framework adopts a dual-stream architecture that consists of four key components: circuit encoder, large-scale netlist transformer (LNT), multimodal fusion module, and decoder. The circuit encoder extracts spatial features from circuit layouts to capture physical power distribution patterns. In parallel, LNT processes netlist information through our direct netlist preprocessing approach, which encodes structural and connectivity features essential for IR drop analysis. The fusion module then combines spatial information from circuit maps and electrical characteristics from netlists, enabling comprehensive context awareness. Finally, the multimodal decoder leverages the fused representations to generate accurate IR drop. This architecture is trained end-to-end with MSE loss

TABLE II Statistics of the testcases.

Testcase	7	8	9	10	13
Nodes	85,591	83,030	166,734	159,940	15,768
Shape	601 × 601	601 × 601	835 × 835	835 × 835	257 × 257

Testcase	14	15	16	19	20
Nodes	15,436	57,508	55,197	181,206	174,304
Shape	257 × 257	489 × 489	489 × 489	870 × 870	870 × 870

Shape is measured in pixels.

that guides the model to learn critical IR drop while minimizing prediction errors. Our framework introduces an innovative two-stage training strategy. In the first phase, we employ a reconstruction task to train the network, enhancing the model's joint representation of circuit and netlist. Subsequently, the pre-trained model undergoes fine-tuning specifically for IR drop prediction. The trained model demonstrates strong generalization capabilities across different designs, offering fast and robust IR drop predictions by effectively capturing both spatial and electrical dependencies.

IV. Experiments

A. Experiments Settings and Benchmarks

We test our Predictor on a platform with NVIDIA GeForce RTX H100 80GB GPU. The programming environment is Python 3.9.18, and the PyTorch version is 2.1.1 with the CUDA version 12.3.

We select 512×512 as the input side length for each batch and use `Adam` as the optimizer, with a batch size of 16. The initial learning rate is set as 0.001 with a total epoch of 200. The ICCAD2023 contest [9] provided 100 fake cases and 10 real cases for training, plus 10 real cases for testing. Due to limited training data, 2000 cases provided by BeGAN [10] are used to expand the dataset in this paper. In addition, since the data provided by the ICCAD2023 competition is very similar to the distribution of real cases during the test phase, this paper over-samples each fake case 10 times and each real case 20 times. Therefore, the training set contains a total of 3310 cases. Finally, we test our model on the 10 hidden case shown in TABLE II.

B. Comparisons with SOTA works

In TABLE III, we compare our proposed algorithm against the winning team of the ICCAD2023 competition, as well as the IRPnet [8] and IREDGe [19] methods, which are re-implemented in PyTorch for this paper. It is worth noting that the third-party code we tested experiences runtime issues and crashes in the last two cases. Despite this, the experimental results show that our algorithm's F1 score improves by 20% compared to the first-place team. This is noteworthy, as the second-place team enhanced their training data by generating additional cases with parameter modifications (around 5,400 cases in total), giving their model a significant competitive advantage. Our algorithm achieves the best F1 score in 6 out of the tested cases, as well as the overall best average F1 score. In contrast, IREDGe [19] performs poorly due to its

TABLE III Comparison with state of the arts.

Circuits	1st Place [9]			2nd Place [9]			IREDGe [10]			IRPnet [8]			Ours		
	F1	MAE	TAT	F1	MAE	TAT	F1	MAE	TAT	F1	MAE	TAT	F1	MAE	TAT
testcase7	**0.78**	0.66	14.61	0.56	0.78	3.22	0.16	5.77	**1.53**	0.17	2.39	2.87	0.72	**0.63**	2.82
testcase8	0.82	**0.82**	12.64	0.8	1.13	2.70	0.20	4.20	**1.27**	0.10	2.30	2.43	**0.84**	0.84	2.57
testcase9	**0.59**	**0.41**	18.84	0.55	0.73	4.25	0.04	4.71	**2.42**	0.00	5.05	3.46	0.47	0.42	4.63
testcase10	0.53	**0.66**	19.05	0.15	1.14	**4.13**	0.01	4.76	2.67	0.00	2.02	2.89	**0.60**	0.71	4.43
testcase13	0.00	2.07	9.60	**0.67**	**1.25**	1.25	0.38	8.42	1.64	0.01	5.78	1.22	0.52	1.52	**1.15**
testcase14	0.00	4.22	10.07	0.10	**2.32**	1.40	0.05	7.43	1.99	0.00	2.33	1.13	**0.44**	3.24	**1.11**
testcase15	0.09	**0.97**	12.99	0.00	1.92	2.15	0.1	5.48	**1.77**	0.00	5.51	2.88	**0.54**	1.49	2.20
testcase16	0.53	**1.60**	12.12	0.48	3.44	2.19	0.31	10.21	**0.97**	0.01	5.78	2.21	**0.55**	3.33	2.43
testcase19	0.50	0.91	19.05	0.49	1.20	4.55	0.05	4.62	**2.52**	0.01	2.71	3.14	**0.61**	**0.74**	4.60
testcase20	**0.71**	1.18	18.75	0.74	1.07	4.58	0.02	7.24	3.39	0.00	5.91	**3.12**	0.54	**0.64**	4.61
Avg	0.46	1.35	14.77	0.45	1.50	3.04	0.13	6.28	**2.02**	0.03	3.98	2.54	**0.58**	**1.35**	3.05
Ratio	0.80	1.00	4.84	0.77	1.11	0.99	0.22	4.65	**0.66**	0.05	2.95	0.83	**1.00**	**1.00**	1.00

MAE in 10^{-4}, TAT in *second*.

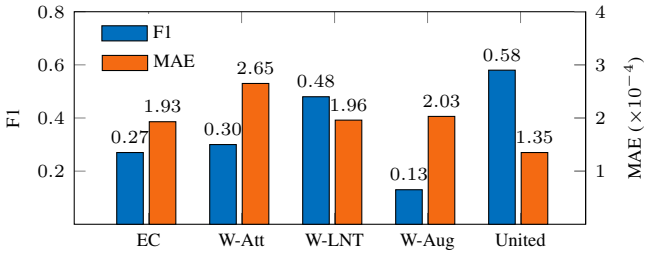


Fig. 4 Ablation studies on ICCAD-2023 contest dataset for different techniques' application.

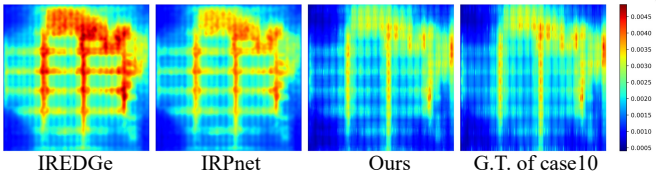


Fig. 5 The IR drop prediction visualizations of SOTA works and ours.

limited feature set, which includes only the current map, PDN density map, and valid distance map, as well as its simpler models. IRPnet [8] was only tested on ten real-circuit data samples, so we reconstructed it to test on the hidden-circuit data. In the hidden case, there are several cases that differs from the training set. The results show that IRPnet does not demonstrate sufficient generalization to the hidden cases.

In conclusion, our proposed algorithm achieves excellent and robust performance without incurring a significant time overhead. Fig. 5 provides a visual comparison of the circuit IR Drop map generated by IREDGe, IRPnet, our method, and ground truth on the testcase10.

C. Ablation studies

As shown in Section III-C, we use a netlist transformer to process large-scale netlists. The ordinary encoder-decoder architecture often struggles to capture the global information that can represent the intricate circuit connections. To address this challenge, we leverage the self-attention block to focus on the high IR-Drop areas, which are a significant factor in the prediction. Additionally, due to the limited data available and the inherent circuit connection relationships within the data, simple cropping and flipping could potentially disrupt the circuit characteristics. Therefore, we employ Gaussian noise augmentation with a standard deviation range of $(0, 1e^{-3})$ to enrich our training set.

To evaluate the effectiveness of the techniques mentioned above, we conduct experiments on the ICCAD2023 benchmark. The results are shown in Fig. 4. The different configurations tested are: EC (only encoder-decoder flow), W-Att (without attention block), W-Aug (without augmentation), W-LNT (without large netlist transformer), and United (using all techniques together). Based on the results shown in Fig. 4, we can evaluate the impact of each technique. For F1 scores, the techniques are ranked in terms of impact from highest to lowest as follows: LNT, attention mechanism, and data augmentation. In terms of MAE, the impact order is LNT, data augmentation, and attention mechanism. The large-scale netlist transformer (LNT) technique has the most significant impact, representing a key contribution to our approach for hierarchical netlist encoding. These results highlight the importance of hierarchical netlist encoding and the effectiveness of the proposed techniques in improving the model's performance on the ICCAD2023 contest benchmark.

V. Discuss and Conclusion

In this paper, a Multimodal Fusion Model based on LNT embedding large-scale netlist is firstly proposed for the task of static voltage drop prediction. Meanwhile, additional feature maps are introduced to enhance the model's learning ability. Experimental results show that the algorithm proposed in this paper achieves the best F1 score and minimun mae compared with the winning team of ICCAD 2023 competition and the most advanced method currently. Future research will further explore the application of specific large language model (LLM) to address large text information in PDN structures to better optimize the model.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (NSFC) under grant No. 92473113.

REFERENCES

[1] T. Li, J. Hou, J. Yan, R. Liu, H. Yang, and Z. Sun, "Chiplet heterogeneous integration technology—status and challenges," *Electronics*, vol. 9, no. 4, p. 670, 2020.

[2] I. P. Vaisband, R. Jakushokas, M. Popovich, A. V. Mezhiba, S. Köse, E. G. Friedman, I. P.-Vaisband, R. Jakushokas, M. Popovich, A. V. Mezhiba *et al.*, "Simultaneous co-design of distributed on-chip power supplies and decoupling capacitors," *On-Chip Power Delivery and Management*, pp. 245–253, 2016.

[3] W.-H. Chang, L.-Y. Lin, Y.-G. Chen, and M. C.-T. Chao, "Power distribution network generation for optimizing ir-drop aware timing," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[4] Y. Zhong and M. Wong, "Fast algorithms for ir drop analysis in large power grid," *International Conference on Computer Aided Design,International Conference on Computer Aided Design*, May 2005.

[5] Z. Xie, H. Ren, B. Khailany, Y. Sheng, S. Santosh, J. Hu, and Y. Chen, "Powernet: Transferable dynamic ir drop estimation via maximum convolutional neural network," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2020.

[6] V. A. Chhabria, V. Ahuja, A. Prabhu, N. Patil, P. Jain, and S. S. Sapatnekar, "Thermal and ir drop analysis using convolutional encoder-decoder networks," in *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, 2021, pp. 690–696.

[7] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[8] Y. Meng, R. Lyu, Z. Bi, C. Yan, F. Yang, W. Hu, D. Zhou, and X. Zeng, "Circuits physics constrained predictor of static ir drop with limited data," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024, pp. 1–2.

[9] T. Sato, C.-Y. Wang, Y.-G. Chen, and T.-W. Huang, "Invited paper: Overview of 2023 cad contest at iccad," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2023.

[10] V. A. Chhabria, V. Ahuja, A. Prabhu, N. Patil, P. Jain, and S. S. Sapatnekar, "Thermal and ir drop analysis using convolutional encoder-decoder networks," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, 2021.

[11] S. K. Samal, K. Samadi, P. Kamal, Y. Du, and S. K. Lim, "Full chip impact study of power delivery network designs in gate-level monolithic 3-d ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 6, pp. 992–1003, 2016.

[12] P. Xu, X. Zhu, and D. A. Clifton, "Multimodal learning with transformers: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 10, pp. 12 113–12 132, 2023.

[13] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.

[14] N. Mu, A. Kirillov, D. Wagner, and S. Xie, "Slip: Self-supervision meets language-image pre-training," in *European conference on computer vision*. Springer, 2022, pp. 529–544.

[15] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[16] O. Oktay, J. Schlemper, L. L. Folgoc, M. Lee, M. Heinrich, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz *et al.*, "Attention u-net: Learning where to look for the pancreas," *arXiv preprint arXiv:1804.03999*, 2018.

[17] H. Woo, E. Kang, S. Wang, and K. H. Lee, "A new segmentation method for point cloud data," *International Journal of Machine Tools and Manufacture*, vol. 42, no. 2, pp. 167–178, 2002.

[18] P. Kumar Datla Jagannadha, M. Yilmaz, M. Sonawane, S. Chadalavada, S. Sarangi, B. Bhaskaran, and A. Abdollahian, "Advanced test methodology for complex socs," in *2016 IEEE International Test Conference (ITC)*, 2016, pp. 1–10.

[19] V. A. Chhabria, K. Kunal, M. Zabihi, and S. S. Sapatnekar, "Began: Power grid benchmark generation using a process-portable gan-based methodology," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021.

LEMOE: LLM-Enhanced Multi-Objective Bayesian Optimization for Microarchitecture Exploration

Jingyuan Li[1], Jianrong Zhang[1], Ye Li[2], Wenbo Yin[1], Lingli Wang[1,*]
[1] State Key Laboratory of Integrated Chips and Systems, Fudan University, Shanghai, China
[2] School of Computer Science and Engineering, University of New South Wales, New South Wales, Australia
[*]Email: jingyuanli22@m.fudan.edu.cn, llwang@fudan.edu.cn

***Abstract*—Designing processor microarchitectures is increasingly challenging due to a vast design space and the need to balance multiple metrics. Traditional algorithm-driven design space exploration (DSE) approaches often struggle to incorporate the extensive domain knowledge of expert architects. To address this, we introduce LEMOE, a multi-objective microarchitecture optimization framework that leverages large language model (LLM) to enhance an implicit Bayesian model. LEMOE features a program-aware warm-up phase utilizing LLM and LLVM to produce an initial design set with rich prior knowledge. By harnessing LLM's contextual learning, our approach improves surrogate modeling and sampling under sparse data conditions. Experiment results show that LEMOE achieves a 22.8% improvement in energy efficiency with the same number of iterations and a 2.9× runtime speedup for the same target compared to prior works.**

***Index Terms*—In-context learning, Bayesian Optimization, Large Language Model, Microarchitecture design**

I. Introduction

RISC-V, an open-source instruction set architecture (ISA), has become a prominent choice in processor design due to its modular and flexible structure. This adaptability enables extensive customization, making RISC-V popular across diverse applications, from embedded systems [1] [2] and high-performance computing [3] [4] to deep neural networks [5] [6].

Designing a RISC-V microarchitecture is a complex task. The numerous architecture parameters—such as fetch width, branch predictor size, and load/store queue entries—create an expansive combinatorial space. Navigating this complexity requires either experienced designers or efficient design space exploration (DSE) methods to identify configurations that balance multiple objective trade-offs. Microarchitecture evaluation typically relies on commercial or open-source simulators [7], [8], which are both time-consuming and labor-intensive, often taking hours or even days per design evaluation [8]. Therefore, reducing the number of iterations needed to find optimal designs is essential to improve exploration efficiency.

Many methodologies have succeeded in RISC-V DSE in recent years. In [9], bai *et.al* employed a microarchitecture-aware active learning algorithm to generate an initial design set and utilized the Gaussian process with deep kernel learning built to characterize the design space. [10] integrated microarchitecture scaling graphs in reinforcement learning and utilized an agent model to mimic human exploration preference. In [11] and [12], the authors employed graph representation learning and multi-objective Bayesian optimization with an ensemble model to explore microarchitecture within a compact embedding space. In [13], a co-training k-nearest neighbors model was used to fit microarchitecture features and assist in searching for a Pareto frontier through parallel emulation.

Existing optimization methods can be broadly categorized into black-box and white-box approaches. Black-box optimization [9] [11] [13] [14] [15] is used in scenarios where the internal structure of the system is completely unknown. This approach relies on observing inputs and outputs, as the objective function's internal mechanisms are inaccessible. While it is more adaptable to different architectures, it often involves blindly testing inputs to identify optimal solutions. Additionally, it also struggles to incorporate designers' prior knowledge, potentially overlooking key design features that architects are already familiar with. White-box optimization [10] [16] [17], on the other hand, leverages knowledge of the system's internal structure. Although white-box approaches can be highly efficient, developing such frameworks requires experienced designers and often lacks portability when the target architecture changes.

Large language models (LLMs) offer a promising bridge between black-box and white-box optimization approaches. Models like GPT-4 [18] and BERT [19] have rapidly evolved into transformative technologies in artificial intelligence. Trained on vast amounts of textual data, these models learn to understand and generate text by identifying complex linguistic patterns and contextual relationships. This capability enables LLMs to produce coherent responses through in-context learning [20]. Recent studies have demonstrated the potential of LLMs in optimization tasks. For instance, [21] and [22] explored the use of LLMs to implement genetic search operators in evolutionary algorithms, while [23] validated the feasibility of using LLMs as surrogate models in Bayesian optimization. The extensive RISC-V domain knowledge embedded in LLMs' training corpora inspires us to integrate the human expert prior knowledge into traditional black-box optimization process.

Specifically, this article presents LEMOE, a novel multi-objective framework for RISC-V microarchitecture DSE. LEMOE introduces a program-aware initialization approach leveraging LLVM. By analyzing input programs at a fine-grained level using LLVM, the framework utilizes prior knowl-

[*]Corresponding author.

edge of RISC-V designs and program features for a warm-start. In the Bayesian optimization, LEMOE replaces the non-informative Gaussian process, integrating an LLM as a surrogate model. This approach combines the strengths of black-box and white-box optimization, utilizing the LLM's rich prior knowledge and designing an insightful sampler to enhance exploration efficiency. Our contributions are summarized as follows:

- We introduce a program-aware initialization method for warm-starting the exploration process. By leveraging the prior knowledge of LLM in RISC-V and LLVM feature analysis, this approach effectively guides iterative exploration.
- We propose a novel multi-objective optimization algorithm that leverages an LLM as a surrogate model and incorporates an insightful sampler for implicit Bayesian inference. This method facilitates efficient trade-offs between performance and power while mitigating the inefficiencies of black-box optimization.
- We perform an ablative study on the various modules and parameters of LEMOE. Comparative analyses with previous work highlight the advantages of LEMOE in enhancing multi-objective exploration efficiency. To the best of our knowledge, this is the first study to utilize an LLM to improve multi-objective Bayesian optimization for RISC-V microarchitecture exploration.

Paper Organization: Section II provides an overview of the background and essential preliminaries. Section III presents our microarchitecture exploration framework. Section IV compares with previous works and performs ablation experiments. Section V comprises the conclusion.

II. Background & Preliminaries

A. Microarchitecture modeling

We utilize Chipyard [24] to model our RISC-V microarchitecture. Chipyard is an open-source framework for designing, simulating, and deploying Chisel-based hardware systems. The Berkeley Out-of-Order Machine (BOOM) is a high-performance RISC-V core that implements out-of-order execution. BOOM is designed to be modular and configurable, offering a wide array of tunable parameters. Key architectural features such as issue width, pipeline depth, cache hierarchy, and branch prediction mechanisms can all be adjusted to meet specific design goals. However, this flexibility comes with the challenge of managing a complex design space, as the interactions between these parameters are often non-linear and difficult to predict. This makes it important to explore the design space to find configurations that balance multiple objective trade-offs. The design space of the BOOM we target is shown in TABLE I. The definition of these parameters can refer to [25].

TABLE I: Design Space Overview

Module	Parameter	Value
I-cache	numSets	$1, 4, 8$
	numWays	$32, 64$
D-cache	numSets	$64, 128$
	numWays	$2, 4, 8$
	numMSHR	$2, 4, 8$
IDU	decodeWidth	$1, 2, 3, 4, 5$
	numRobEntries	$30, 36, 60, 72, 90, 108, 136$
	IntPhyRegister	$42, 62, 70, 80, 90, 100, 110, 128, 146$
	FpPhyRegister	$38, 48, 54, 64, 74, 86, 96, 118, 128$
EU	memIssueWidth	$1, 2$
	intIssueWidth	$1, 2, 3, 4, 5$
	fpIssueWidth	$1, 2, 3, 4, 5$
LSU	numMemEntries	$6, 10, 14, 20, 28$
	numLdqEntries	$6, 8, 12, 16, 20, 24, 28, 32$
IFU	fetchWidth	$4, 8$
	FetchBufferEntries	$6, 12, 16, 24, 30, 36, 45$
Branch	maxBrCount	$6, 8, 12, 16, 20, 24$
	FetchTargetQueue	$14, 20, 30, 36, 50$

B. Definition & Target

Pareto frontier. Pareto frontier is a set of optimal solutions in multi-objective optimization where improving one objective means sacrificing another. For a M-dimension minimization problem, for two solutions $s_i = (f_1^i, ..., f_M^i)$ and $s_j = (f_1^j, ..., f_M^j)$, s_i dominates s_j ($s_i \succeq s_j$) if and only if $f_1^i \leq f_1^j, f_2^i \leq f_2^j, ..., f_M^i \leq f_M^j$. s_i is *Pareto efficient* if no other solutions dominate s_i. All such Pareto efficient solutions are defined as the *Pareto frontier*.

Hypervolume. Assume a set of sampling points after evaluation $\mathcal{S} = \{\boldsymbol{s_1, ..., s_n}\}$, given its Pareto frontier, $\mathcal{P}(\mathcal{S})$. Choose a reference point $s_{ref} \in \mathbb{R}^L$, which is dominated by each point in $\mathcal{P}(\mathcal{S})$. The hypervolume is calculated as:

$$\mathrm{HV}(\mathcal{P}(\mathcal{S})) = \int_{\mathbb{R}^L} \mathbb{I}[\boldsymbol{s} \succeq \boldsymbol{s_{ref}}][1 - \prod_{s^* \in \mathcal{P}(\mathcal{S})} \mathbb{I}[\boldsymbol{s}^* \not\succeq \boldsymbol{s}]]d\boldsymbol{s} \quad (1)$$

where $\mathbb{I}(.)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise. A higher hypervolume indicates a better spread and coverage of optimal solutions, showing a balanced trade-off across the objectives.

Target. The BOOM microarchitecture is evaluated based on two key metrics: performance, measured as *instructions per cycle (IPC)*, and *power*. High-performance designs often lead to increased power consumption, our goal is to identify Pareto-efficient microarchitecture designs that effectively balance these metrics within a limited number of iterations.

III. LEMOE Framework

A. Overview

The LEMOE framework is outlined in Fig. 1 and comprises five steps. **Program-Aware Initialization**: LLVM analyzes the program set, generating features that are transformed into an LLM prompt to initialize Bayesian optimization. **Sampling**: Both existing data and the LLM's prior knowledge of the BOOM microarchitecture, obtained through in-context learning, are used for sampling. **Surrogate Modeling**: The LLM serves as a surrogate model to estimate the sampled points. **Acquisition Function**: A multi-objective acquisition function is introduced to identify optimal sampling points. **RISC-V Evaluation**: The selected sampling points undergo evaluation

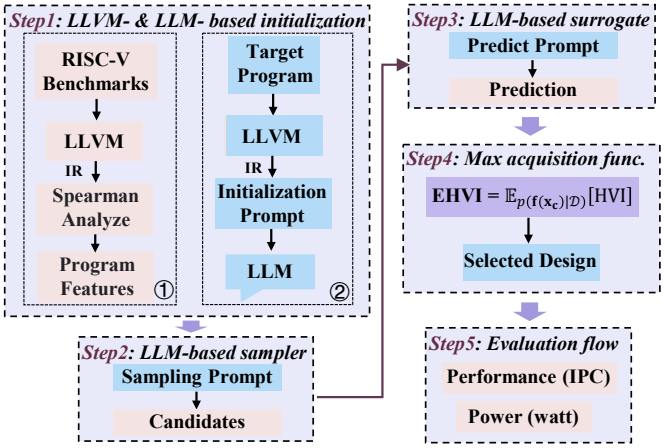


Fig. 1: LEMOE flowchart overview.

TABLE II: PROGRAM FRATURES

Type	Feature	Description
Structure	numBB	# of basic blocks
	avgPred	Average predecessors of a BB
	avgSucc	Average successors of a BB
	numBranch	# of branch instructions
	numInst	# of instructions (of all types)
	numCfgEdges	# of edges in CFG[1]
	numPhi	# of *Phi* nodes
	numRet	# of return instructions
	numCall	# of call instructions
Memory	numLoad	# of load instructions
	numStore	# of store instructions
	numAlloc	# of alloca instructions
Computation	numUnaryInst	# of unary instructions
	numBinaInst	# of binary instructions
	numComp	# of computation instructions
	numFloatComp	# of float instructions

[1] : CFG represents Control Flow Graph.

within the VLSI flow. Steps 2 to 5 are iteratively repeated to identify the final Pareto frontier.

B. Program-aware Initialization

Compared to random initialization [26] or initialization based on architecture features [9], program-aware initialization methods aim to capture the structure of the target program efficiently. Input program features directly influence design decisions related to performance optimization, energy efficiency, and resource allocation. For instance, if a program exhibits many memory-bound operations, optimizing the cache hierarchy—such as adjusting cache associativity or block size—can significantly enhance performance. Similarly, frequent floating-point operations in a program may warrant optimized issue widths for floating-point instructions. To achieve this, input programs written in C/C++ are processed through an LLVM front-end, which translates the source code into LLVM Intermediate Representation (IR). The LLVM IR retains detailed information about the program's control flow, data types, and instructions. A summary of the program's parsed characteristics is provided in TABLE II.

```
You are assisting me with automated RISC-V BOOM core
parameters setting, the following are paremeters and the
range of their values: {design space}.
The average characteristics of the input program are:
{program features}. Please suggest {num} diverse yet
effective configurations based on the input program
characteristics to initiate a Bayesian Optimization
process. You mustn't include "None" in the
configurations. Your response should include only a list
of dictionaries, where each dictionary describes one
recommended configuration. Do not enumerate the
dictionaries.
```

Fig. 2: Program-aware initialization prompt. Blue text indicates variables.

TABLE III: FEATURE SELECTION

Key Feature	Highly correlated	Correlation
numBB	numBranch	0.99
	numCfgEdges	0.99
	numPhi	0.95
	numCall	0.96
numLoad	numStore	0.99
numInst	numUnaryInst	0.91
	numBinaInst	0.93
	numComp	0.98
avgPred	avgSucc	0.98

However, the features in TABLE II are not entirely independent. For instance, the number of basic blocks (*numBB*) is closely related to the number of branch instructions (*numBranch*), as the structure and quantity of basic blocks significantly influence the number of branch instructions. To streamline program feature information, we extract features from the target benchmark set [27] and analyze the Spearman correlation coefficient [28] between them. This coefficient ranges from -1 to 1, with values closer to -1 or 1 indicating a stronger correlation between the two features. Finally, we remove the features with the correlation absolute value greater than 0.9, as shown in TABLE III, and select the following 6 features: ***numBB***, ***numLoad***, ***numInst***, ***numRet***, ***avgPred***, ***numFloatComp***. Then we initialize the design set using the prompt as shown in Fig. 2.

Program-aware initialization leverages prior knowledge of specific workloads to guide the selection of initial design parameters. Aligning initialization with program characteristics ensures that exploration begins with optimized and relevant configurations, avoiding the selection of highly implausible initial points. This methodology makes microarchitecture exploration application-specific. The evaluation of this approach is detailed in Section IV.

C. LLM-enhanced Bayesian optimization

After the initialization phase, we proceed by iteratively selecting the most promising points for exploration. Consider the sampled dataset $\mathcal{D}_n = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}_{i=1}^n$ comprising n sets of microarchitecture design $\mathcal{X} = \{\boldsymbol{x_i}\}_{i=1}^n$ and m corresponding evaluation metrics $\mathcal{Y} = \{\boldsymbol{y_i}\}_{i=1}^n = \{y_1^1, ..., y_n^1; y_1^2, ..., y_n^2; ...; y_1^m, ..., y_n^m\}^\top$, the relationship be-

```
The following outlines the parameter configurations for
a RISC-V BOOM core, along with the corresponding metrics
measured in [IPC, power]. The BOOM core consists of
{tot_feats} parameters, with allowable ranges defined as
{design space}. Recommend a configuration that achieves
the target power-performance hypervolume of {target}.
Avoid selecting values at the minimum or maximum of the
allowable range. Your response should only include the
predicted configuration in the format
## configuration ##.
Metric: {IPC1,power1}
Configuration: {configuration 1}
…
Metric: {IPCN,powerN}
Configuration: {configuration N}
Metric:{target}
Configuration:
```

Fig. 3: Samping prompt.

```
The following outlines the parameter configurations for
a RISC-V BOOM core, along with the corresponding metric
measured in [IPC, power]. The BOOM core consists of
{tot_feats} parameters. Your response should only
include the predicted [IPC, power] in the format
## [IPC, power] ##.
Configuration: {configuration 1}
Metric: {IPC1,power1}
…
Configuration: {configuration N}
Metric: {IPCN,powerN}
Configuration: {configuration to predict}
Metric:
```

Fig. 4: Surrogate model prompt.

tween the samples is independently and identically distributed. When considering a new sampling point x^*, the predictive distribution of the d-th objective can be given by:

$$p(y^d|x^*;\mathcal{D}_n) = \int_\Theta p(y^d|x^*;\mathcal{D}_n,\theta)p(\theta|x^*;\mathcal{D}_n)d\theta \quad (2)$$

where θ is a latent variable and $p(\theta|x^*;\mathcal{D}_n)$ denotes the posterior distribution. In Bayesian optimization, a Gaussian process (GP) is often used as a surrogate model to capture the relationship between x^* and y^d, where θ represents the hyperparameters of the GP model. From the Bayes' theorem, we note that $p(\theta|x^*;\mathcal{D}_n) \propto p(\theta)p(\mathcal{D}_n|\theta,x^*)$, $p(\theta)$ serves an important prior role. However, traditional black-box optimization often relies on an uninformative prior, potentially overlooking valuable domain-specific knowledge that could enhance optimization efficiency. This limitation inspired us to leverage LLMs to enhance Bayesian optimization.

An effective surrogate model should accurately represent the central predicted values for different input points, along with the confidence or uncertainty in those predictions to balance between exploration and exploitation. For this purpose, we represent the sampled dataset $\mathcal{D}_n$ as a natural text $\mathcal{D}_n^{'}$, which is combined with a predefined prompt to generate mean and uncertainty of x^* for estimation, denoted as $(m^d, v^d) = LLM(\mathcal{D}_n^{'}, x^*)$. To increase robustness, we introduce a shuffling mechanism in $\mathcal{D}_n^{'}$, since LLM is sensitive to the order of examples in the context [29].

The acquisition function uses the predictions of the surrogate model to determine where to sample next. Common acquisition functions [30] are probability of improvement (PI), expected improvement (EI), and upper confidence bound (UCB). To better balance power and performance, we introduce expectation hypervolume improvement (EHVI) [31] to evaluate the final candidate solutions and it is defined as follows:

$$\text{EHVI}(\boldsymbol{x}_{n+1}|\mathcal{D}) = \mathbb{E}_{p(\boldsymbol{f}(\boldsymbol{x}_{n+1})|\mathcal{D})}[HV_{n+1} - HV_n] \quad (3)$$

Typically, HV_{n+1} is calculated using the posterior distribution of x_{n+1} generated by the surrogate model. We introduce the LLM-based sampling method to compute EHVI, which offers three main advantages. First, the EHVI function is notorious for its high computational complexity, and the LLM-based non-analytical method reduces the computational overhead. Second, explicit uncertainty estimation for the sampling points is unnecessary with the non-analytical method. The LLM only needs to estimate the mean values of the sampling points, denoted as $m^d = LLM(\mathcal{D}_n^{'}, x^*)$. This is a method similar to the popular SMAC [32] for obtaining probability estimates. Third, the LLM-based sampler incorporates prior knowledge of RISC-V microarchitecture, which enhances sampling efficiency. Comparative experiments will be presented in Section IV.

To sample design points that would improve exploration, we set target values for exploration. For multi-objective optimization, we use hypervolume as a composite metric due to its comprehensive response to solution quality. The following objective is used as sampling expectations:

$$hv^* = hv_{cur} + \alpha \times (hv_{cur} - hv_{init}) \quad (4)$$

where hv_{cur} and hv_{init} denote the currently realized hypervolume and the hypervolume of the initial sampling set. hv^* is defined relative to the best hypervolume value, with the difference proportional to the observed improvement in hypervolume. The hyperparameter $\alpha > 0$ regulates explorability, where larger values of α result in higher expectations. The determination of the parameters is studied by Section IV. We use the prompt shown in Fig.3 to sample N candidate points in parallel, denoted as $\mathcal{C} = \{x_n\}_{n=1}^N$. These points are then evaluated using the LLM to estimate performance and power, as illustrated in Fig.4, resulting in $\hat{\mathcal{M}} = \{y_n\}_{n=1}^N$. After collecting and estimating N points in parallel during each iteration, the next potential point x^* is sampled by maximizing the EHVI:

$$x^* = \arg\max_{x\in\mathcal{C}}[\text{EHVI}(\mathcal{D}_n, \mathcal{C}, \hat{\mathcal{M}})] \quad (5)$$

The design x^* is then fed into the automated RISC-V modeling and evaluation flow. The pseudocode of our framework is presented in Algorithm 1.

Algorithm 1 LEMOE pseudocode

Input:
$\mathcal{X}$ the design space of RISC-V microarchitecture;
$\mathcal{B}$ target program set;
$N_i, N_l, N \in \mathbb{N}$: numbers of initial samples & LLM samples per iteration & iterations;
Output:
Pareto-optimal design $\mathcal{P}$;

$\mathcal{I} \leftarrow ProgramAwareInit(N_i, \mathcal{X}, \mathcal{B})$;
$\mathcal{D} = Evaluate(\mathcal{I}) \cup \mathcal{I}$;
for $t \leftarrow 1, 2, ..., n$ **do**
  $\mathcal{C} \leftarrow LLMSampler(\mathcal{D}, N_l)$;
  $\{(i_l, p_l)\}_{l=1}^{N_l} \leftarrow LLMSurrogate(\mathcal{C})$;
  $x^* \leftarrow \arg\max_{x \in \mathcal{C}} [\text{EHVI}(\mathcal{D}, \mathcal{C}, \{(i_l, p_l)\}_{l=1}^{N_l})]$;
  $\mathcal{D} = Evaluate(x^*) \cup \mathcal{D}$;
end for
return $\mathcal{P} = Pareto(\mathcal{D})$

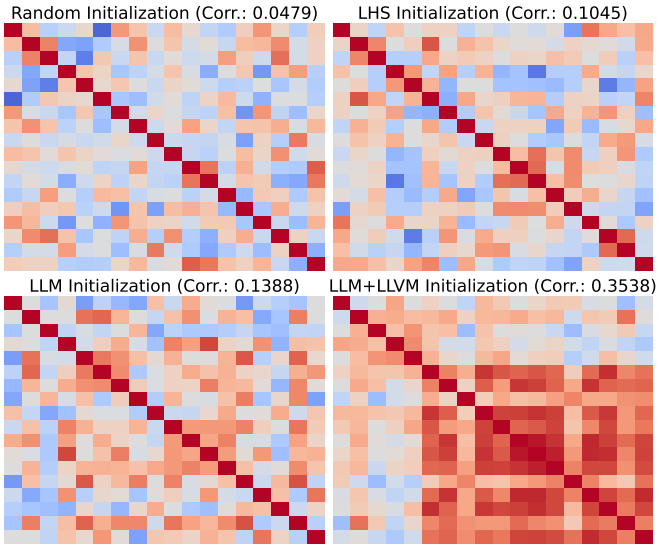


Fig. 5: Correlation heatmap of sampled points. The top shows the average correlation for each method.

IV. Evaluation

In this section, we systematically evaluate the performance of LEMOE compared to previous works, conduct ablative studies, and provide quantitative and qualitative analyses.

A. Experiment setup

We compare LEMOE with several prior works, including the Random Forest algorithm, the Tree-structured Parzen Estimator (TPE) algorithm [33] (NeurIPS'11), its multi-objective version [34] (JAIR'22), the ML-based architecture exploration platform [35] (ISCA'23), BOOM-Explorer [9] based on Bayesian optimization and deep kernel learning (ICCAD'21), and a single-objective optimization framework using LLM sampling [36] (ICLR'24).

In our framework, we use GPT-3.5 and GPT-4o as the LLMs. The total number of sampling points is set to 15, with an initial sample size of 5 points. Our method utilizes a program-aware initialization, while the ICCAD'21 method employs a microarchitecture-aware active learning algorithm, and the other methods use random initialization. The test benchmarks, sourced from [27], include *mm*, *median*, *vvadd*, and *qsort*. These benchmarks cover all types of instructions. Metrics are evaluated based on the average power and average IPC across all four benchmarks. Experiments are conducted on a workstation with 2 Intel Xeon 2.10GHz CPUs and 256 GB memory.

B. Initialization comparison

We begin with a systematic study of the individual components of LEMOE before performing an end-to-end comparison. Specifically, we compare different initialization methods: random initialization, Latin Hypercube Sampling (LHS), initialization using only the LLM, and the program-aware initialization based on LLM and LLVM. For each method, we calculate the Spearman correlation matrix and the average correlation for 50 initial sampling points. The correlation heatmap is shown in Fig. 5.

The top of the figure illustrates the average correlation for each method. The results show that the LLM-based initialization method achieves a higher correlation between sampling point parameters compared to both the random initialization and LHS methods. This indicates that LLM effectively captures parameter relationships. Moreover, the LLVM- and LLM-based program-aware initialization further enhances sampling correlations, offering greater insight into the input program compared to the LLM-based method.

C. Ablation study

We conduct an end-to-end comparison of LEMOE using three configurations: (L1) using random sampling without the LLM-based sampler, (L2) omitting LLVM IR for initialization, (L3) excluding RISC-V-specific content for contextual learning, and finally, the full LEMOE configuration (L4). We evaluate the hypervolume and IPC/power under GPT-3.5 and GPT-4o, as shown in TABLE IV. The results show that using random sampling instead of LLM reduces solution quality due to the absence of an insightful sampler. Similarly, omitting LLVM for feature enhancement results in a decline in quality. The most significant quality drop occurs when RISCV- specific information is excluded, highlighting the crucial role of prior knowledge in an efficient exploration framework.

In equation 4, α represents the intensity of our exploration, with larger values of α indicating more aggressive exploration. We evaluate different values of α under GPT-3.5 and GPT-4o, as shown in TABLE V. With GPT-3.5, an α of 0.05 yields the highest solution quality, while with GPT-4o, a more aggressive α of 0.1 performs best. Both larger (0.2) and smaller (0.01) α values tend to reduce solution quality.

TABLE IV: ABLATION STUDY

LLM	Method	IPC / Power[1]	Ratio	HV [2]	Ratio
GPT-3.5	L1	42.271	1.213×	0.754	85.6%
	L2	49.107	1.409×	0.824	93.5%
	L3	34.853	1	0.676	76.7%
	L4	51.369	1.474×	0.881	1
GPT-4o	L1	43.473	1.078×	0.801	89.6%
	L2	49.196	1.220×	0.854	95.4%
	L3	40.321	1	0.752	84.0%
	L4	52.090	1.292×	0.895	1

[1] : Power is measured in milliwatts (mW).
[2] : HV represents hypervolume.

TABLE V: IMPLICATIONS OF EXTRAPOLATION

LLM	Alpha	IPC / Power	Ratio	HV	Ratio
GPT-3.5	0.2	47.931	1.092×	0.823	93.4%
	0.1	46.241	1.054×	0.845	95.9%
	0.05	**51.369**	**1.170**×	**0.881**	**1**
	0.01	43.891	1	0.820	93.0%
GPT-4o	0.2	48.661	1.112×	0.847	94.6%
	0.1	**52.090**	**1.191**×	**0.895**	**1**
	0.05	48.197	1.102×	0.840	93.8%
	0.01	43.753	1	0.816	91.2%

D. End-to-end Comparsion

The results of end-to-end comparison with prior works are shown in TABLE VI. Among all methods, our framework utilizing GPT-4o achieves superior performance. Compared to Ours-4o, the results show a 45.3% improvement in the IPC/Power and a 22.1% increase in the hypervolume over the Random Forest. Relative to NeurIPS'11 and JAIR'22, IPC/power improves by 23.3% and 7.5%, while hypervolume improves by 24.5% and 16.0%, respectively. For ISCA'23, improvements of 11.6% and 12.9% are observed, while comparisons with ICLR'24 and ICCAD'21, our method shows respective improvements of 19.2% and 29.6% in IPC/power and 17.2% and 9.0% in hypervolume. Figure 6 illustrates the Pareto frontiers achieved by each method. For clearer visualization, we use the inverse of IPC, represented as cycles per instruction (CPI), ensuring that both metrics are minimized.

We also compare runtime, which consists of two parts: the surrogate modeling time, typically very short (ranging from seconds to a few minutes), and the EDA tool's runtime, which takes several hours to tens of hours per design evaluation. Consequently, the surrogate modeling time constitutes a minimal portion of the total runtime. The previous works perform modeling locally, while the LLM operates in the cloud. Although this setup makes the runtime comparison somewhat lenient, the impact of this difference is negligible. We evaluate the Runtime ratio for the same Hypervolume Target (RHT), which compares the total runtime of prior methods to the runtime required by our Ours-4o method to achieve the same hypervolume. Higher RHT values indicate faster runtime for reaching the same target. As shown in TABLE VI, our method achieves a speedup ranging from 1.956× to a maximum of 3.565×. LEMOE's superior performance arises from its domain-aware approach, which leverages RISC-V-specific knowledge to provide an informed starting point for optimization. By utilizing surrogate models and samplers that incorporate prior information, LEMOE effectively guides the exploration, concentrating on the space with high-potential solutions.

TABLE VI: COMPARSION WITH PRIOR WORKS

Method	IPC / Power	Ratio	HV	Ratio	RHT [1]
Random Forest	35.846	1	0.697	77.9%	3.509×
NeurIPS'11 [33]	42.263	1.179×	0.676	75.5%	3.565×
JAIR'22 [34]	48.460	1.351×	0.751	84.0%	2.925×
ISCA'23 [35]	46.676	1.302×	0.780	87.1%	2.167×
ICCAD'21 [9]	40.178	1.121×	0.814	91.0%	1.956×
ICLR'24 [36]	43.695	1.219×	0.741	82.8%	2.996×
Ours-3.5	51.369	1.433×	0.881	98.5%	-
Ours-4o	52.090	1.453×	0.895	1	-

[1] : RHT represents Runtime ratios for the same Hypervolume Target.

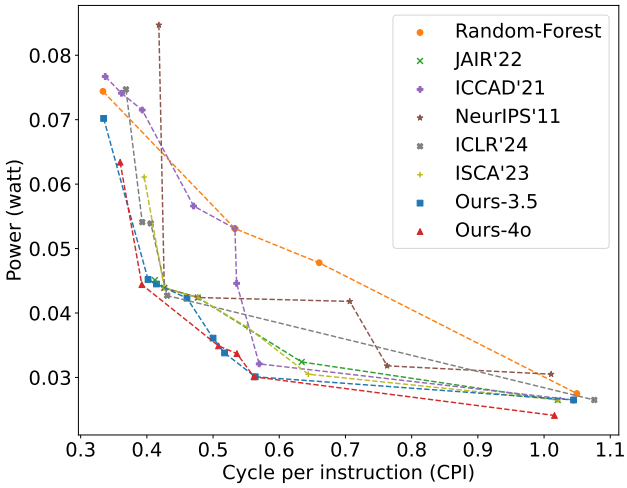


Fig. 6: Pareto frontiers of different methods, where a smaller indicator represents better quality. Note that CPI and IPC are inversely related.

V. Conclusion

This article presents LEMOE, a multi-objective optimization framework that leverages the LLM-enhanced Bayesian model to explore the design space of RISC-V microarchitecture. LEMOE incorporates domain-specific knowledge through LLM integration and employs a program-aware initialization algorithm, utilizing LLVM and LLM to warm up the optimization effectively. By replacing the traditional Gaussian process with LLM and incorporating a multi-objective acquisition function for sampling and prediction, LEMOE enhances iterative efficiency. Evaluation of LEMOE on the BOOM demonstrates its capability to optimize key metrics. Future work includes comparing additional LLMs and addressing potential hallucination phenomena that may arise with LLMs.

Acknowledgment

This work is supported by National Science and Technology Major Project (2021ZD0114701).

REFERENCES

[1] B. Sá, J. Martins, and S. Pinto, "A first look at risc-v virtualization from an embedded systems perspective," *IEEE Transactions on Computers*, vol. 71, no. 9, pp. 2177–2190, 2021.

[2] Y. Kra, Y. Shoshan, Y. Rudin, and A. Teman, "Hamsa-di: A low-power dual-issue risc-v core targeting energy-efficient embedded systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2023.

[3] F. Minervini, O. Palomar, O. Unsal, E. Reggiani, J. Quiroga, J. Marimon, C. Rojas, R. Figueras, A. Ruiz, A. Gonzalez, *et al.*, "Vitruvius+: an area-efficient risc-v decoupled vector coprocessor for high performance computing applications," *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 2, pp. 1–25, 2023.

[4] A. Bartolini, F. Ficarelli, E. Parisi, F. Beneventi, F. Barchi, D. Gregori, F. Magugliani, M. Cicala, C. Gianfreda, D. Cesarini, *et al.*, "Monte cimone: paving the road for the first generation of risc-v high-performance computers," in *2022 IEEE 35th International System-on-Chip Conference (SOCC)*, pp. 1–6, IEEE, 2022.

[5] S. Lee, J. Choi, S. Noh, J. Koo, and J. Kung, "Dbps: Dynamic block size and precision scaling for efficient dnn training supported by risc-v isa extensions," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[6] L. Huang, C. Fang, Q. Li, J. Lin, and Z. Wang, "A precision-scalable risc-v dnn processor with on-device learning capability at the extreme edge," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 927–932, IEEE, 2024.

[7] J. Zhai, C. Bai, B. Zhu, Y. Cai, Q. Zhou, and B. Yu, "Mcpat-calib: A risc-v boom microarchitecture power modeling framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 1, pp. 243–256, 2022.

[8] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[9] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2021.

[10] C. Bai, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Towards automated risc-v microarchitecture design with reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 12–20, 2024.

[11] X. Yi, J. Lu, X. Xiong, D. Xu, L. Shang, and F. Yang, "Graph representation learning for microarchitecture design space exploration," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2023.

[12] Z. Wu, J. Shen, X. Yi, L. Shang, F. Yang, and X. Zeng, "Prior-boosted grl: Microarchitecture design space exploration via graph representation learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[13] X. Zheng, M. Cheng, J. Chen, H. Gao, X. Xiong, and S. Cai, "Bsse: Design space exploration on the boom with semi-supervised learning," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2024.

[14] D. Luo, Q. Sun, X. Li, C. Bai, B. Yu, and H. Geng, "Knowing the spec to explore the design via transformed bayesian optimization," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

[15] J. Li, Y. Hu, Y. Dai, H. Kuang, and L. Wang, "Auger: A multi-objective design space exploration framework for cgras," in *2023 International Conference on Field Programmable Technology (ICFPT)*, pp. 88–95, IEEE, 2023.

[16] B. Sá, L. Valente, J. Martins, D. Rossi, L. Benini, and S. Pinto, "Cva6 risc-v virtualization: Architecture, microarchitecture, and design space exploration," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2023.

[17] N. A. Koca, C. H. Chang, A. T. Do, and V. P. Nambiar, "Exploring error correction circuits on risc-v based systems for space applications," in *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2024.

[18] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[19] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[20] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[21] A. Chen, D. Dohan, and D. So, "Evoprompting: language models for code-level neural architecture search," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[22] J. Lehman, J. Gordon, S. Jain, K. Ndousse, C. Yeh, and K. O. Stanley, "Evolution through large models," in *Handbook of Evolutionary Machine Learning*, pp. 331–366, Springer, 2023.

[23] M. C. Ramos, S. S. Michtavy, M. D. Porosoff, and A. D. White, "Bayesian optimization of catalysts with in-context learning," *arXiv preprint arXiv:2304.05341*, 2023.

[24] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton, *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, vol. 40, no. 4, pp. 10–21, 2020.

[25] J. Zhao, B. Korpan, A. Gonzalez, and K. Asanovic, "Sonicboom: The 3rd generation berkeley out-of-order machine," in *Fourth Workshop on Computer Architecture Research with RISC-V*, vol. 5, pp. 1–7, 2020.

[26] A. I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R. R. Griffiths, A. M. Maraval, H. Jianye, J. Wang, J. Peters, *et al.*, "Hebo: Pushing the limits of sample-efficient hyper-parameter optimisation," *Journal of Artificial Intelligence Research*, vol. 74, pp. 1269–1349, 2022.

[27] "Risc-v unit tests benchmark suites," GitHub, https://github.com/riscv-software-src/riscv-tests, 2023.

[28] C. Wissler, "The spearman correlation formula," *Science*, vol. 22, no. 558, pp. 309–311, 1905.

[29] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2022.

[30] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[31] S. Daulton, M. Balandat, and E. Bakshy, "Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9851–9864, 2020.

[32] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter, "Smac3: A versatile bayesian optimization package for hyperparameter optimization," *Journal of Machine Learning Research*, vol. 23, no. 54, pp. 1–9, 2022.

[33] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011.

[34] Y. Ozaki, Y. Tanigaki, S. Watanabe, M. Nomura, and M. Onishi, "Multiobjective tree-structured parzen estimator," *Journal of Artificial Intelligence Research*, vol. 73, pp. 1209–1250, 2022.

[35] S. Krishnan, A. Yazdanbakhsh, S. Prakash, J. Jabbour, I. Uchendu, S. Ghosh, B. Boroujerdian, D. Richins, D. Tripathy, A. Faust, *et al.*, "Archgym: An open-source gymnasium for machine learning assisted architecture design," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, pp. 1–16, 2023.

[36] T. Liu, N. Astorga, N. Seedat, and M. van der Schaar, "Large language models to enhance bayesian optimization," in *The Twelfth International Conference on Learning Representations (ICLR)*, 2024.

GoPTX: Fine-grained GPU Kernel Fusion by PTX-level Instruction Flow Weaving

Kan Wu, Zejia Lin, Mengyue Xi, Zhongchun Zheng, Wenxuan Pan, Xianwei Zhang[#], Yutong Lu[#]
Sun Yat-sen University, Guangzhou, China
{wukan3, linzj39, ximy, zhengzhch3, panwx5}@mail2.sysu.edu.cn, {zhangxw79, luyutong}@mail.sysu.edu.cn

***Abstract*—GPUs have been heavily utilized in diverse applications, and numerous approaches, including kernel fusion, have been proposed to boost GPU efficiency through concurrent kernel execution. However, these approaches generally overlook the opportunities to mitigate warp stalls and improve instruction level parallelism (ILP) in inter-kernel resource sharing. To address this issue, we introduce `GoPTX`, a novel design for kernel fusion that improves ILP through deliberate weaving instructions at the PTX level. `GoPTX` establishes a merged control flow graph (CFG) from original kernels, enabling to interleaving of instructions that were sequentially executed by default and minimizing pipeline stalls on data hazards. We further propose a latency-aware instruction weaving algorithm for more efficient instruction scheduling and an adaptive code slicing method to enlarge the scheduling space. Experimental evaluation demonstrates that `GoPTX` achieves an average speedup of 11.2% over the baseline concurrent execution, with a maximum improvement of 23%. The hardware resource utilization statistics show significant enhancements in eligible warps per cycle and resource use.**

***Index Terms*—GPU, Kernel Fusion, ILP, Warp Stall, Data Hazard**

I. Introduction

As GPUs incorporate an increasing amount of computing resources, it becomes difficult for a single GPU kernel to fully utilize the vast resources. One solution is to share resources through concurrent kernel execution, where GPU vendors provide hardware parallel task queues and management, such as cuStream, MPS [1], and MIG [2] to execute multiple tasks. Essentially, these solutions rely on the GPU's thread-block (TB) scheduler to allocate resources among the co-running kernels. Nonetheless, the leftover policy [3], [4] employed by the TB scheduler results in imbalanced resource allocation, greatly limiting the effectiveness of intra-SM resource sharing.

To overcome the hardware limitations, software-based approaches have been proposed for improving inter-kernel resourcing utilization. One such representative method is kernel fusion, which has been adopted by mainstream deep learning frameworks [5], [6], tools [7], [8] and systems [9], [10]. The fused kernel enforces the co-execution of instructions from two kernels onto the same SM which issues warps that utilize complementary resources, thereby improving the overall SM utilization. However, such warp-level concurrency incurs frequent thread context switching and stalls because of resource contention [3], which can negatively impact the instruction pipeline and further ILP. To mitigate such overhead, carefully instruction scheduling is needed and thus a fine-grained technique to weave instruction flows from concurrent kernels into a unified flow is urgently demanded. By weaving independent instructions, the data dependency length can be increased, allowing instructions from another kernel to proceed without stalling or warp switching, filling pipeline bubbles, hiding the execution latency, as shown in Figure 1.

However, several unique challenges are associated with instruction weaving. *First*, kernels may have complex control flow graphs (CFGs) that include synchronization barriers for groups of threads. Weaving the instructions of these kernels requires a careful operation to merge the diverging CFGs into a unified structure while ensuring correctness and avoiding potential deadlocks. *Second*, achieving perfect instruction reordering is an NP-hard problem [11], [12], especially for GPUs with thousands of parallel threads. Identifying a near-optimal weaving output within a limited time is crucial for a feasible solution. To the best of our knowledge, no existing software-based solutions automatically weave instructions from distinct kernels to share resources and optimize ILP. Previous attempts [13], [14] have been limited to fusion at the source code level, focusing primarily on exploiting TLP.

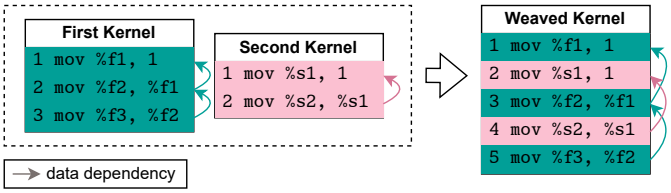


Fig. 1: Increase dependency length by instruction weaving.

In this paper, we present `GoPTX`, a novel design for kernel fusion that operates at the PTX level. `GoPTX` introduces a new CFG merging algorithm to unify CFGs from multiple kernels, which involves critical techniques to preserve the execution semantics and resolve deadlocks caused by synchronization. Moreover, we develop a latency model to guide the interleaving of independent instructions within the basic blocks of newly generated CFGs, which effectively helps fill pipeline bubbles to optimize execution performance. Additionally, the latency model is used to estimate the elapsed cycles of basic blocks (BBs), enabling the adaptive slicing of long codes into balanced segments before CFG merging, creating more opportunities for instruction weaving.

The contributions of this paper are summarized as follows:

- Our analysis reveals that GPU warp stalls primarily originate from the scoreboard, hinders the achievable ILP.
- We propose `GoPTX`, a design for kernel fusion that weaves instructions from distinct kernels and fills the pipeline bubbles. `GoPTX` is further augmented with code slicing to expose more ILP opportunities. `GoPTX` particular handles to avoid deadlocks.
- Evaluations show that `GoPTX` effectively improves execution performance with optimized hardware utilization and reduced stalls, outperforming the prior arts of kernel fusion.

II. Background and Motivation

A. Background

a) GPU Hardware Architecture: A GPU[1] consists of multiple streaming multiprocessors (SMs), each containing hundreds of processing cores and on-chip components like register files and

[#]Corresponding author.

[1]We use the terms from NVIDIA GPU design in this paper.

L1 caches. Logical threads execute on these cores in a single-instruction multiple-thread (SIMT) fashion. A group of 32 threads (a warp) operates in lock-step within the execution pipeline. If a warp stalls, the scheduler switches to another eligible warp, continuously issuing instructions (Figure 2). This thread-level parallelism (TLP) hides instruction pipeline latency, making the GPU a high-throughput computational device.

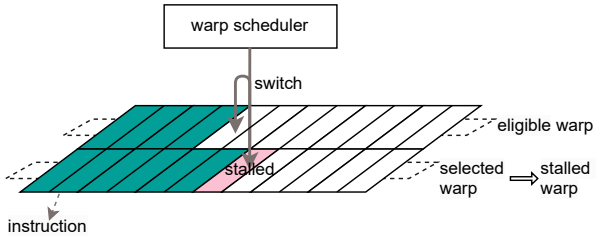


Fig. 2: The GPU warp scheduler frequently encounters stalls, causing a switch to another thread and resulting in "bubbles" (pink colored) on the execution path.

b) CUDA Compilation Workflow: Programmers define parallel portions of their applications as kernels within the vendor's dialect (e.g., CUDA C). As shown in Figure 3, the CUDA frontend (e.g., `nvcc` or `clang` [15]) parses the source code and generates an intermediate representation (IR) named PTX. Subsequently, the PTX code is compiled by the backend compiler `ptxas` into an executable binary file (*cubin*) that is tailored for the target GPU architecture. PTX is defined by NVIDIA and resembles LLVM IR [16]. A kernel in PTX consists of basic blocks (BBs) forming a control flow graph (CFG). Each BB contains a bunch of diverse instructions and ends with a terminator instruction (e.g., a branch or function return). All the instructions are in single-assignment form with limitless registers available and may include hardware-specific primitives like barriers and tensor core operations. PTX offers a direct abstraction of the underlying GPU hardware, enabling effective backend compiler optimizations.

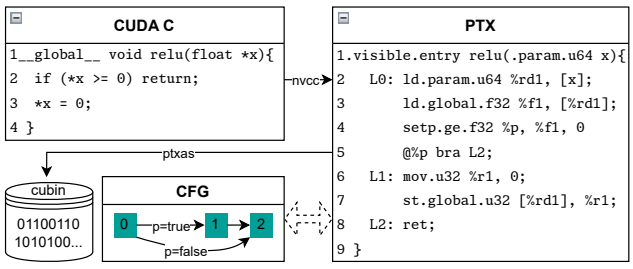


Fig. 3: CUDA compilation workflow.

c) Kernel Fusion: Targeting improved utilization and performance, kernel fusion combines two concurrent kernels at the software level to share on-chip resources. Unlike hardware methods, kernel fusion incurs no hardware overhead or limitations, making it flexible and efficient. Figure 4 shows two primary existing fusion techniques and our desire. Vertical fusion (`VFuse`) [13] sequentially combines the instructions from input kernels, horizontal fusion (`HFuse`) [14] distributes the instructions to different warps, while our desired fine-grained weaves two kernels together.

B. Opportunity to Mitigate Warp Stalls

We analyze utilization statistics from representative application kernels (Section IV-B) to explore the potential of ILP. Figure 5a compares the average number of eligible warps per cycle (EWPC) of the kernels. We observed that the EWPC of other kernels is significantly lower compared to the computationally intensive MICND kernel. Figure 5b shows the stall reasons. The most prevalent stall cause for kernels other than MICND is the scoreboard, which handles data dependencies and enables out-of-order instruction execution. This suggests that data hazards and resource contention cause pipeline stalls and limit the achievable EWPC and ILP.

To alleviate the scoreboard stalls , fine-grained instruction scheduling is required. However, current intra-kernel instruction reordering techniques reach a performance ceiling because of data dependents. Therefore, we turn to "weave" instructions between concurrent kernels for further scheduling space. Figure 1 shows an example, where each instruction depends on the previous one so all the data dependency length is 1, and can not be reordered or pipelined to improve ILP. After weaving, all the dependency length increases to 2, improving the pipeline's ability to process dependent instructions.

III. Design

In this section, we present `GoPTX`, a novel design for kernel fusion that improves ILP by weaving instructions at PTX level, enabling efficient resource sharing between kernels while mitigating warp stalls. Figure 6 depicts the overall workflow of `GoPTX`, generating a weaved PTX code from two input PTX codes. The workflow comprises three phases:

a) CFG Merging: A new control flow graph (CFG) is constructed, unifying the execution paths of both kernels. Branching conditions and dummy nodes are inserted to maintain equivalents and avoid deadlocks.

b) Instruction Weaving: A low-complexity weaving strategy constructs a locally optimal instruction sequence, minimizing pipeline stalls. Instructions are scheduled based on latencies measured via micro-benchmarking.

c) Code Slicing (Optional): Long BBs are split into multiple short segments before CFG merging, creating opportunities for cross-BB instruction weaving.

A. CFG Merging

Algorithm 1 describes CFG merging (CFM) as a traversal procedure working on BB-level. CFM merges the input kernels, denoted as F and S into a merged CFG M, ensuring that the generated CFG preserves the execution semantics of both original kernels. Initially, the output CFG M is set to empty. The traversal begins at the initial state $\langle 0, 0\rangle$ of M, corresponding to the starting BBs F_0 and S_0 of both CFGs (lines 3–6). For each traversed state $\langle f, s\rangle$ (line 8), the followings are performed. Firstly, CFM fuses BBs F_f and S_s into a single BB $M[\langle f, s\rangle]$, providing the foundation for *instruction weaving*. Then, the successors of the merged BB are retrieved using the function get_next_list (line 10). The traversal continues until all possible state $M[\langle f, s\rangle]$ are consumed (lines 11–15).

The function get_next_list is tailored to handle branching in CFG merging. Figure 7 illustrates the outputs of this function, depending on the characteristics of the origin basic blocks (BB) $F[f], S[s]$. For trivial BBs without branches (Figure 7a), the function returns the succeeding BB if $M[\langle f, s\rangle]$ is not an end node, otherwise an empty node. When the BB has conditional branches (Figure 7b), the function exhaustively combines each conditional branch from the original kernels. To handle the branching logic correctly, two dummy nodes are inserted, allowing the complete conditions to be determined and facilitating correct execution across the merged paths.

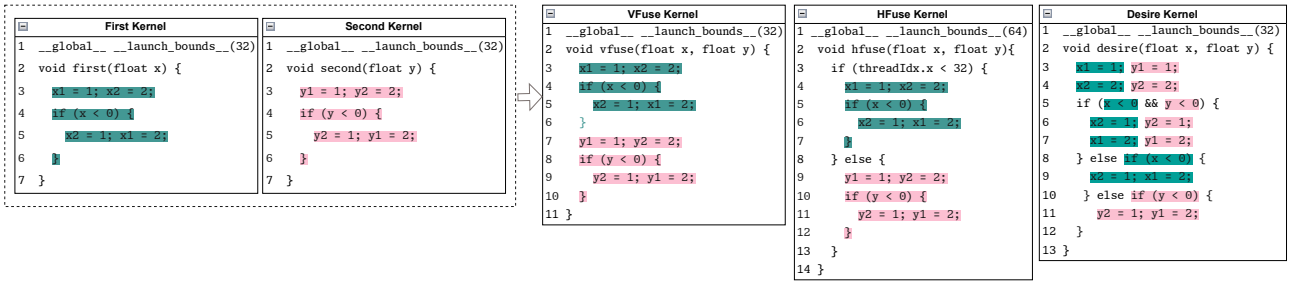
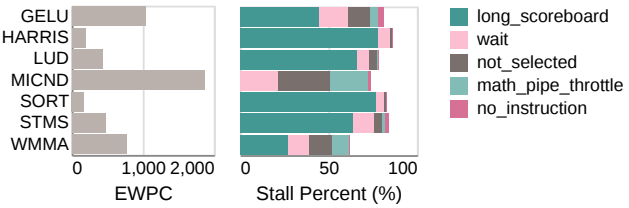


Fig. 4: Illustrative examples of `first` and `second` input kernel with coarse-grained (`VFuse`, `HFuse`) and fine-grained (`Desired`) fusion.



(a) Significant disparity of EWPC.

(b) Stall reasons breakdown, of which scoreboard contributes the most.

Fig. 5: EWPC and stall reasons distribution.

Algorithm 1 Control Flow Graph Merging

input: $F \neq \emptyset, S \neq \emptyset$ {the first and the second input CFG}
output: M {the merged CFG}
$M \leftarrow \emptyset$
$M.add_node(< 0, 0 >)$
$Stack \leftarrow \{< 0, 0 >\}$
$Pushed \leftarrow \{< 0, 0 >\}$
while $Stack \neq \emptyset$ **do**
  $< f, s > \leftarrow Stack.pop()$
  $M[< f, s >].block \leftarrow F[f].block + S[s].block$
  for $< u, v > \in get_next_list(< f, s >, F, S)$ **do**
    if $< u, v > \notin Pushed$ **then**
      $M.add_node(< u, v >)$
      $Stack.push(< u, v >)$
      $Pushed.insert(< u, v >)$
    end if
    $M.add_edge(< f, s >, < u, v >)$
  end for
end while
return M

B. Latency-Aware Instruction Weaving

Having fused the BBs from two kernels in Section III-A, we now focus on scheduling instructions from both input BBs within a limited time frame. However, finding the best instruction scheduling for a given basic block is a well-known NP-hard problem [11], [12] when the instruction dependencies form a directed acyclic graph (DAG) instead of a tree. To generate an optimized heuristic input for the `ptxas` backend compiler, we simplify the problem and propose a latency-aware instruction weaving method. We propose a latency-aware weaving algorithm based on the metrics measured in Paragraph III-D0b, which can be viewed as a greedy algorithm that takes instructions lists of the fusing BBs as input. As shown in Figure 8, in each iteration, `GoPTX` calculates the sum of instruction latencies for instructions already woven from each list, and selects a new instruction from the list with the lower sum. If the sums are equal, the instruction with the higher individual latency is chosen. The process continues until all instructions are woven.

C. Adaptive Code Slicing

The basic CFM approach can overlook weaving opportunities when block sizes differ significantly as the smaller block may not provide sufficient instructions to hide the latency of the longer instruction flow. To address this, we introduce code slicing, dynamically adjusting block sizes to balance latencies and maximize weaving potential. Slicing involves dividing blocks into smaller segments, ensuring each segment's execution time remains below a threshold, which is determined by the average number of execution cycles of all basic blocks, as shown in Equation 1. This strategy balances block sizes, maximizes weaving opportunities, and improves latency hiding.

$$threshold = \left\lceil \frac{\sum_{i=1}^{N_{blocks}} \sum_{j=1}^{M_{block_i}} latency(block_i, j)}{N_{blocks}} \right\rceil \quad (1)$$

D. Implementation

a) PTX Code Transformation: Due to the lack of official PTX processing tools from NVIDIA, we developed a custom PTX parser based on ANTLR4 [17]. This parser enables in-depth control flow and identifier analysis and allows us to rename identifiers from two kernels before CFG merging to prevent namespace conflicts. Compared to source code level work [13], [14], PTX-level transformation provides a closer abstraction to the underlying GPU hardware while maintaining a simple intermediate representation, benefiting instruction-level manipulation in `GoPTX`. Our design also avoids modifying existing CUDA C source code and executables, ensuring easy adaptability to existing code for current and future devices.

b) Instruction Latency Model: We measure instruction latency via microbenchmark [18] which records the average execution cycles of a single thread repeatedly executing the target PTX instruction. For shared memory and global memory accesses, we employ a pointer-chasing technique to ensure accurate latency measurements. This technique serializes memory accesses, preventing simultaneous multiple memory accesses that could lead to inaccurate measurements. The resulting latency profile enables precise performance predictions and optimizations specific to the GPU architecture, providing a versatile solution for instruction weaving.

c) Race Condition Avoiding: GPU thread synchronization instructions, such as barriers and tensorcore operations, can lead to deadlocks when weaving instructions from two kernels. As shown in Figure 9a, both the input CFGs contain blocks with synchronization instructions. However, one of the CFGs exhibits different threads performing synchronization operations in different branches, disrupting the synchronization order in the merged CFG, and leading to potential deadlocks. To address this issue (Figure 9b), during

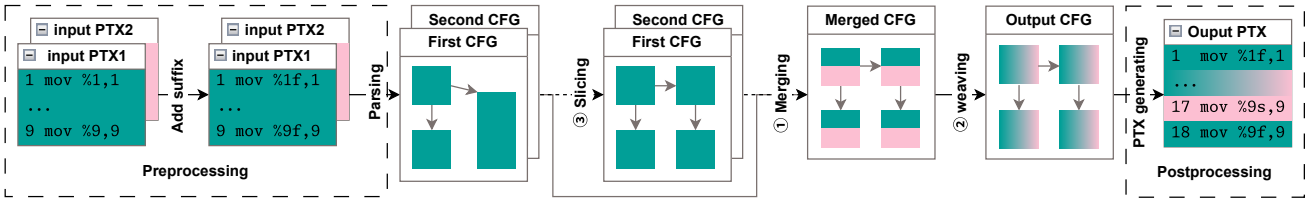
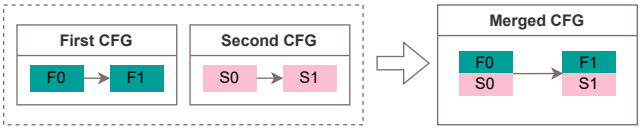
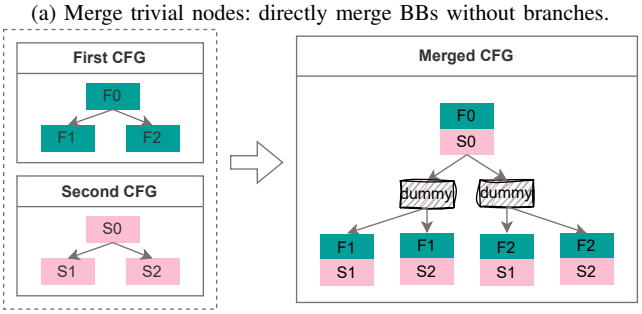


Fig. 6: Overall workflow of GoPTX, with three key phases of *Slicing*, *Weaving* and *Merging*, and additional phases of *Preprocessing* and *Postprocessing*.



(a) Merge trivial nodes: directly merge BBs without branches.



(b) Merge conditional branches: insert dummy nodes and branching for divergent control flow.

Fig. 7: Merging the control flow from two kernels.

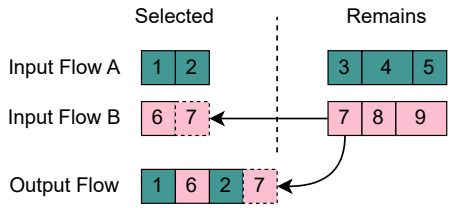


Fig. 8: The weaving process. Instruction 7 is selected because instruction 6's latency is lower than the sum of instruction 1 and instruction 2.

CFG merging, when encountering a block with synchronization instructions from the second kernel, we insert dummy blocks before it until the preceding block and its successors from the first kernel have no synchronization instructions. This ensures that threads do not stall indefinitely and preserves the original control flow, resulting in deadlock-free instruction weaving.

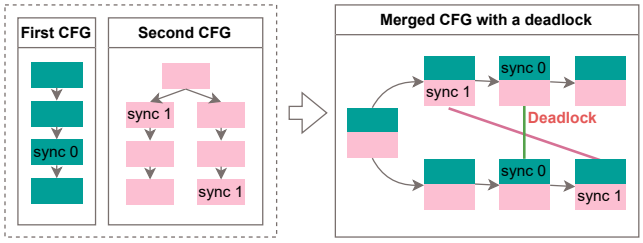
d) Register Tuning: CFG merging can increase register pressure, impacting occupancy and performance. Based on profiling, we measure the optimal `-maxrregcount` parameter in (32, 40, 48, 64, 80, 128, 256) to balance ILP and TLP.

e) Multiple Kernel Fusing: N-kernel fusion can be decomposed into sub-problems, e.g., (A+B)+C for 3-kernel fusion. However, previous studies [14] showed that merging more kernels diminishes returns due to the long tail effect.

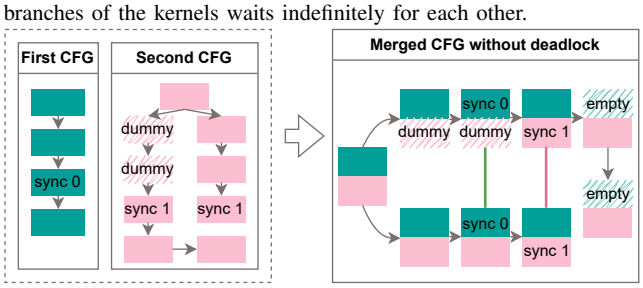
IV. Evaluation Methodology

A. Hardware and Software Platform

We conduct experiments on a server featuring AMD EPYC 7742 CPU, 256GB DRAM, and NVIDIA A100-PCIE-40GB GPU. We lock



(a) Deadlock occurs when the synchronization barrier in different branches of the kernels waits indefinitely for each other.



(b) Deadlock solved by inserting dummy blocks before synchronizations, delaying the barriers to prevent conflicts.

Fig. 9: Addressing race condition in the weaved kernel.

the GPU on max frequency 1410 MHz. The operating system is Ubuntu 24.04LTS, and CUDA driver version is 555.42.06. We use `nvcc` and `ptxas` shipped with CUDA 12.5.1, and `clang` 17.0.6 as the host compiler with compile option `-O3 -arch=sm_80`.

B. Workloads

We evaluate GoPTX using multiple representative kernels drawn from realistic applications like the Rodinia Benchmark Suite [19], ONNX-runtime [20], CUDA Samples [21], and HIPACC Samples [22]. Table I characterizes the kernels in terms of register count (Reg), arithmetic operations (INT, FP), synchronization barrier (Sync), shared memory (Smem) and Tensor Core use (TC). For evaluation, we pair each kernel with all others, including itself, to create concurrent execution cases per kernel. This showcases GoPTX's ability to transcend source code limitations and integrate seamlessly into existing scheduling systems [23], [24]. To maintain consistent experimental conditions, we adjust the kernels to employ a standardized launch configuration without compromising correctness or performance, as supported by previous studies [13], [14]. In detail, we set the *blockDim* to (256,1,1) and the *gridDim* to (1048576,1,1). For `HFuse`, the *blockDim* is specifically set to (512,1,1) since `HFuse` directly combines two kernels at the block level. This launch configuration fulfills A100's stream multiprocessors to isolate the impact of instruction weaving and other factors on overall performance.

TABLE I: Kernels and their resource requirements.

kernel	Time	Reg	INT	FP	Sync	Smem	TC
GELU [20]	1.96ms	19		✓			
HARRIS [22]	1.49ms	14	✓				
LUD [19]	3.68ms	30		✓	✓	✓	
MICND [21]	1.25ms	15		✓			
SORT [19]	3.19ms	17			✓	✓	
STMS [22]	1.63ms	12	✓	✓			
WMMA [21]	2.81ms	44		✓	✓		✓

C. Metrics

To evaluate the effectiveness of our approach, we collect statistics with the CUDA Profiling Tools Interface (CUPTI) [25], with metrics being listed in Table II. *EWPC* is the indicator we focus on, representing the average number of warps ready per clock cycle. The higher the value, the more instructions can be executed at the same time, and the higher the ILP. *AOC* is the number of active warps per cycle divided by the GPU's maximum supported, representing TLP. Other indicators measure the GPU hardware utilization.

TABLE II: Hardware metrics to reflect resource usages.

Metric	Note	Type
EWPC	eligible warps per cycle	ILP indicator
AOC	achieved occupancy	TLP indicator
DU	dram utilization	off-chip resource
L2U	l2 cache utilization	
L2HR	l2 cache hit rate	
ISU	issue slot utilization	on-chip resource
LSU	load-store unit utilization	
SPU	single precision unit utilization	
TSU	tex/l1 unified cache utilization	
THR	tex/l1 cache hit rate	

D. Comparing Schemes

We compare GoPTX with the following schemes shown in Figure 4 in our experiments: ❶ Baseline, launching two kernels in separate cuStreams to enable concurrent execution; ❷ VFuse [13], concatenating two kernels sequentially for fusion; ❸ HFuse [14], scheduling two kernels into different warps; ❹ GoPTX, our proposed design, merging control flow and weaving instructions.

V. Results and Analysis

A. Performance Improvement

Figure 10 shows the overall speedup achieved by GoPTX, which delivers an average of 11.2% speedup over the baseline, ranging from 23% (*STMS+LUD*) to -2% (*WMMA+GELU*). In contrast, VFuse and HFuse achieve moderate geometric mean speedups of 6.4% and 1%, respectively. While VFuse can achieve significant performance gains in certain compute-intensive benchmarks (*LUD*, *MICND*, *WMMA*), its benefits are less pronounced in others. HFuse, on the other hand, does not always lead to performance improvements as it requires extensive profiling to determine the optimal kernel combine ratio [14]. GoPTX significantly outperforms both HFuse and VFuse in the vast majority of benchmarks, except for *WMMA*. The *WMMA* kernel employs a large number of warp synchronization instructions to utilize tensor cores. To avoid deadlocks, GoPTX adopts a conservative merge strategy, which introduces overhead and leads to a slight performance decrease (0.05%) compared to VFuse.

Figure 11 showcases the impact of our approach on instruction-level parallelism (ILP), as measured by the number of eligible warps per cycle (EWPC). The data reveals that HFuse reduces EWPC by 14.4% in exchange for TLP improvement, while VFuse exhibits minimal changes in EWPC due to the sequential concatenation of kernels. In contrast, our approach demonstrably increases EWPC by 5.5% and max 50% (*STMS*, *LUD*) and min -35% (*WMMA*, *GELU*). The *SORT* benchmark presents unique cases. *SORT* is memory-bound, with a high volume of read operations and insufficient computation to effectively hide latency with other kernels. Consequently, despite performance gains achieved through inter-kernel resource sharing, ILP remains impacted by -12%. The distribution of EWPC acceleration across the benchmarks highlights the consistent effectiveness of GoPTX in enhancing ILP over existing techniques, and thus translates to performance gains, as demonstrated by the overall speedup presented in Figure 10.

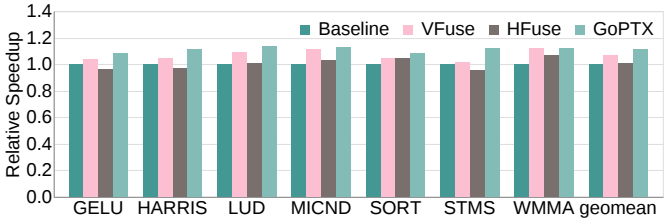


Fig. 10: Performance improvement.

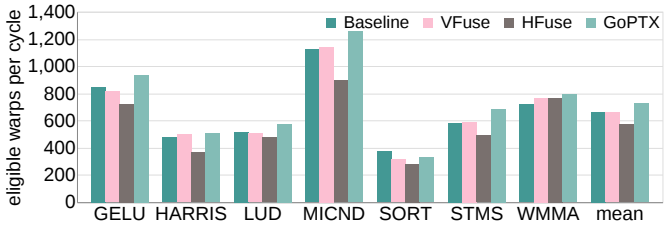


Fig. 11: Eligible warps per cycle comparison.

B. Resource Utilization

Figure 12 shows that VFuse and GoPTX improve occupancy (AOC) by 4% compared to the baseline, while HFuse suffers a 5.7% decrease because it launches more warps in a block waiting for others to finish. This suggests that register pressure did not significantly impact performance, as modern GPU compilers use advanced register allocation strategies to optimize even with high register usage. For example, *WMMA* uses 40 registers per block (75% theoretical occupancy), while *HARRIS* uses 14 registers (100% occupancy). Our weaved kernel uses 40 registers (75% occupancy). VFuse and GoPTX improve off-chip resource utilization, e.g DU by 3.7% and 6.5%, respectively, while HFuse shows no change. GoPTX also achieves the highest instruction throughput (ISU) increase (4.0%). For on-chip resources, VFuse and HFuse show increments but suffer from inconsistent performance gains due to warp competition. Overall, GoPTX's ability to improve ILP and hide latency leads to the best resource utilization and ISU among all evaluated techniques.

C. Case Study of Stalls

We analyze the impact of instruction weaving on warp stalls using *WMMA+GELU* and *WMMA+HARRIS*. These workloads share a common kernel (WMMA) but exhibit significantly different performance improvements. GoPTX achieves a 20% performance improvement and a 38% increase in EWPC for *WMMA+HARRIS*,

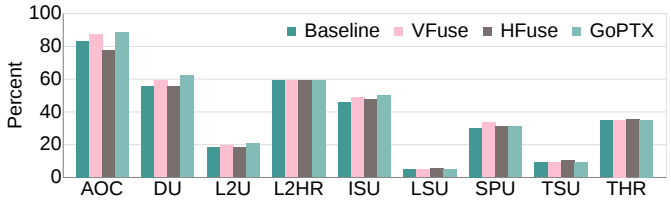


Fig. 12: Hardware resource utilization comparison.

while experiencing a -2% performance degradation and a -35% decrease in EWPC for *WMMA+GELU*. As Figure 13 illustrates, after `GoPTX` processing, *WMMA+HARRIS* experiences a 20.5% reduction in scoreboard stalls, while *WMMA+GELU* incurs a 30.4% increase (still better than `VFuse` and `HFuse`). For total stall cycles, *WMMA+HARRIS* sees a 34.5% reduction, while *WMMA+GELU* remains unchanged. This disparity can be attributed to the differing hardware resource contention between the kernels involved, as Table I shows. Both WMMA and GELU involve a significant amount of half-precision computation, leading to contention for ALU resources. In contrast, WMMA and HARRIS (primarily integer computations) exhibit complementary computation types. This result highlights the importance of resource complementarity for ILP enhancement.

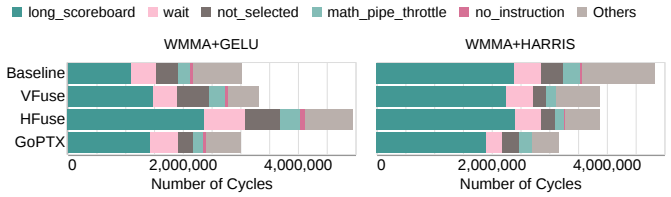


Fig. 13: Warp stalled cycles and reasons.

D. Performance Breakdown

a) Contributions of Merging, Weaving and Slicing: To isolate the effects of each technique, we conducted a series of experiments where we selectively enabled weaving and slicing. Even without both, control flow optimization alone achieves a performance improvement of 9.7%. This enhancement is attributed to the effective sharing of resources within the SM and the instruction reordering capabilities of the `ptxas` backend. Slicing incurs a slight decrease of 0.6% while creating opportunities for weaving to achieve a significant performance improvement of 2.1%. Without slicing, weaving only improves 0.3%. These findings underscore the necessitity of both weaving and slicing in facilitating instruction parallelism.

b) Threshold for Code Slicing: We compared our adaptive slicing algorithm with fixed-threshold approaches, where the slicing threshold ranged from 256 to 1024. As evident from Figure 14, the effect of the threshold varies significantly and no single value is universally optimal. Our adaptive algorithm outperforms the fixed-threshold approach in most cases. However, the WMMA benchmark exhibits better performance with the fixed-threshold approach due to the presence of tensor core instructions, which occur at fixed intervals. This highlights the sensitivity of the fixed-threshold approach to workload characteristics. In contrast, our adaptive approach dynamically adjusts the slicing threshold, making it more robust and adaptable to a wider range of workloads.

VI. Related Work

a) Concurrent Kernel Execution: Methods based on cuStream, MPS, or MIG [23], [26], [27] focus on scheduling and optimizing

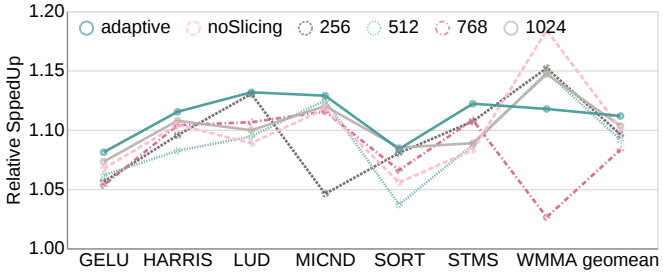


Fig. 14: Speedup of the threshold for code slicing, compared to our adaptive algorithm.

multiple kernels but don't fully address resource under-utilization [3], [28], [29]. Xu *et al.* propose Warped-Slicer [28], which dynamically and statically schedules thread blocks from different kernels to a single SM. Wang *et al.* introduce SMK [29], which uses preemption for block-level scheduling by efficiently dividing resources. Although these approaches can enhance utilization, Dai *et al.* [3] have shown that intra-SM sharing schemes may reduce overall performance due to substantial interference between kernels.

b) Code Combination Techniques: Guevara *et al.* [30] and Gregg *et al.* [31] combine kernels at the thread block level, which can incur significant overhead. Wang *et al.* [13] suggest three thread-level fusion strategies, but their method cannot handle synchronization [4], [13]. Li *et al.* [14] present automatic horizontal fusion, allowing GPUs to distribute instructions to different warps. Even with the above designs, severe kernel interference remains unresolved, and our evaluation demonstrates that `GoPTX` outperforms those prior designs on parallelism enhancement.

c) GPU ILP Optimization: Shobaki *et al.* [11] propose a compile-time Branch-and-Bound algorithm to balance ILP and occupancy, while their later work [12] parallelizes instruction scheduling, further enhancing occupancy and reducing schedule length. WASP [32] introduces warp specialization to enable parallel tasks within a block, boosting utilization. These approaches generally overlook resource sharing and latency hiding, thereby limiting ILP and utilization across kernels constrained by hardware units, while our fine-grained `GoPTX` method addresses this by enabling resource sharing to enhance ILP.

VII. Summary

Targeting at raising instruction-level parallelism of GPU executions, this paper presents `GoPTX` to weave instructions from two different kernels, serving as a fine-grained kernel fusion at PTX level. The design overall encompasses CFG merging, instruction weaving and code slicing to generate highly efficient codes for better utilization of the underlying resources. We implement the prototype of `GoPTX` based on NVIDIA off-the-shelf CUDA software stack, with particular handling of race conditions and deadlock. Experimental results on representative kernels demonstrate that `GoPTX` effectively improves performance with higher ILP and utilization.

Acknowledgment

We thank the anonymous reviewers for their constructive comments and suggestions. This research was supported by the National Natural Science Foundation of China-#62472462/#62461146204, and sponsored by CCF-Tencent Rhino-Bird Open Research Fund (CCF-Tencent RAGR20240102).

REFERENCES

[1] "NVIDIA Multi-Process Service (MPS)." http://docs.nvidia.com/deploy/mps/index.html.

[2] "NVIDIA Multi-Instance GPU (MIG)." https://www.nvidia.com/en-sg/technologies/multi-instance-gpu/.

[3] H. Dai, Z. Lin, C. Li, C. Zhao, F. Wang, N. Zheng, and H. Zhou, "Accelerate gpu concurrent kernel execution by mitigating memory pipeline stalls," in 2018 IEEE International Symposium on High Performance Computer Architecture (HPCA), pp. 208–220, 2018.

[4] S. Pai, M. J. Thazhuthaveetil, and R. Govindarajan, "Improving gpgpu concurrency with elastic kernels," in Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS '13, (New York, NY, USA), p. 407–418, Association for Computing Machinery, 2013.

[5] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. K. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, S. Zhang, M. Suo, P. Tillet, X. Zhao, E. Wang, K. Zhou, R. Zou, X. Wang, A. Mathews, W. Wen, G. Chanan, P. Wu, and S. Chintala, "Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation," in Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2, ASPLOS '24, (New York, NY, USA), p. 929–947, Association for Computing Machinery, 2024.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: a system for large-scale machine learning," in Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation, OSDI'16, (USA), p. 265–283, USENIX Association, 2016.

[7] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: an automated end-to-end optimizing compiler for deep learning," in Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation, OSDI'18, (USA), p. 579–594, USENIX Association, 2018.

[8] Z. Ye, R. Lai, J. Shao, T. Chen, and L. Ceze, "Sparsetir: Composable abstractions for sparse compilation in deep learning," in Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, ASPLOS 2023, (New York, NY, USA), p. 660–678, Association for Computing Machinery, 2023.

[9] Y. Gui, Y. Wu, H. Yang, T. Jin, B. Li, Q. Zhou, J. Cheng, and F. Yu, "Hgl: accelerating heterogeneous gnn training with holistic representation and optimization," in SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–15, IEEE, 2022.

[10] H. Zhao, W. Cui, Q. Chen, Y. Zhang, Y. Lu, C. Li, J. Leng, and M. Guo, "Tacker: Tensor-cuda core kernel fusion for improving the gpu utilization while ensuring qos," in 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 800–813, 2022.

[11] G. Shobaki, A. Kerbow, and S. Mekhanoshin, "Optimizing occupancy and ilp on the gpu using a combinatorial approach," in Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization, CGO 2020, (New York, NY, USA), p. 133–144, Association for Computing Machinery, 2020.

[12] G. Shobaki, P. Muyan-Özçelik, J. Hutton, B. Linck, V. Malyshenko, A. Kerbow, R. Ramirez-Ortega, and V. Gordon, "Instruction scheduling for the gpu on the gpu," pp. 435–447, 03 2024.

[13] G. Wang, Y. Lin, and W. Yi, "Kernel fusion: An effective method for better power efficiency on multithreaded gpu," in 2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing, pp. 344–350, 2010.

[14] A. Li, B. Zheng, G. Pekhimenko, and F. Long, "Automatic horizontal fusion for gpu kernels," in 2022 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp. 14–27, 2022.

[15] J. Wu, A. Belevich, E. Bendersky, M. Heffernan, C. Leary, J. Pienaar, B. Roune, R. Springer, X. Weng, and R. Hundt, "gpucc: an open-source gpgpu compiler," in Proceedings of the 2016 International Symposium on Code Generation and Optimization, CGO '16, (New York, NY, USA), p. 105–116, Association for Computing Machinery, 2016.

[16] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization, CGO '04, (USA), p. 75, IEEE Computer Society, 2004.

[17] A. community, "grammars-v4." https://github.com/antlr/grammars-v4/blob/753536777d827ccc0c9b108531ea67375c2039ac/asm/ptx/ptx-isa-2.1/Ptx.g4, 2023.

[18] H. Abdelkhalik, Y. Arafa, N. Santhi, and A.-H. A. Badawy, "Demystifying the nvidia ampere architecture through microbenchmarking and instruction-level analysis," 2022 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–8, 2022.

[19] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in 2009 IEEE International Symposium on Workload Characterization (IISWC), pp. 44–54, 2009.

[20] ONNX Runtime developers, "ONNX Runtime," Nov. 2018.

[21] NVIDIA, "Cuda samples." https://github.com/NVIDIA/cuda-samples/tree/v12.5, 2024.

[22] B. Qiao, O. Reiche, F. Hannig, and J. Teich, "From loop fusion to kernel fusion: A domain-specific approach to locality optimization," in 2019 IEEE/ACM International Symposium on Code Generation and Optimization (CGO), pp. 242–253, 2019.

[23] J. Kim, J. Kim, and Y. Park, "Navigator: Dynamic multi-kernel scheduling to improve gpu performance," in 2020 57th ACM/IEEE Design Automation Conference (DAC), pp. 1–6, 2020.

[24] H. Wu, Y. Yu, J. Deng, S. Ibrahim, S. Wu, H. Fan, Z. Cheng, and H. Jin, "StreamBox: A lightweight GPU SandBox for serverless inference workflow," in 2024 USENIX Annual Technical Conference (USENIX ATC 24), (Santa Clara, CA), pp. 59–73, USENIX Association, July 2024.

[25] NVIDIA, "Cupti 12.5 documentation." https://docs.nvidia.com/cupti/index.html, 2024.

[26] J. Zhong and B. He, "Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling," IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 6, pp. 1522–1532, 2014.

[27] Z. Lin, Z. Mo, X. Huang, X. Zhang, and Y. Lu, "Kesco: Compiler-based kernel scheduling for multi-task gpu applications," in 2023 IEEE 41st International Conference on Computer Design (ICCD), pp. 247–254, IEEE, 2023.

[28] Q. Xu, H. Jeon, K. Kim, W. W. Ro, and M. Annavaram, "Warped-slicer: Efficient intra-sm slicing through dynamic resource partitioning for gpu multiprogramming," ACM SIGARCH Computer Architecture News, vol. 44, no. 3, pp. 230–242, 2016.

[29] Z. Wang, J. Yang, R. Melhem, B. Childers, Y. Zhang, and M. Guo, "Simultaneous multikernel gpu: Multi-tasking throughput processors via fine-grained sharing," in 2016 IEEE international symposium on high performance computer architecture (HPCA), pp. 358–369, IEEE, 2016.

[30] M. Guevara, C. Gregg, K. M. Hazelwood, and K. Skadron, "Enabling task parallelism in the cuda scheduler," 2009.

[31] C. Gregg, J. Dorn, K. Hazelwood, and K. Skadron, "Fine-grained resource sharing for concurrent gpgpu kernels," in Proceedings of the 4th USENIX Conference on Hot Topics in Parallelism, HotPar'12, (USA), p. 10, USENIX Association, 2012.

[32] N. C. Crago, S. Damani, K. Sankaralingam, and S. W. Keckler, "Wasp: Exploiting gpu pipeline parallelism with hardware-accelerated automatic warp specialization," in 2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA), pp. 1–16, 2024.

UniCoS: A Unified Neural and Accelerator Co-Search Framework for CNNs and ViTs

Wei Fu[1*], Wenqi Lou[2*†], Cheng Tang[1,2], Hongbing Wen[1,2], Yunji Qin[1], Lei Gong[1], Chao Wang[1†], Xuehai Zhou[1]

[1] University of Science and Technology of China, China.

[2] Suzhou Institute of Advanced Research, University of Science and Technology of China, China.

{louwenqi,cswang}@ustc.edu.cn

ABSTRACT

Current algorithm-hardware co-search works often suffer from lengthy training times and inadequate exploration of hardware design spaces, leading to suboptimal performance. This work introduces UniCoS, a unified framework for co-optimizing neural networks and accelerators for CNNs and Vision Transformers (ViTs). By introducing a novel training-free proxy that evaluates accuracy within seconds and a clustering-based algorithm for exploring heterogeneous dataflows, UniCoS efficiently navigates the design spaces of both architectures. Experimental results demonstrate that the solutions generated by UniCoS consistently surpass state-of-the-art (SOTA) methods (*e.g.*, 3.54× energy-delay product (EDP) improvement with a 1.76% higher accuracy on ImageNet) while requiring notably reduced search time (up to 48×, ~3 hours). The code is available at https://github.com/mine7777/Unicos.git.

KEYWORDS

zero-shot proxy, heterogeneous multi-core architecture, hardware software co-exploration

1 INTRODUCTION

In recent years, CNNs and ViTs have become dominant backbones in computer vision, driving intense competition in the field [6]. Extensive research explores the design spaces to improve performance [24], yet identifying a definitive superior architecture remains challenging. Simultaneously, these models' high computational and storage demands have driven the development of specialized hardware accelerators, such as Application-Specific Integrated Circuits (ASICs) [4, 8, 22] and Field-Programmable Gate Arrays (FPGAs) [10, 20, 28], optimized for specific models to enhance performance. Recent research highlights the interdependence between hardware optimization and network architectural design, showing that focusing exclusively on either aspect often leads to suboptimal solutions [11, 19].

Consequently, neural-accelerator co-search has been developed to explore better customization and acceleration under predefined accuracy and resource constraints [5, 29]. These approaches typically employ an iterative *two-stage* search strategy to manage the exponential complexity of algorithmic and hardware parameter spaces. In the first stage, candidate networks (also referred to as subnets) are evaluated for accuracy using a one-shot method, which shares operations through a supernet to eliminate the need for separate training of each subnet. In the second stage, the subnets are evaluated against hardware constraints, incorporating feedback on hardware-related losses. This iterative process continues, leveraging accuracy and hardware loss feedback from each epoch to guide the joint optimization.

Table 1: Comparison with Prior Hardware Search or Co-search Work. T&P stands for Tile and Parallism.

Work	Neural Architecture Search			Hardware Dataflow Search		
	CNN	ViT	Train-free	T&P Size	Loop Order	Multicore
GAMMA [12]	–	–	–	✓	✓	✗
HPCA21 [15]	–	–	–	✓	✓	✓
DIAN [29]	✓	✗	✗	✓	✓	✗
DANCE [5]	✓	✗	✗	✓	✓	✗
NAF [21]	✓	✗	✗	✓	✗	✓
Ours	✓	✓	✓	✓	✓	✓

However, current neural-accelerator co-search methodologies face severe limitations that hinder their optimization potential. *Algorithmic Limitations*: network accuracy evaluation depends on supernet training, which is time-consuming and highly specific to the neural search space. For example, training a supernet for CNNs requires about 100 GPU hours, while ViTs demand up to 576 GPU hours [3]. Moreover, any modification to the search space necessitates retraining or redesigning the supernet. *Hardware Limitations*: The exploration of the hardware design space remains constrained. Existing approaches mainly focus on single-core accelerators with a single dataflow (*e.g.*, tile size and loop order), failing to accommodate the computational heterogeneity across model layers [15]. While developing large accelerators with multiple sub-accelerators or chiplets is gaining traction, it further complicates hardware optimization [2].

To this end, we propose **UniCoS**, a fast and **Uni**fied framework for the **Co-S**earch of neural architectures and accelerators to explore customized CNNs/ViTs and their corresponding ASIC accelerators. Table 1 summarizes the key features of UniCoS and highlights its distinctions from existing approaches. First, we propose a proxy metric on the algorithmic side that eliminates the need for training. This metric utilizes gradient information from the model to rank the accuracy of neural architectures, avoiding the design and training burdens associated with supernets. Second, on the hardware side, we push beyond searching only single-core accelerators by introducing a heterogeneous dataflow design space. Then, we quickly obtain optimized solutions under an expanded design space using layer cluster and heuristic search methods.

DAC '25, June 22–25, 2024, San Francisco, CA, USA
2025. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn
*Equal contribution. †Corresponding authors.

The major contributions of this paper are as follows:

- We propose a training-free co-exploration framework for accelerators and network architectures, delivering superior application performance. To the best of our knowledge, this framework is the fastest fully open-source co-search solution currently available.
- We develop a zero-cost accuracy evaluation method that assesses the accuracy of a given network within seconds, with adaptability across diverse neural architecture search spaces.
- We integrate a novel heterogeneous dataflow exploration into the framework, utilizing an efficient heuristic-based search method that completes on the desktop-class CPU in just over ten seconds.
- Experimental results show that, for CNN backbones on ImageNet, the network-accelerator solutions generated by UniCoS deliver 3.54× and 12.96× EDP improvements compared to DIAN [29] and DANCE [5] frameworks, respectively. For ViT design space, UniCoS also achieves a favorable accuracy-EDP trade-off.

2 MOTIVATION

Accuracy Assessment Overhead. In one-shot NAS, the supernet serves as an over-parameterized hyper-network, incorporating all candidate operators at each layer. This design results in a linear increase in the size and training overhead of the supernet with operator complexity and quantity. While this approach mitigates the cost of accuracy evaluation, it still demands substantial training data and computational resources (hundreds of GPU hours) [26]. Furthermore, the supernet's architecture is tightly coupled with the search space, presenting challenges when adapting to varying network design requirements. For instance, modifying the number of candidate operators necessitates retraining the supernet, while altering the types of operators (*e.g.*, introducing transformer operators) may demand a complete redesign of the supernet with a larger dataset for training. Thus, an efficient and plug-and-play method for accuracy assessment is imperative to overcome these limitations.

Hardware Customization Overhead. In existing co-search research, the hardware design space is typically confined to a single dataflow accelerator. However, variations in layer shapes and operators demand different dataflow styles, always leading to software/hardware mismatches on the single accelerator design [15]. To address this, we introduce a heterogeneous dataflow architecture with multiple sub-accelerators in our co-search framework to better balance hardware customization and resource efficiency. While conceptually straightforward, integrating heterogeneous dataflows poses challenges in controlling hardware evaluation costs while exploring an expanded design space. Unlike previous dataflow explorations, co-search requires fast hardware evaluations for numerous subnets, making previous long search times impractical (tens of minutes) [13]. Thus, efficient HW design space exploration is essential for integrating heterogeneous dataflows.

3 CO-SEARCH OVERVIEW

Our proposed co-search framework, shown in Fig. 1, offers a comprehensive solution for designing task-specific neural networks and accelerators to meet user-defined hardware constraints and dataset specifications.

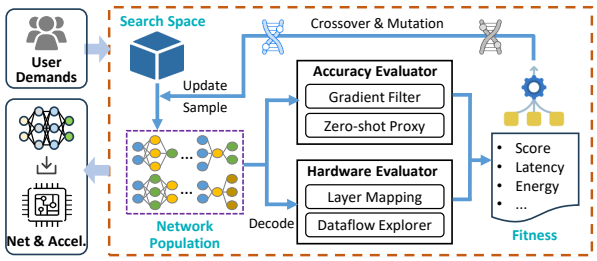


Figure 1: The overview of UniCoS framework.

The exploration process employs an evolutionary algorithm within a well-defined search space incorporating CNN or ViT backbones, where candidate network architectures are represented as individuals. Upon decoding, two specialized evaluators rigorously assess each sampled network for accuracy and hardware performance. In contrast to previous approaches that evaluate network accuracy by inheriting supernet weights, we introduce a training-free *accuracy evaluator* to estimate network performance (Section 4.1). This evaluator computes a zero-shot proxy score derived from gradient information filtered from a mini-batch of training images, ranking networks based on gradient consistency. Notably, this process is completed within mere seconds.

Meanwhile, the *hardware evaluator* assesses the network's optimal achievable performance under hardware resource constraints (Section 4.2). First, network layers are clustered based on their computational characteristics, and then resources are allocated to these clusters using a greedy strategy, completing the layers to sub-accelerators mapping. Subsequently, heuristic algorithms combined with pruning strategies are employed to efficiently explore the extensive heterogeneous dataflow design space (within tens of seconds). Finally, based on the results from the two evaluators,UniCoS calculates the fitness score for each individual in the network population, defined as $\text{Fitness} = \frac{\text{Proxy Score}}{\text{Energy-Delay Product}}$. The population's distribution is then updated through *selection*, *crossover*, and *mutation* processes.

4 PROPOSED METHOD

4.1 Accuracy Proxy Design

Fig. 2 illustrates the workflow of our proposed accuracy evaluator and emphasizes its key distinction from conventional one-shot supernet training methods. Specifically, our approach utilizes proxy metrics to eliminate the substantial computational overhead of training on the entire dataset, a critical advantage of this method. In contrast, the supernet, which is significantly larger than subnets, typically requires extensive training on the entire dataset until convergence. This process demands considerable computational resources and storage capacity from the training platform.

Inspired by [17, 30], we propose a zero-shot proxy for evaluating neural networks, eliminating the design and training overhead associated with supernets at minimal computational cost. We employ a gradient-based evaluation to assess a network's performance

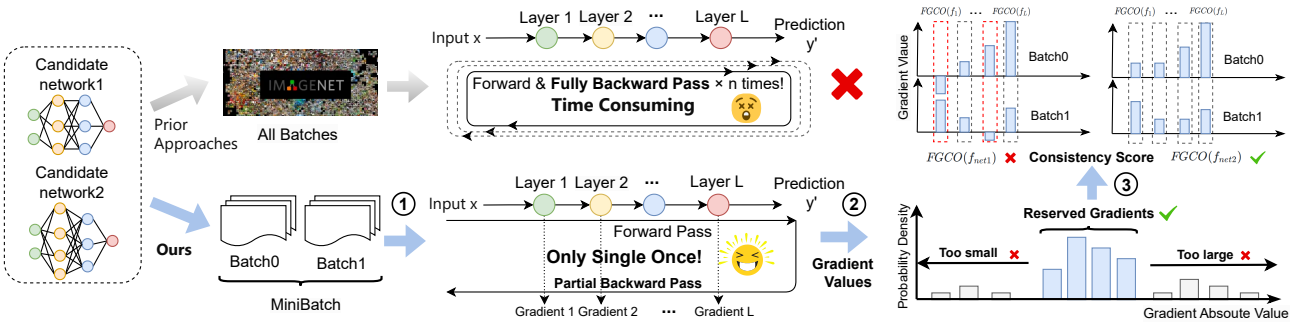


Figure 2: The workflow (①-③) and key features of accuracy evaluator.

potential during training. The network is scored efficiently by extracting gradients from a single forward and backward pass on a mini-batch. This approach significantly reduces the time required for evaluation, as illustrated at the bottom of Fig. 2.

① *Calculate Gradient*. In the first step, we employ Eq. 1 to compute the gradient information for the current batch of data. Let $f(x;\theta)$ denote the candidate network under evaluation, where θ represents the network parameters, and x is the input. Define $Batch = \{(x_1, y_1), ..., (x_{bs}, y_{bs})\}$ as the set of data points in the training batch, where x_i, y_i, and bs denote the i-th input, the label, and the batch size, respectively. θ_j denote the j-th weight tensor.

$$grad(\theta_j, Batch) = \frac{1}{bs}\sum_{i=1}^{bs}\frac{\partial Loss(y_i, f(x;\theta))}{\partial \theta_j} \tag{1}$$

② *Gradient Filter*: Although gradient information can serve as a valuable indicator of convergence, we argue that indiscriminately leveraging all gradients may introduce evaluation bias, as not all gradients contribute equally to network convergence. We categorize excessively large or small gradients as gradient noise to mitigate this issue, as shown in Fig. 2. To identify the noise range for each network architecture, we adopt a fine-grained gradient preprocessing technique to filter out noisy information before employing the zero-shot proxy for evaluation. Specifically, for each network, all gradients are collected and divided into intervals based on their absolute values. The interval with the high gradient density is selected for proxy computation, while gradients from other intervals are excluded.

③ *Zero-shot Proxy* : Using the **F**iltered gradients, our proxy computes the final score based on **G**radient **CO**nsistency, referred to as FGCO. Specifically, we evaluate the consistency of filtered gradients across different batches to capture the oscillation trend during weight updates. Let $Batch_k, k \in [1, n]$ denote one of the mini-batches. The fundamental gradient consistency proxy is then formulated as follows:

$$\text{FGCO}(f(x;\theta)) = \sum_{i=1}^{L} log(\sum_{\theta_j \in \Theta_i}\frac{|E_{Batch_k} fg(\theta_j, Batch_k)|}{\sqrt{Var_{Batch_k} fg(\theta_j, Batch_k)}}) \tag{2}$$

Here, L represents the number of network layers, Θ encompasses all network parameters, and Θ_i denotes the parameters of i-th layer. fg means the filter function.

We design the metric based on the consistency of gradients obtained from different batches. Specifically, we calculate this by dividing the variance of the filtered gradients of each layer by their mean and then accumulating the consistency across all layers. Notably, we retain the sign of the gradients before summing them to capture the oscillation tendencies in gradient directions and weight updates. This metric reflects the expectation that training data from different batches should produce consistent gradients for a single parameter. Higher gradient consistency indicates reduced oscillations in parameter updates, which can lead to faster convergence.

In our experiments, we observed that even using data from just two batches ($n = 2, bs = 32$) is sufficient to achieve strong performance.

4.2 Hardware Dataflow Search

During the hardware evaluation phase, the input is a neural network architecture, and the output includes the hardware performance metrics under its best-fit configuration. As shown in Fig. 3, the hardware exploration process is divided into two stages to navigate the extensive design space efficiently: 1) *Inter-core mapping*: Neural network operators (*i.e.*, layers), the fundamental computational units, are assigned to different computational cores (sub-accelerators). 2) *Intra-core Exploration*: Once operators are allocated, each core's workload is determined, and a sub-evolution algorithm optimizes the dataflow within each core. Below, we will provide a detailed description of each stage.

4.2.1 Inter-core Mapping. Neural network computations are abstracted into a sequence of operator executions. These operators are represented using layer parameters: K, C, R, S, X, Y, where K and C denote the number of output and input channels, R and S the kernel dimensions, and X and Y the input feature map dimensions. While convolution operations are directly expressed using these dimensions, matrix multiplications and fully connected layers are modeled as convolutions with a kernel size of 1 ($R\&S = 1$).

Then, operators are clustered in a multi-dimensional space based on their computational parameters, which dictate their behavior on dataflow accelerators. Clustering is performed as follows: 1) When the number of cores is a search parameter, the DBSCAN algorithm [9] determines clusters efficiently in milliseconds. 2) For a given fixed number of cores, the K-means algorithm identifies clusters. This clustering method is adaptable and can be substituted

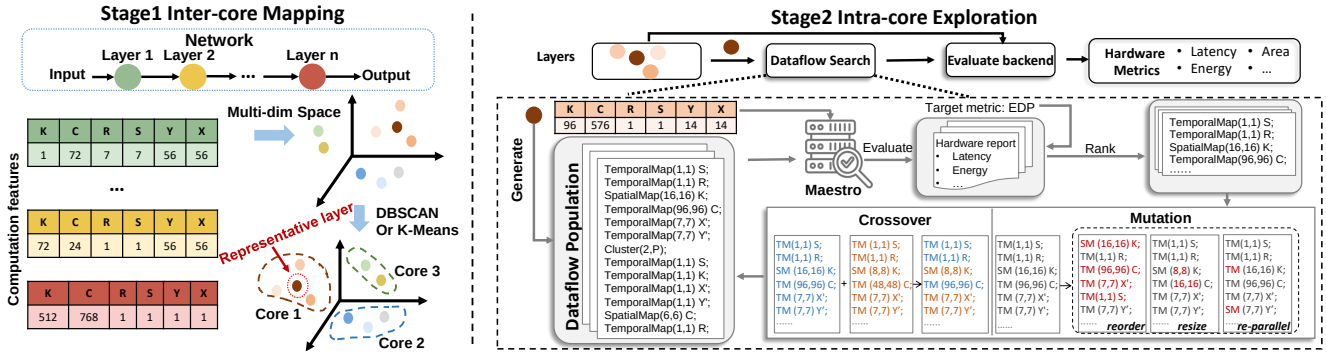


Figure 3: Hardware evaluator workflow with inter-core mapping (left) and intra-core exploration (right).

as needed. Once clustering is complete, hardware resources are allocated proportionally to the computational load of each core.

4.2.2 Intra-core Exploration. With operators clustered and computational resources allocated, the next step involves optimizing the hardware dataflow for each core. Dataflow is characterized by time and space reuse across computational units, described in a data-oriented manner, Maestro [14]. Among that, time mapping (`TemporalMap`) refers to the scheduling of data, while space mapping (`SpatialMap`) pertains to the parallelization of computation. By abstracting time and space reuse, the description of a computational core can be simplified to the reuse strategies and the degree of reuse applied across different dimensions.

A genetic algorithm is employed for this exploration, where each dataflow configuration represents an individual. The evaluation backend computes the hardware performance of each individual. During the mutation phase, parameters such as dimension ordering, reuse factors (*e.g.*, tiling and unrolling sizes), and parallelization strategies are adjusted iteratively to enhance performance.

4.2.3 Design Space Pruning. As mentioned earlier, the hardware design space is vast, with exponential growth in complexity. For instance, a single layer with dimensions (1280, 640, 7, 7, 56, 56) results in a search space of approximately $O((7 \times 7 \times 1280 \times 640 \times 56 \times 56 \times 6! \times 6)^2) \approx O(3 \times 10^{29})$, encompassing loop order, tiling size, and parallel dimensions. For multi-core systems with multiple layers, this complexity escalates dramatically. Evaluating a single network may take tens of minutes [13], and the exploration time grows linearly with the number of cores. To accelerate the process, we implement two pruning strategies:

Value-based Pruning. This approach refers to the use of multiples of 4 when considering dataflow reuse dimensions. This approach not only aligns better with the design logic but also effectively reduces the search space.

Strategy-Based Pruning: Instead of evaluating every layer in a cluster, which results in excessively long times, representative layers are selected for each cluster, and the optimal strategy is derived based on these points. This strategy is subsequently extended to the entire cluster, serving as the final optimized solution.

These strategies significantly reduce the search cost, enabling the hardware evaluator to complete evaluations within 15 seconds on a desktop CPU (see Section 5.3), compared to the tens of minutes required in previous studies.

Table 2: Kendall's τ and Spearman's ρ of different proxies on NASBench201 and AutoFormer design space. Img16 means ImageNet16-120.

Benchmark	NASBench-201						Benchmark	AutoFormer	
Dataset	CIFAR10		CIFAR100		ImagNet16		Dataset	ImageNet-1k	
Cor	K	S	K	S	K	S	Cor	K	S
ZenScore	0.29	0.38	0.28	0.36	0.29	0.40	Snip	0.46	0.66
ZiCo	0.61	0.80	0.61	0.81	0.60	0.79	ZiCo	0.62	0.80
GradSign	0.58	0.77	0.59	0.79	0.59	0.78	DSS	0.20	0.30
#Param	0.57	0.75	0.55	0.73	0.52	0.69	Param	0.52	0.72
#FLOP	0.54	0.73	0.51	0.71	0.49	0.67	FLOPs	0.52	0.73
Ours	**0.655**	**0.839**	**0.658**	**0.842**	**0.627**	**0.814**	**Ours**	**0.638**	**0.820**

5 EXPERIMENTS

5.1 Experimental Setup

All experiments were conducted on a desktop-level Intel i5-13400 CPU and an NVIDIA A6000 GPU with 48 GB of video memory. Hardware emulation was performed using MAESTRO [14], a cost simulator validated through RTL simulations for DNN accelerators with flexible dataflows. To evaluate latency, energy, power, and area, the simulator was extended to support multi-sub-accelerator configurations, following methodologies established in prior research.

5.2 Accuracy Evaluator Validation

We ranked networks based on zero-shot proxy scores and evaluated their correlation with true accuracy rankings using Kendall's τ (K) and Spearman's ρ (S), standard correlation coefficients ranging from [-1, 1]. Values closer to 1 indicate stronger agreement between the proxy-based and actual accuracy rankings.

Benchmarks: For the CNN design space, we used the NASBench-201 dataset [7], which comprises 15,625 unique architectures with diverse topologies and test accuracy data for CIFAR10, CIFAR100, and ImageNet16-120. In the ViT design space, we randomly sampled 1,000 networks and obtained their accuracies from the Autoformer design space.

Baselines: We compared our proxy against existing zero-shot NAS proxies: ZiCo [17], Zen [18], and GradSign [30] for CNNs, and SNIP [16], ZiCo [17], and DSS [31] for ViTs. Additionally, we included parameter (#Param) and FLOP count (#FLOP) as baselines, as they are critical metrics for neural network evaluation.

Table 3: Test accuracy (%) of searched architectures on NASBench-201 with different zero-shot proxies.

	Optimal Acc.	Naswot	ZiCo	GradSign	#Param	**Ours**
CIFAR10	94.37	92.96	94.0	93.31	93.80	**94.21**
CIFAR100	73.51	69.98	71.1	70.33	71.11	**72.91**
ImageNet16	47.31	44.44	41.8	42.42	41.44	**46.34**

Table 4: The performance and cost of architecture searched in the AutoF-T design space.

Model	#Param (M)	Top-1 Acc. (%)	Top-5 Acc. (%)	GPU Days
Deit-T [23]	5.7	72.2	91.1	Manual
PVT-T [25]	13.2	75.1	-	Manual
AutoFormer-Tiny [3]	5.7	74.7	92.6	24
Ours	**6.7**	**75.39**	**92.85**	**0.05**

Results: Table 2 demonstrates the strong correlation between our proxy scores and model accuracy. Specifically, Kendall's τ coefficients exceed 0.8 on NASBench-201 and AutoFormer design space, surpassing previous zero-shot methods across all datasets. These results highlight the robustness of our proxy in maintaining high accuracy correlation across diverse search spaces, including CNN and ViT backbones. This performance stems from our effective gradient-based filtering and analysis. As a network evaluation metric, our approach delivers accuracy-related scores within seconds, ensuring both reliability and significant efficiency improvements for collaborative search.

Case Study: We evaluated our accuracy predictor through a NAS experiment on the NASBench-201 and AutoFormer-Tiny (AutoF-T) design spaces. The results, summarized in Tables 3 and 4, highlight the effectiveness of our approach.

For the CNN backbone, our zero-shot proxy identified an architecture achieving test accuracy comparable to the highest reported in the dataset. Notably, it outperformed prior proxy methods, demonstrating its efficacy in discovering optimal architectures. For the ViT backbone, within one hour, our method identified high-accuracy networks that surpassed both existing one-shot NAS methods and manually designed architectures, showcasing its generalizability to diverse search spaces.

5.3 Hardware Evaluator Validation

Baselines: The hardware dataflow search in this framework utilizes a heterogeneous multi-core architecture optimized for search efficiency. We compare our work with single-core-based exploration and typical partition-based exploration. For CNNs, the typical partition strategy is based on resolution stages. For ViTs, the typical partition strategy is based on dimensions of matrix multiplication. We also verify the time reduction brought by our design space pruning.

Results: Fig. 4 presents each approach's final energy-delay product (EDP) reductions and search times. Leveraging the heterogeneous dataflow architecture, we significantly optimize hardware efficiency, achieving EDP reductions ranging from 4.3× to 192.6× for CNNs and 2.7× to 235.8× for ViTs. These substantial gains illustrate the ability of our hardware evaluator to identify optimized designs tailored to the operator's computation features. Notably,

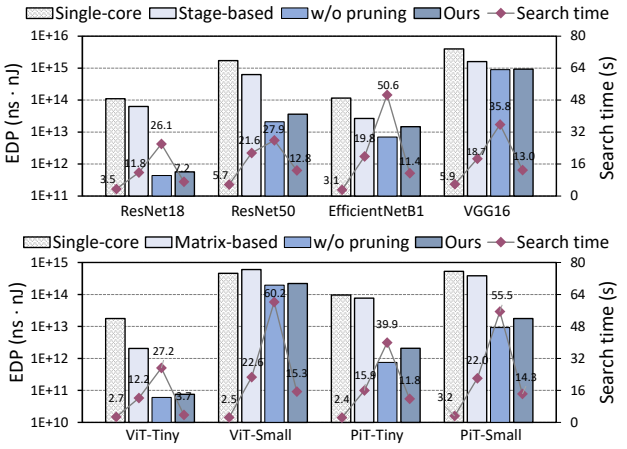


Figure 4: Comparison of the hardware search results and the time cost by different search strategies target on CNNs (top) and ViTs (bottom).

the improvements are more pronounced in smaller networks, where the limited number of layers amplifies the impact of mapping and dataflow strategies on hardware efficiency.

Additionally, our search strategy achieves comparable runtimes to single-core methods while significantly reducing EDPs. This efficiency stems from a design space pruning strategy that, coupled with effective clustering of operator computation features, retains high-quality solutions while eliminating redundant design points. By accelerating the search process by 2.2× to 7.4× compared to unpruned methods, our approach demonstrates the synergistic impact of clustering and pruning in enhancing search efficiency and hardware optimization.

5.4 Collaborative Search Validation

5.4.1 Search Space. We conduct experiments on the co-exploration of neural architecture and accelerator in both the CNN and ViT spaces. In the CNN domain, we used ProxylessNAS [1] as the search space; in the ViT domain, we used Autoformer as the search space. We combined the network results from their respective papers with our hardware search engine to obtain the corresponding EDP values, which served as the baseline.

5.4.2 Collaborative Search Results. We compare our UniCoS framework with several SOTA co-exploration methods. For the CNN search space, comparisons are made with DANCE [5] and DIAN [29]. As shown in Table 5, the network identified by UniCoS achieves superior accuracy under similar search space and hardware resource constraints. When comparing DIAN and DANCE, UniCoS achieves substantially higher gains with 1.76% and 3.81% improvements in accuracy on the Imagenet-1k dataset, respectively, while reducing EDP at significantly higher rates (3.54× and 12.96×). In terms of search efficiency, UniCoS offers a 48× speedup over DIAN, reducing search time from 144 hours to just 3 GPU hours, and requires no pretraining.

For ViT search spaces, UniCoS is evaluated against AutoFormer, as no prior co-search frameworks have tackled ViT spaces. Hardware results, obtained using our dedicated evaluator, highlight that UniCoS identifies design points with superior trade-off performance.

Table 5: Comparison of existing co-exploration results on ImageNet-1k dataset.

Method	Space	Acc. (%)	EDP (nJ·ns)	Pretrain	Search Cost[a]
DANCE [5]	CNN	70.15	3.5E+14	Yes	>3h[b]
DIAN [29]	CNN	72.20	9.5E+13	Yes	144h (48×)
Ours	CNN	**73.96**	**2.7E+13**	**No**	**3h**
AutoF-T [3]	ViT	74.7	9.9E+11	Yes	420h (161×)
Ours	ViT	**75.4**	**8.5E+11**	**No**	**2.6h**
AutoF-S [3]	ViT	81.7	4.3E+12	Yes	420h (140×)
Ours	ViT	**81.5**	**3.3E+12**	**No**	**3.0h**
AutoF-B [3]	ViT	82.4	2.5E+13	Yes	420h (131×)
Ours	ViT	**82.3**	**1.6E+13**	**No**	**3.2h**

[a] *The reported GPU hours using the baseline's original papers.*
[b] *This data does not count the pre-training overhead of the accuracy predictor.*

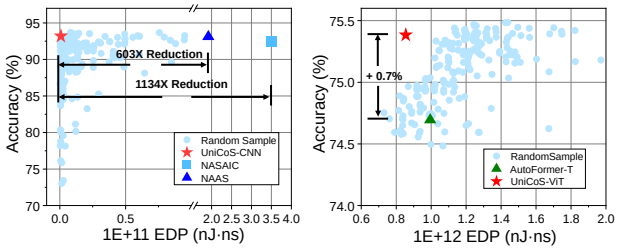


Figure 5: Accuracy and EDP trade-off. CNN design space on the CIFAR10 dataset (left) and ViT design space on the ImageNet-1k dataset.

Compared to AutoF-B, UniCoS achieves a 36% reduction in EDP and accelerates the search process by 131× (only requiring 3.2 GPU hours), with an accuracy of 82.3% on the ImageNet-1k dataset.

5.4.3 EDP VS. Accuracy. We evaluated our framework's generalization capability across diverse search spaces by sampling 200 networks from the AutoF-T space on ImageNet-1k and the CNN space on CIFAR10, comparing its performance against NAAS [19] and NASAIC [27].

As shown in Fig. 5, the design points (UniCoS-CNN and UniCoS-ViT) identified by our framework are consistently close to the *Pareto front*. In the CNN design space on CIFAR10, our approach achieves a remarkable 603× reduction in EDP compared to NAAS [19]. Likewise, within the Autoformer-Tiny search space, UniCoS demonstrates superior accuracy and EDP, even when using our hardware evaluator for Autoformer, highlighting its effectiveness in jointly optimizing hardware and network architectures.

6 CONCLUSION

This paper presents UniCoS, a unified and efficient framework for co-exploring neural architectures and hardware accelerators within the design spaces of CNNs and ViTs. UniCoS addresses existing limitations by integrating a zero-cost accuracy proxy with a novel heterogeneous dataflow design space optimized through heuristic search. Experimental results demonstrate that UniCoS significantly accelerates joint design space exploration, achieving superior solutions that balance EDP and accuracy compared to prior approaches.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China (Grants 2022YFB4501600, 2022YFB4501603), the National Natural Science Foundation of China (Grant 62172380), Jiangsu Provincial Natural Science Foundation (Grant BK20241818), Youth Innovation Promotion Association CAS (Grant Y2021121), and USTC Research Funds of the Double First-Class Initiative (Grant YD2150002011).

REFERENCES

[1] Han Cai et al. 2019. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In *ICLR*.
[2] Jingwei Cai et al. 2024. Gemini: Mapping and Architecture Co-exploration for Large-scale DNN Chiplet Accelerators. In *HPCA*.
[3] Minghao Chen et al. 2021. Autoformer: Searching transformers for visual recognition. In *ICCV*.
[4] Yu-Hsin Chen et al. 2016. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. *ACM SIGARCH computer architecture news* (2016).
[5] Kanghyun Choi et al. 2021. Dance: Differentiable accelerator/network co-exploration. In *DAC*.
[6] Xiaohan Ding et al. 2024. UniRepLKNet: A Universal Perception Large-Kernel ConvNet for Audio Video Point Cloud Time-Series and Image Recognition. In *CVPR*.
[7] Xuanyi Dong et al. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. In *ICLR*.
[8] Zidong Du et al. 2015. ShiDianNao: Shifting vision processing closer to the sensor. In *ISCA*.
[9] Martin Ester et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*.
[10] Hongxiang Fan et al. 2022. Adaptable butterfly accelerator for attention-based NNs via hardware and algorithm co-design. In *MICRO*.
[11] Weiwen Jiang et al. 2020. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2020).
[12] Sheng-Chun Kao et al. 2020. GAMMA: Automating the HW mapping of DNN models on accelerators via genetic algorithm. In *ICCAD*.
[13] Sheng-Chun Kao et al. 2022. Digamma: Domain-aware genetic algorithm for hw-mapping co-optimization for dnn accelerators. In *DATE*.
[14] Hyoukjun Kwon et al. 2020. Maestro: A data-centric approach to understand reuse, performance, and hardware cost of dnn mappings. *MICRO* (2020).
[15] Hyoukjun Kwon et al. 2021. Heterogeneous dataflow accelerators for multi-DNN workloads. In *HPCA*.
[16] Namhoon Lee et al. 2018. SNIP: Single-shot Network Pruning based on Connection Sensitivity. In *ICLR*.
[17] Guihong Li et al. 2023. ZiCo: Zero-shot NAS via inverse Coefficient of Variation on Gradients. In *ICLR*.
[18] Ming Lin et al. 2021. Zen-nas: A zero-shot nas for high-performance image recognition. In *ICCV*.
[19] Yujun Lin et al. 2021. NAAS: Neural accelerator architecture search. In *DAC*.
[20] Wenqi Lou et al. 2022. OctCNN: A High Throughput FPGA Accelerator for CNNs Using Octave Convolution Algorithm. *IEEE Trans. Comput.* (2022).
[21] Wenqi Lou et al. 2024. Unleashing Network/Accelerator Co-Exploration Potential on FPGAs: A Deeper Joint Search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2024).
[22] Liqiang Lu et al. 2021. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*.
[23] Hugo Touvron et al. 2021. Training data-efficient image transformers & distillation through attention. In *ICML*.
[24] Ao Wang et al. 2024. Repvit: Revisiting mobile cnn from vit perspective. In *CVPR*.
[25] Wenhai Wang et al. 2021. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*.
[26] Bichen Wu et al. 2019. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *CVPR*.
[27] Lei Yang et al. 2020. Co-exploration of neural architectures and heterogeneous asic accelerator designs targeting multiple tasks. In *DAC*.
[28] Shulin Zeng et al. 2024. Flightllm: Efficient large language model inference with a complete mapping flow on fpgas. In *FPGA*.
[29] Yongan Zhang et al. 2021. DIAN: Differentiable accelerator-network co-search towards maximal dnn efficiency. In *ISLPED*.
[30] Zhihao Zhang et al. 2022. GradSign: Model Performance Inference with Theoretical Insights. In *ICLR*.
[31] Qinqin Zhou et al. 2022. Training-free transformer architecture search. In *CVPR*.

Logic Restructuring with Preserved Logic Blocks

Siang-Yun Lee
Cadence Design Systems,
Munich, Germany

Heinz Riener
Cadence Design Systems,
Munich, Germany

Sascha Richter
Cadence Design Systems,
Munich, Germany

Ankush Sood
Cadence Design Systems,
San Jose, CA, USA

***Abstract*—During technology mapping, complex cells such as adders and multiplexers are often available in the standard cell library, which helps improve the final PPA results. However, technology-independent optimization tends to over-optimize towards the cost metrics measurable with technology-independent representations (e.g., AIG size) by decomposing blocks of logic that could have been mapped into complex cells. Besides, it is also of practical interest to model and preserve some special logic components, such as the enable logic of flops, during optimization. This paper studies "boxing" logic blocks before technology-independent optimization and preserving them during optimization. A wire-based resubstitution is proposed to optimize boxed networks. Experiments with academic and industrial benchmarks show that transparent-boxing, i.e., preserving boxes while utilizing the information on their logic, achieves better results than black-boxing, i.e., completely ignoring the logic in boxes. Specifically, in the technology-independent evaluation, transparent-boxing reduces the mapped network size by 4% more than black-boxing; in the full-flow evaluation, transparent-boxing achieves 1.8% more improvement on timing and similar improvements in area, power, and wire length compared to black-boxing.**

***Index Terms*—Logic synthesis, combinational circuits, Boolean optimization**

I. Introduction

Due to scalability concerns, modern logic synthesis flows are typically separated into technology-independent optimization, which focuses on removing logical redundancies using a homogeneous network data structure like the *AND-Inverter Graphs* (AIGs) [1], and technology mapping [2], which converts the optimized *subject graphs* (e.g. AIGs) into mapped netlists using a technology-specific standard cell library. In technology mapping, the subject graph is *covered* by blocks of logic available in the library by structural and logical analysis. When multiple possibilities exist, choices are made to optimize towards cost metrics measurable using information provided in the standard cell library, such as cell area, power, and delay. In contrast, in technology-independent optimization, choices are made to minimize rather simple and approximated cost metrics, such as the number of AIG nodes and the AIG depth.

While the contents of different standard cell libraries differ, some special, commonly used cells are often available, such as half adders and full adders. A *half adder* (HA) is a two-input, two-output cell computing the following functions:

$$\mathrm{HA}_c(x, y) = x \wedge y$$
$$\mathrm{HA}_s(x, y) = x \oplus y.$$

A *full adder* (FA) is a three-input, two-output cell computing the following functions:

$$\mathrm{FA}_c(x, y, z) = (x \wedge y) \vee (y \wedge z) \vee (x \wedge z)$$
$$\mathrm{FA}_s(x, y, z) = x \oplus y \oplus z.$$

Empirical observation has been found that correctly utilizing adder cells, instead of composing the same functions using elementary logic gates, may improve the *power, performance, and area* (PPA) results of the final chip [3]. As another example, modeling the sequential enable and set/reset logic of flip-flops as complex flops enhances the synthesis quality, e.g., demonstrated in [4] for an FPGA synthesis flow. A complex flop composed of a simple flop (with inputs D, *clk*, and output Q) and one or two MUXes connected to the D input of the simple flop. To model the sequential enable logic, the MUX computes

$$D = (\mathit{en} \wedge d) \vee (\neg \mathit{en} \wedge Q),$$

where *en* is the sequential enable and d is the data. To model the sequential set/reset logic, the MUX computes

$$D = (\mathit{rst} \wedge \mathit{init}) \vee (\neg \mathit{rst} \wedge d),$$

where *rst* is the sequential reset and *init* is the initial value. If both the enable and the set/reset logic are to be modeled, the two MUXes are concatenated. To not destroy the complex flop, one must guarantee that the MUXes are preserved during logic optimization. Besides these, some other cells or logical components are also of practical interest to be modeled and preserved in logic synthesis, such as clock gating logic [5], pre-mapped user hierarchical instances, and components with regular structures (e.g. XORs and MUXes) [6].

It is thus desired to preserve some special logic blocks exemplified above during optimization. However, without intentionally protecting these special logic blocks, classic technology-independent optimization algorithms have a high chance of destroying them structurally, making it difficult to restore or re-discover them later in technology mapping. This problem is tackled in some open-source and industrial tools by *black-boxing* special logic blocks. When a block is black-boxed, AIG nodes representing its function are removed, creating an empty hole in the network whose interfaces are connected to additional virtual PIs and POs to keep the graph acyclic and non-dangling. However, to correctly identify the critical path, one needs to annotate the virtual PIs with additional level information and update this information if the logic changes during optimization. While the black-boxing strategy successfully protects these blocks, we argue that a *transparent-boxing* strategy that does not neglect the logical information within boxes would be more advantageous. In a transparent-boxing strategy, boxed nodes are kept in the network and are utilized to, for example, compute don't care conditions for other nodes. In the context of critical path identification, it is easier to update the level information at the transparent box outputs when their input levels are updated.

In order to take advantage of the transparent boxes, we need to develop new algorithms or adjust existing logic synthesis algorithms. In this context, we present a new optimization algorithm, called wire-based resubstitution that, in contrast to existing node resubstitution techniques, focuses on input pins of transparent boxes. To leverage the additional information provided by the transparent boxes, observability don't cares (ODCs) play an important role in this algorithm. Since nodes in the transparent boxes have to be preserved, optimization is only possible at the input pins, whereas the boxed nodes contribute to the ODC computation. As the node connected to an input pin may have other fanouts outside of the box, the proposed wire-based

resubstitution is necessary to optimize the box where existing node-based methods cannot.

Our experiment on technology-independent evaluation with the EPFL benchmarks shows that transparent boxing achieves the most reduction in the number of free ANDs as well as in the estimated mapped network size, compared to black boxing and no boxing. Experiments on industrial benchmarks using integration with an industrial synthesis tool show improvements of 9.8% in total negative slack (TNS), 0.5% in area, 1.8% in power, and 2.7% in wire length with black-boxing. On top of such, with the addition of the proposed wire-based resubstitution and transparent boxing, 1.8% more improvement in TNS and similar improvements in area, power, and wire length are observed.

In summary, the contributions of this paper are three-fold:

1) Detailed explanations on how boxing techniques are efficiently implemented, especially for transparent-boxing.
2) A novel wire-based resubstitution algorithm for optimizing transparent-boxed networks.
3) Experimental evaluation on both academic and industrial benchmarks showing PPA improvements given by the proposed methods, demonstrating the importance of boxing and preserving special logic blocks.

II. Preliminaries

A. Logic Networks

Logic networks are technology-independent representations used in logic synthesis. A *network* is a *directed acyclic graph* (DAG) where nodes model simple logic gates and edges model wires potentially with integrated inverters (fanin negations). The *primary inputs* (PIs) and the *primary outputs* (POs) are the interfaces between a network and the outside environment. A representative example of logic networks is the *AND-Inverter Graph* (AIG) where each node is a two-input AND gate. In the figures in this paper, wires with inverters are drawn as dashed edges.

In a typical AIG data structure, a list of nodes is stored, where each node stores pointers to its two fanin nodes, together with fanin negation tags. PIs are represented by special nodes, identified by having identical fanins. POs are stored separately as a list of pointers to nodes with negation tags. During AIG construction and throughout optimization, trivial cases are checked and simplified such that the following conditions should never happen: (1) A node has a constant (0 or 1) fanin. (2) A non-PI node's two fanins point to the same node (with or without the same fanin negation tag). Whenever these cases appear, the node should not be created and should be replaced with either a constant or its other fanin. Such replacement and simplification may propagate towards the transitive fanouts.

Moreover, a technique called *structural hashing* is often adopted to avoid redundancies. A hash table is used to quickly look up a node using its fanins (negation tag considered) and to ensure that there is not a second node with exactly the same fanins, i.e., a node is unique to its fanins. During network modification (e.g., node substitution), some nodes' fanins may change, and they should be re-hashed, potentially becoming identical to another existing node and thus merged. Such changes may propagate towards the transitive fanouts.

B. Don't-Cares

Don't cares are flexibilities in a network and are often computed and leveraged in logic optimization to enhance optimization power. They indicate where the local function of a node in the network is allowed to be modified without affecting the global functionality the network computes. Two types of don't-cares are typically considered.

First, for a set of internal nodes, there might be some value combinations that never appear at these nodes. For example, an AND gate g_1 and an OR gate g_2 sharing the same fanins can never have $g_1 = 1$ and $g_2 = 0$ at the same time. This combination is a *satisfiability don't-care* (SDC) for a common transitive fanout of g_1 and g_2.

Second, a value assignment $\vec{x} \in \mathbb{B}^n$ to the PIs is said to be *unobservable* with respect to a node n if none of the POs changes its value when n is replaced by its negation $\overline{n}$. $\vec{x}$ is an *observability don't-care* (ODC) of n because the function of n under $\vec{x}$ does not matter.

Typically, SDCs are computed at the transitive fanins of the node under optimization, whereas ODCs are computed using the transitive fanouts of the node.

C. Simulation-Guided Resubstitution

Boolean resubstitution is a technology-independent logic optimization algorithm that attempts to *resynthesize* the *maximum fanout-free cone* of a *root* node using existing *divisor* nodes having common support as the root. In cut-based resubstitution, divisors are collected within a smaller window supported by a cut of size 8-10 (typically no larger than 16), and the resynthesis is performed with (complete) truth tables exhaustively simulated from the cut. In contrast, in simulation-guided resubstitution [7], partial truth tables computed from the primary inputs using a non-exhaustive set of simulation patterns are used to approximate nodes' global functions. One of the advantages of simulation-guided resubstitution is that global SDCs are considered implicitly with global simulation without needing extra effort to compute them. It is also possible to leverage ODCs in simulation guided resubstitution. When doing so, ODCs are often computed only up to 5 levels of the transitive fanout cone because involving too large fanout cones would result in a high runtime.

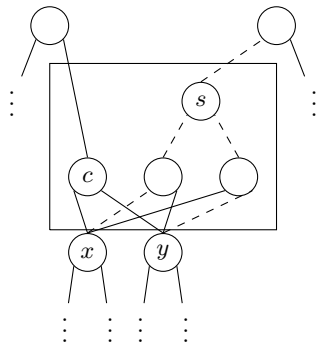
D. Black-box and Transparent-box Testing

In the field of software testing, the term *black-box testing* refers to a testing strategy that examines the functional correctness of software according to its input-output relationship without knowledge of its internal workings. In contrast, *transparent-box testing* (or *white-box testing*, *clear-box testing*, *glass-box testing*) tests the internal structures of software.

In the context of logic synthesis, we use the term *boxing* to mention the extraction of blocks of logic in a technology-independent representation that can be mapped into (complex, potentially multi-output) cells. We borrow the term *black-boxing* to mention the strategy where the boxed logic is taken away with a "hole" of some dangling wires remaining in the network. In contrast, we use the term *transparent-boxing* for the strategy where the logic function and the internal implementation of the boxes are known and can be utilized during optimization.

III. Boxing Technology-Independent Logic Networks

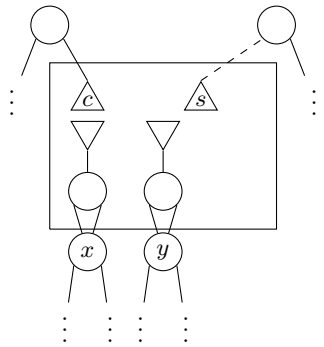
To preserve specific logic blocks in the network during logic optimization, we propose to create boxes to protect them. A *box* in a network is a connected subset of nodes. Two boxes in the same network shall not overlap. A box can be identified by a set of *box inputs*, which are nodes outside the box having a fanout inside the box, and a set of *box outputs*, which are nodes inside the box with at least one fanout outside the box. Optionally, a label referring to the logic function(s) computed by the box outputs in terms of the



Box inputs $= \{x, y\}$

Box outputs $= \{c, \neg s\}$

Fig. 1: A boxed half adder in a network.



Box inputs $= \{x, y\}$

Box outputs $= \{c, \neg s\}$

Fig. 2: Data structure implementation of a black box in a network.

box inputs may be associated with the box. For example, Figure 1 shows a half adder box in a network. In this section, we discuss how black and transparent boxes can be implemented in an AIG-like data structure.

A. Black-Boxing

In a black-boxing strategy, nodes within the box are taken away from the network, and the box's inputs and outputs become dangling. To keep the network a valid DAG, additional virtual PIs are added and used to replace the box outputs, and additional virtual POs are added and connected to the box inputs. Practically, we insert additional buffers (nodes with identical fanins) between the box inputs and the added virtual POs to easily identify these POs as box inputs, in case the box inputs are also connected to a real PO of the network. Figure 2 illustrates such a black-boxing strategy.

Classic logic optimization algorithms can be applied directly on a black-boxed network without incurring algorithmic or correctness errors (e.g., the optimized network becomes cyclic or functionally non-equivalent to the original network). This is because (1) the box outputs are replaced with independent PIs, so no assumptions on their values in relation to other nodes in the network would be made, and (2) the box inputs are connected to additional POs, so their functions would be kept the same under all circumstances.

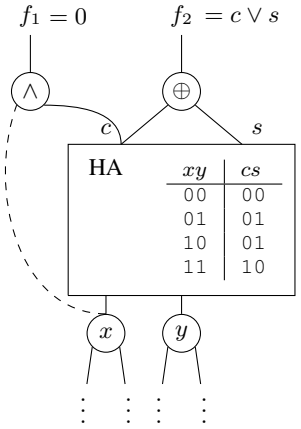


Fig. 3: Example of optimization with boxes.

B. Transparent-Boxing

A transparent-boxing strategy keeps track of, in addition to the interfacing box inputs and outputs, the logic function computed by the box and the internal implementation of the box. There are several ways of storing the box function. For example, storing truth tables, using binary decision diagrams (BDDs), or linking to separated sub-networks. To minimize the impact on the network data structure and to facilitate the adaptation of existing algorithms to boxed networks, we propose to simply keep the boxed nodes in the network but mark them as *don't touch*es. This is an extension to the standard AIG conventions introduced in Section II-A, where some conditions that should not appear in standard AIGs become possible in the transparent boxes.

A don't touch is an additional flag stored for each node in the network. Network modification methods, such as node substitution, are modified to avoid changing nodes marked as don't touch. Special attention should be paid to handling trivial cases, structural hashing, and their propagation towards transitive fanouts. Specifically, (1) constants and repeated fanins can be allowed at box inputs; decisions on whether to simplify the box can be made in a post-optimization process. (2) If a node in a box is (or becomes) structurally identical to a node outside the box, both of them should be kept.

The advantage of this transparent-boxing strategy is that the network data structure remains the same as its non-boxed counterpart, so most non-modifying computations, such as simulation, cut enumeration, and structural analysis, can be done in the same way. An illustration of a white-boxed data structure would look the same as in Figure 1, except that nodes in the box are tagged as don't touches.

IV. Logic Optimization with Boxes

Using a black-boxing strategy, special logic blocks are easily preserved during technology-independent logic optimization. This strategy has the advantage of being straightforward and guarantees to work with any optimization algorithm without adaptation. However, the logical relations between the box inputs and outputs are completely neglected during optimization, which may lead to oversight of potential optimization opportunities. Moreover, by connecting box inputs and outputs to virtual POs and PIs, the overall structure (or the "shape") of the network is changed. For example, a box may be originally "deep" in the network, i.e., having a large logic depth from PIs, but the depth of the box outputs will become very small after black-boxing because they are connected to virtual PIs which are

usually assigned to level 0. While this distortion can be mitigated by assigning non-zero input levels to these PIs according to their real arrival times, this information may still become inaccurate during optimization. When a black box input's level changes due to an optimization made, this change cannot be propagated immediately and correctly to the black box's outputs. As another example, due to the "holes" and disconnections caused by black-boxing, some cuts may disappear. These structural changes may have an impact on the quality of the results of some optimization algorithms.

To avoid the disadvantages of black boxing, we propose to keep the structural and logical information with transparent boxing. Using the proposed transparent-boxing strategy, most optimization algorithms should perform the same as without boxing when optimizing non-boxed nodes. However, some optimization algorithms may benefit more from the information provided by the transparent boxes because they leverage don't-care computations in larger windows beyond the MFFC of the node being optimized. One prominent example of such algorithms is simulation-guided resubstitution [7], which relies on whole-circuit simulation to approximate nodes' global functions, where global satisfiability don't cares are implicitly considered. Another example is the new rewriting framework capable of using don't cares [8].

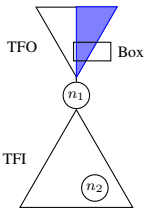
Figure 3 shows two examples of optimizations that are only discoverable with transparent boxing. First, the output function f_1 depends on a box output and a box input, which are related. With a black box in between, optimization algorithms cannot recognize that f_1 always evaluates to constant 0. In contrast, with a transparent box, an algorithm capable of computing don't cares would realize the satisfiability don't care condition $x = 0, c = 1$ and use this fact to substitute the AND gate with constant 0. Second, the output function f_2 depends on two outputs of the same box, which are also related. Again, with a black box model, the two box outputs are unrelated virtual PIs and cannot be optimized. However, with a transparent box, the don't care condition $c = 1, s = 1$ can be computed and used to optimize the XOR gate to an OR gate, which may have a lower cost.

A. Wire-Based Resubstitution

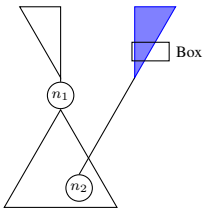
In addition to the existing algorithms suitable for optimizing transparent-boxed networks, in this section, we further devise an algorithm tailored for this purpose. The proposed algorithm aims to complement the other optimizations when being integrated together in a flow by targeting especially on rewiring the input pins of boxes. To better utilize the information provided by the transparent boxes, and also because optimizations happen at box inputs, ODCs are computed and play an important role in this algorithm.

The basic idea is similar to simulation-guided resubstitution, except that instead of substituting a node with another, we try to substitute an input pin of a box with another node in its transitive fanin cone. In other words, instead of requiring functional equivalence of two nodes (subject to don't cares), we seek unchanged functionality at the outputs in the partial transitive fanout cone of a node that passes through the considered box.

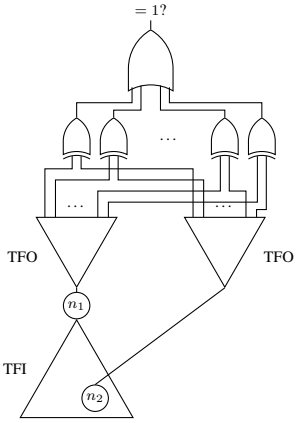
The difference between classic resubstitution and wire-based resubstitution is illustrated in Figure 4. Figure 4a shows the original network with a transparent box at the fanout of node n_1. Part of the transitive fanout cone (TFO) of n_1 passes through the box (blue), whereas the other part does not. A node n_2 in the transitive fanin cone (TFI) is being considered as a divisor to substitute n_1 with. In classic resubstitution considering ODCs, the algorithm checks if substituting n_1 with n_2 would result in any difference in the TFO with a miter of the original TFO and a duplicated TFO connected



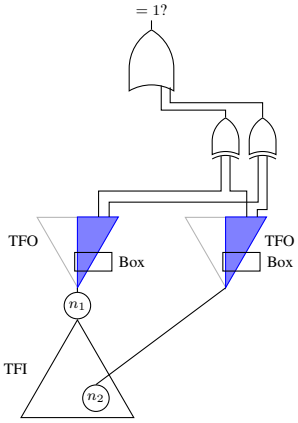
(a) Original network with a transparent box.



(b) Optimized by wire-based resubstituion.



(c) Classic resubstitution considering ODCs.



(d) Wire-based resubstitution.

Fig. 4: Illustrations of ODC-based optimization.

to n_2 (Figure 4c). In contrast, wire-based resubstitution focuses on the partial TFO passing through the box (Figure 4d). Since a smaller part of the network is being affected, wire-based resubstitution has a higher chance to find valid optimizations.

As there may be other fanouts that still require n_1 after substitution, reduction in AIG nodes is not guaranteed. However, we only consider substituting with a node in the TFI, so the move cannot make the AIG size or depth worse. The metrics either stay the same or get better. In summary, the proposed wire-based resubstitution is a relatively light-weight optimization that does not consume much runtime. Nevertheless, it finds some key optimizations and complements other higher-effort optimizations that cannot take full advantage of the transparent-box information.

V. Experimental Results

A. Technology-Independent Evaluation

In this experiment, we implement the boxing techniques discussed in this paper in the C++ logic synthesis library `mockturtle`[1] [9], [10] and use the EPFL benchmark suite [11] for the technology-independent evaluation of the two boxing techniques discussed in this paper. Since these benchmarks are given in form of AIGs and do not contain high-level (RTL) information that may give rise to logical components to be boxed, we perform adder identification to find half adders and full adders. The benchmarks are pre-processed with the balancing command `&b` in ABC [12] as a heuristic to increase the number of recognizable adders. Before optimization, half adders and full adders are identified and boxed by enumerating and analyzing the functions of 3-cuts [13].

[1] Available: https://github.com/lsils/mockturtle

TABLE I: Evaluation of different boxing techniques on EPFL benchmarks.

	Before optimization		Without boxing		Black-boxing			Transparent-boxing		
Bench.	A	A+B	A+B	Time	A	A+B	Time	A	A+B	Time
adder	0	128	128	0.0	0	128	0.0	0	128	0.0
bar	3214	3214	3155	0.0	3155	3155	0.0	3155	3155	0.0
div	13784	17695	17615	2.7	13580	17436	0.4	13575	17518	1.8
hyp	113550	133666	114507	98.1	109682	122860	2.9	108642	113954	63.6
log2	20527	21981	21483	16.8	20451	21915	0.4	19822	21300	2.5
max	2865	2865	2862	0.0	2863	2863	0.0	2863	2863	0.0
multiplier	15384	16895	16826	2.1	15381	16890	0.5	15315	16824	0.2
sin	3554	3820	3743	1.1	3520	3787	0.1	3447	3728	1.5
sqrt	14734	16553	16467	5.2	14615	16438	0.1	14595	14602	2.7
square	7949	9327	8175	0.7	7238	8388	0.5	7067	8169	0.6
arbiter	11839	11839	11839	0.3	11603	11603	0.4	11603	11603	0.4
cavlc	672	679	616	0.2	633	643	0.3	598	614	0.2
ctrl	165	166	93	0.0	112	114	0.0	100	103	0.0
dec	304	304	304	0.0	304	304	0.0	304	304	0.0
i2c	1187	1191	1084	0.0	1105	1110	0.1	1089	1094	0.1
int2float	226	227	212	0.1	209	210	0.2	212	214	0.2
mem_ctrl	46543	46584	38083	2.7	40113	40150	4.3	38298	38353	4.2
priority	843	843	518	0.0	560	560	0.0	560	560	0.0
router	133	171	133	0.0	133	171	0.0	119	164	0.0
voter	1231	2558	2037	1.0	1231	2544	0.0	528	1493	0.6
Total	258704	290706	259880 (-10.6%)	131.0	246488 (-4.7%)	271269 (-6.7%)	10.2	241892 (-6.5%)	256743 (-11.7%)	78.7

TABLE II: Evaluation of different boxing techniques on industrial benchmarks.

	Without boxing	Black-boxing		Transparent-boxing	
		Avg.	W/L	Avg.	W/L
TNS	±0% (baseline)	-9.78%	7/6	-11.55%	8/5
WNS	±0% (baseline)	+3.57%	7/4	+1.19%	8/5
Area	±0% (baseline)	-0.48%	10/2	-0.52%	10/3
Power	±0% (baseline)	-1.75%	10/3	-1.27%	10/3
WL	±0% (baseline)	-2.73%	13/0	-2.36%	12/1

In Table I, statistics are reported for benchmarks before optimization, optimized normally without boxing, optimized with black-boxing, and optimized with transparent-boxing. In all cases, the optimization algorithm is (classic) simulation-guided resubstitution with a cut size (for divisor collection) of 8 and a maximum dependency circuit size of 20. Columns "A" show the number of free AIG nodes (nodes not in any boxes) and columns "A+B" show the number of free AIG nodes plus the number of boxes, which resembles the size of the mapped network using a standard cell library consisting of only AND2, half adder, and full adder. Columns "Time" show the optimization time in seconds.

It is observed that optimization with transparent-boxing achieves the most reduction in both the number of free ANDs and the mapped network size. The runtime for black-boxing is the fastest, because many nodes are not seeable by the optimization algorithm, thus reducing computational effort for simulating, verifying, and substituting. However, for the same reason, the optimization quality is the worst.

B. Full-Flow Synthesis Results

The motivation to preserve special logic blocks is that their existence improves physical metrics that are not measurable during technology-independent optimization. Results after a full-flow synthesis are necessary to demonstrate this effect but are not available with open-source academic tools. Thus, in the second experiment, we integrate the discussed boxing techniques, as well as the proposed wire-based resubstitution for transparent-boxed networks, into an industrial flow. Another advantage of experimenting with an industrial tool is that, by starting from RTL netlists, we have access to the practical sources of logic blocks to be boxed, such as complex flops and MUXes.

In Table II, averaged and normalized results on 13 industrial benchmarks synthesized with an industrial flow are presented. The baseline is a vanilla flow without attempting to preserve special logic blocks. Physical metrics on timing (total negative slack, TNS, and worst negative slack, WNS), area, power, and routing (wire length, WL) are compared, where the average delta (Avg.) and the numbers of wins and losses (W/L) among the 13 benchmarks are listed. All of these metrics are better with a smaller value (i.e., a negative delta is good). In the column "Transparent-boxing", the wire-based resubstitution algorithm proposed in Section IV-A is additionally applied after black-boxed optimizations, with black boxes being transformed into transparent boxes.

We observe that most metrics improve over the baseline with boxing techniques, proving the claim that preserving specific logic components is indeed beneficial for physical metrics. Although WNS degrades on average, it is actually improved in more benchmarks than where it degrades. Comparing transparent-boxing against black-boxing, the most prominent improvement lies in the timing metrics. This is reasonable, as the wire-based resubstitution tries to rewire a box input pin with a transitive fanin node, which may have a better arrival time than the original node. The area metric improves in most benchmarks, even though only by about 0.5% on average. This is likely because, as discussed before, the wire-based resubstitution does not ensure a reduction on AIG size. It simply accepts any possible substitution, and zero-gain moves are allowed. Even though the metrics on power and wire length do not improve as much with transparent-boxing as with black-boxing, they still improve by 1.27% and 2.36%, respectively, over the baseline, and the improvements are consistent in most benchmarks. In conclusion, migrating from black-

boxing to transparent-boxing allows more aggressive optimizations being done to further improve on timing and area, while compromising marginally on the other physical metrics as a drawback.

As the industrial synthesis flow is complicated and includes various heuristics, the results could be affected by different factors. The presented experiment does not involve tuning on hyperparameters or improvements to the heuristics. We simply added the boxing strategies to an existing flow. Thus, it is very promising to see improvements in most metrics on most benchmarks, and without outliers.

VI. Conclusions

In this paper, we discuss methods to "box" special logic blocks in technology-independent optimization to preserve them. We argue that don't-care-based optimization algorithms would take advantage of the information of the boxed logic, thus we propose the transparent-boxing technique to enhance optimization quality with boxes. Especially for such, we propose a novel wire-based resubstitution algorithm tailored for finding hidden optimizations in transparent-boxed networks that are otherwise hard to find by other existing optimizations. We also discuss potential problems and details to pay attention to in the implementation of the data structure of black and transparent boxes. Finally, our experimental evaluations show that preserving specific logic blocks—either by black-boxing or transparent-boxing—indeed improves on the physical metrics in an industrial synthesis flow. Moreover, compared to black-boxing, transparent-boxing achieves better optimization quality thanks to the additional information retained by the transparent boxes.

Simple technology-independent representations like AIGs have been shown to be an efficient, scalable and powerful data structure for logic optimization in academic tools. [14] However, in the industrial context, with consideration of physical metrics and information on complex logic components, our experimental evaluation shows that vanilla AIGs are perhaps too naive. Boxing techniques play an essential role in many industrial synthesis tools. The main reason for this is the well-known gap between technology-independent representations and the physical implementations. Various approaches have been explored to close this gap, such as performing early mapping rounds in generic synthesis to provide more accurate estimations on the real costs. These approaches are often complicated and time-consuming. Thus, it is encouraging to see that such a simple technique as transparent-boxing can already help in this context.

References

[1] A. Mishchenko and R. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

[2] K. Keutzer, "DAGON: technology binding and local optimization by DAG matching," in *Proceedings of the 24th ACM/IEEE Design Automation Conference. Miami Beach, FL, USA, June 28 - July 1, 1987*, 1987, pp. 341–347.

[3] A. T. Calvino and G. D. Micheli, "Technology mapping using multi-output library cells," in *IEEE/ACM International Conference on Computer Aided Design, ICCAD 2023, San Francisco, CA, USA, October 28 - Nov. 2, 2023*. IEEE, 2023, pp. 1–9.

[4] B. L. C. Barzen, A. Reais-Parsi, E. Hung, M. Kang, A. Mishchenko, J. W. Greene, and J. Wawrzynek, "Narrowing the synthesis gap: Academic FPGA synthesis is catching up with the industry," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2023, Antwerp, Belgium, April 17-19, 2023*. IEEE, 2023, pp. 1–6.

[5] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 3, pp. 415–420, 2000.

[6] T. Kutzschebauch, "Efficient logic optimization using regularity extraction," in *Proceedings of the IEEE International Conference On Computer Design: VLSI In Computers & Processors, ICCD '00, Austin, Texas, USA, September 17-20, 2000*. IEEE Computer Society, 2000, pp. 487–493.

[7] S.-Y. Lee, H. Riener, A. Mishchenko, R. K. Brayton, and G. De Micheli, "A simulation-guided paradigm for logic synthesis and verification," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2573–2586, 2022.

[8] A. Tempia Calvino and G. De Micheli, "Scalable logic rewriting using don't cares," in *Proceedings of DATE*, 2024.

[9] H. Riener, E. Testa, W. Haaswijk, A. Mishchenko, L. G. Amarù, G. D. Micheli, and M. Soeken, "Scalable generic logic synthesis: One approach to rule them all," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 2019, p. 70.

[10] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, D. S. Marakkalage, and G. De Micheli, "The EPFL logic synthesis libraries," 2022. [Online]. Available: http://arxiv.org/abs/1805.05121

[11] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The EPFL combinational benchmark suite," in *Proceedings of IWLS*, 2015.

[12] R. K. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Computer Aided Verification, 22nd International Conference, CAV 2010*, 2010, pp. 24–40.

[13] A. Mishchenko, B. Sterin, and R. Brayton, "Structural reverse engineering of arithmetic circuits."

[14] A. Mishchenko and R. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

PreDAC: An Efficient Framework of Pre-Refining Enhanced Design Space Exploration for Approximate Computing

Ziying Cui*, Ke Chen*, Bi Wu, Yu Gong, Chenggang Yan, and Weiqiang Liu
Nanjing University of Aeronautics and Astronautics, Nanjing, China
{chen.ke, liuweiqiang}@nuaa.edu.com

***Abstract*—Approximate computing has emerged as a promising solution in energy-efficiency applications. Recently, attention has shifted from approximate components to Design Space Exploration (DSE) algorithms. However, traditional DSE algorithms face challenges in efficiently obtaining optimal solutions within large and complex design spaces. This paper introduces a pre-refining enhanced design space exploration framework that provides customized design space and cost-performance formula for applications. Experimental results demonstrate that integrating this pre-refining step into various DSE algorithms leads to substantial performance gains, including up to 87× speedup and a 23% improvement in hardware overhead. Moreover, the innovative cost-performance-based DSE algorithm attains a 7.7× acceleration and further optimizes hardware metrics by an additional 8.8% compared to advanced frameworks employing the same pre-refinement.**

***Index Terms*—Approximate Computing, Design Space Exploration, High-Level Synthesis**

I. Introduction

Approximate computing has emerged as an effective paradigm to mitigate hardware overheads by leveraging the inherent fault tolerance of applications in the post-Moore era. This approach significantly enhances the design methodologies for artificial intelligence (AI), the Internet of Things (IoT), and communication systems. Despite the existence of numerous approximate design methods [1]–[3], the process of crafting an approximate scheme tailored to a specific application remains a time-consuming and labor-intensive task. Currently, the derivation of such approximation schemes predominantly relies on manual expertise, with limited support from Electronic Design Automation (EDA) tools. To address these challenges, researchers have increasingly turned their attention to the development of design space exploration (DSE) algorithms. Existing DSE approaches can be broadly classified into two categories. The first employs heuristic algorithms [4]–[7], which iteratively search within the design space but often require substantial runtime. The second category adopts stepwise approximation methods, guided by metrics such as cost-performance trade-offs [8] or impact factors [9]. These methods achieve reduced synthesis and validation times but often produce suboptimal results. Consequently, the algorithms proposed in existing studies face significant limitations: excessive runtime, poor scalability, or unsatisfactory outcomes due to the vastness of the design space. Moreover, existing approximate libraries are constrained in scope and fail to comprehensively capture the full design space.

This paper proposes PreDAC, an efficient pre-refining enhanced design space exploration framework. PreDAC considers application-specific input distributions, optimization objectives, and accuracy constraints to construct a comprehensive, redundancy-free design space tailored to the application. Additionally, an optimized DSE algorithm is introduced, which employs a stepwise approach, progressing from the preliminary exact design towards the periphery of the design space—where designs with higher error and minimal hardware overhead dominate—guided by a cost-performance ratio rather than random generation. Furthermore, a novel strategy is incorporated to further minimize hardware overhead. PreDAC is evaluated across the design spaces of four widely used applications, demonstrating favorable results. The key contributions of this work are summarized as follows:

- A generic approximation library is constructed by integrating commonly used approximate multipliers.
- A novel two-stage pre-refining step is introduced, leveraging application characteristics to provide a customized design space compatible with arbitrary DSE algorithms.
- An enhanced DSE algorithm is proposed, incorporating a cost-performance formula with adjustable parameters and a fine-tuning strategy.
- PreDAC achieves an average speedup of 7.7× compared to FPAX [8] and delivers an additional 8.8% optimization over JS [9].

II. Proposed Pre-Refining Framework

The proposed PreDAC framework consists of two main steps, as illustrated in Fig. 1. The first step involves the pre-refinement of the approximate multiplier library, which serves as a preparatory phase and can be seamlessly integrated into any design space exploration algorithm. The second step entails an enhanced design space exploration algorithm, which systematically transitions from the initial fully accurate design to configurations with progressively higher error and reduced hardware overhead, guided by the cost-performance ratio.

*These authors contributed to the work eqully and should be regarded as co-first authors.

This work is supported by grants from the National Key R&D Programme of China (2022YFB4500200), National Nature Science Foundation of China (62425404, 92364201).

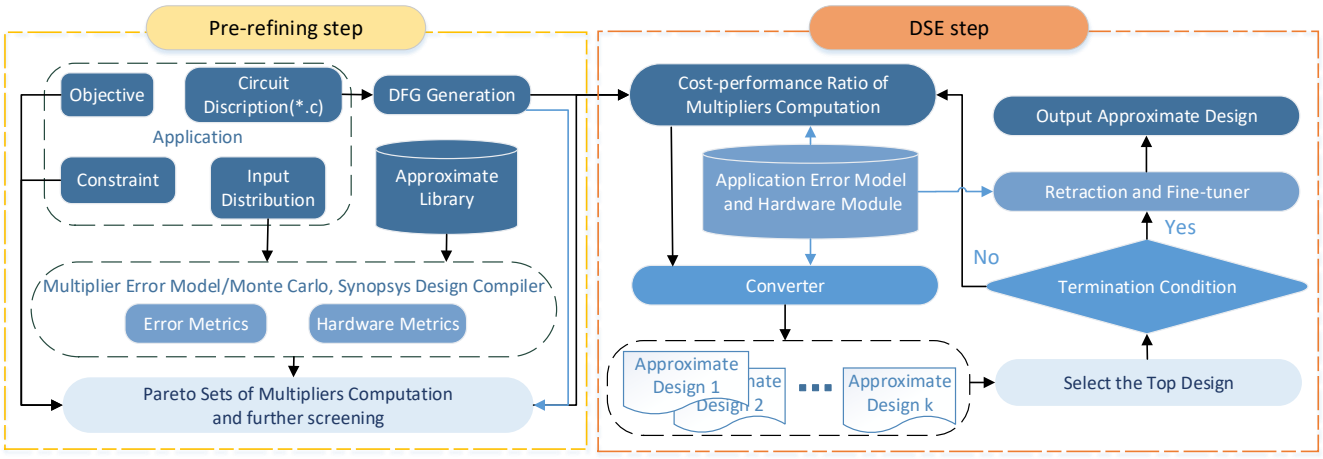


Fig. 1. The frame diagram of PreDAC.

A. Approximate Multipliers Library

Since multiplication is one of the most frequently used arithmetic operations in a wide range of algorithms, approximate multipliers are employed to construct a comprehensive approximation library capable of accommodating diverse application requirements. Approximate multipliers can be broadly classified into two categories: manually designed multipliers and automatically generated multipliers. To ensure comprehensive coverage, this paper selects representative samples from both categories. In the category of manually designed multipliers, the widely used DRUM multipliers [10] and ETM multipliers [11] are included, resulting in a total of 10 approximate multipliers, achieved by tuning configurable parameters. Additionally, a commonly employed approximate multiplier design methodology [12], which incorporates techniques such as radix-4 Booth encoding and decoding, input truncation and compensation, partial product truncation and compensation, and approximate compressors, is also considered. This methodology yields 34 approximate multipliers by varying the relevant parameters. In the category of automatically generated multipliers, 11 approximate multipliers derived using a genetic algorithm [13] are selected for inclusion in the library. In total, the approximation library comprises 55 approximate multipliers, numbered sequentially from 1 to 55, providing a robust and diverse foundation for exploring and optimizing hardware designs across various applications.

B. Pre-Refining of the Approximate Library

The size of the design space grows exponentially with the number of multipliers in the library. For example, in a 13-tap FIR filter, if the number of available approximate multipliers is n, the design space expands to 13^{n+1}. By reducing n from 8 to 7, approximately 10 billion design configurations can be eliminated, highlighting the necessity of refining the approximate library prior to executing the DSE algorithm. To address this, the 55 approximate multipliers in the library undergo a pre-refinement step, which identifies optimal approximate

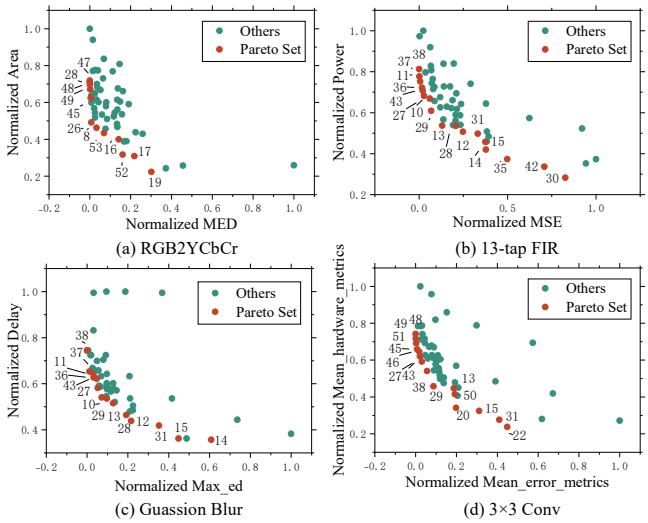


Fig. 2. Pareto set of multipliers for four distinct applications along with corresponding hardware and accuracy metrics.

configurations tailored to specific applications. This process ensures that the design space remains both manageable and relevant, significantly enhancing the efficiency of subsequent design space exploration and optimization.

In addition, different applications exhibit unique input distributions, and the various error metrics associated with multipliers have distinct strengths and weaknesses. As a result, the error characteristics of a given multiplier can vary significantly depending on the input distribution and the selected error metric. Furthermore, the behavior of a multiplier may change dramatically depending on the ultimate optimization objective of the application, such as minimizing area, power consumption, or delay. To illustrate these considerations, four distinct applications, RGB2YCbCr, 13-tap FIR filter, Gaussian Blur filter, and 3x3 convolution were selected, each paired with corresponding hardware and accuracy metrics. The multiplier

error model described in [12] was employed to compute error metrics, as it offers a more time-efficient alternative to the Monte Carlo method. Hardware metrics were synthesized using Synopsys Design Compiler (DC) to ensure accuracy. The resulting Pareto-optimal sets of multipliers for these applications are highlighted in red in Fig. 2. The corresponding hardware metrics are area, power consumption, latency, and a comprehensive hardware metric. The accuracy metrics are MED (Mean Error Distance), MSE (Mean Square Error Distance), MAX_ED (Maximum Error Distance), and a comprehensive accuracy metric. As shown in Fig. 2, the Pareto sets of multipliers differ significantly across applications, emphasizing the need for application-specific refinements. Since even a single reduction in the number of approximate multipliers can drastically impact the design space, a second screening of the approximate library is necessary. This screening considers the error constraints specific to each application, complementing the initial pre-refinement step.

The proposed pre-refining step is structured into two distinct parts. The first part involves selecting the Pareto-optimal set of multipliers from the library, guided by the application's specific input distribution. The second part focuses on eliminating multipliers that exhibit excessive errors, ensuring compliance with the given error constraints. The detailed procedure is outlined in Algorithm 1.

Algorithm 1 Pre-refining

Input: DFG (G); Input Distribution (D); 55 Approximate Multipliers (M_o); Objective (O); Error Constraints (EC);
Output: Multipliers for DSE (M_l); Abandon for Level (AL);

1: Initialize: $HW_{metric} = [], Err_{metric} = [], AL = []$;
2: **for** $i \leftarrow 1$ to 55 **do**
3: $HW = DC(D, O, M_o[i])$;
4: $HW_{metric}.append(HW)$;
5: $Err = Multiplier_Error_model(D, O, M_o[i])$;
6: $Err_{metric}.append(Err)$;
7: **end for**
8: $M_l = Pareto_set(HW_{metric}, Err_{metric})$;
9: $(N_c, Mul_c) = Classifier(G)$;
10: **for** $i \leftarrow 1$ to N_c **do**
11: $Mul = Random_select(Mul_c[i])$;
12: **for** $j \leftarrow length(Mul_{pareto})$ to 1 **do**
13: $abandon = []$;
14: $Mul = Mul_{pareto}[j]$;
15: $Err = Application_Error_model(D, G, O, Mul)$;
16: **if** $Err \leq EC$ **then**
17: break
18: **else**
19: $abandon.append(j)$;
20: **end if**
21: **end for**
22: $AL.append(abandon)$;
23: **end for**
24: **return** M_l, AL

The first section of Algorithm 1 (steps 2–8) utilizes input distributions, objective metrics, and the 55 approximate multipliers to compute the Pareto-optimal set, following the computation of the corresponding hardware and error metrics. The input dataset is transformed into a probability density function, which is integrated into the error model of the approximate multipliers. This enables the cumulative error generated by each input case to be weighted by its probability, effectively incorporating the input distribution into the analysis. The resulting Pareto sets of multipliers are arranged in ascending order of error magnitude, facilitating subsequent operations.

The second component of Algorithm 1 (steps 9–23) is then executed. The input to the framework is a high-level language model (*.c) of the application, which is first converted into a Data Flow Graph (DFG) using GAUT [14]. Candidate multipliers are classified by their levels in the DFG, determined by the number of operations performed from the input to the multiplier, using the $Classifier$ function in Algorithm 1. This approach assumes that multipliers at the same level in DFG have comparable impacts on the final result and, thus, should be categorized together. Next, for each class, a candidate multiplier is randomly selected for replacement with the multiplier exhibiting the largest error in the library (steps 11 and 14). The application error model [15] is then used to compute the final result and verify whether it satisfies the prescribed error constraints. If the error exceeds the threshold, the approximate multiplier is deemed unsuitable, and the replacement strategy is excluded from DSE algorithms for that category (step 19). In the subsequent step, the multiplier with the second-highest error is selected and replaced, and the process is repeated until the error meets the specified limits or all library multipliers have been evaluated. Otherwise, the operation is repeated for the subsequent class of multipliers. This process is iteratively applied to all classes of multipliers, ensuring that only one candidate multiplier is replaced per iteration, while others remain in exact form.

The multipliers obtained from the pre-refining stage are sorted and numbered based on their error magnitudes in ascending order. These refined multipliers, along with the abandon set, constitute the approximation library for the DSE algorithm. This pre-refinement step significantly accelerates the exploration process by reducing the design space and simultaneously delivers superior results compared to employing the full unrefined multiplier library.

C. Cost-Performance ratio Based Design Space Exploration Method

In this paper, the proposed DSE algorithm transitions systematically from an initial error-free design to designs with progressively higher error and reduced hardware overhead, guided by the cost-performance ratio. The algorithm consists of four principal stages. The initial step is Cost-Performance Calculation, which compute the cost-performance ratio for the multiplier replacement schemes, serving as the primary evaluation metric. The second step is Approximate Design Generation, which fed the computed cost-performance ratio into the converter to obtain a variety of approximate designs.

The third step is Validation Against Abort Condition, which determine whether the generated design meet the predefined abort conditions. If the conditions are not satisfied, the process loops back to the first step. The fourth step is Retraction and Fine-Tuning, which enable the algorithm proceeds to a retraction and fine-tuning phase, optimizing the final design further before outputting the result if the abort conditions are met. Throughout the algorithm, error metrics and hardware metrics are obtained using an applied error model [15] and a hardware model [16], ensuring accurate evaluation of both approximation quality and hardware performance.

1) Application Error Model and Hardware Model: During the process of DSE, it is crucial to perform regular verification to ensure that the current design adheres to specified error constraints while optimizing hardware utilization compared to an exact counterpart. In practice, Monte Carlo methods and DC synthesis are the most frequently employed techniques for achieving these objectives. However, Monte Carlo methods are computationally intensive. Although reducing the number of iterations can decrease runtime, it may lead to intolerable errors and unsatisfactory outcomes in DSE. This issue arises due to the unclear relationship between the number of iterations and the resulting accuracy. From a hardware perspective, increasing the frequency of interfacing with DC during DSE exacerbates system complexity and significantly extends runtime. To address these challenges, this paper employs the analytical model proposed in [15], which computes final errors with higher accuracy and is three orders of magnitude faster than Monte Carlo methods. For hardware evaluation, the theoretical analysis method introduced in [16] is used as a substitute for DC synthesis, aiming to reduce DSE complexity. It is important to note that all outputs are validated using DC to ensure the accurate assessment of approximate designs.

2) Cost-Performance Computation: In PreDAC, the approximation process involves sequentially refining the candidate multipliers within an application. To systematically guide this refinement, it is necessary to define the depth—the number of candidate multipliers currently under consideration. This facilitates the incremental approximation of multipliers in a structured manner. For each iteration, the cost-performance ratio is calculated to evaluate the trade-offs of replacing a candidate multiplier, starting from the current depth to the final multiplier, with a substitute from the approximate library that exhibits a higher error. This cost-performance ratio serves as the foundation for executing the replacement strategy. It is assumed that the replacement cost-performance ratios for multipliers of the same class are identical.

To illustrate, consider a 3x3 convolution operation with three multipliers in the approximation library, labeled 1, 2, and 3. Initially, all candidate multipliers are exact (denoted as 0). Following several iterations, the resulting configuration might appear as [1, 1, 0, 1, 0, 2, 1, 1, 0]—a total of nine candidate multipliers of the same class. At this stage, the cost-performance ratio must be computed for various replacement scenarios. For example, the fourth multiplier could be replaced with approximate multipliers 1, 2, or 3; similarly, the fifth multiplier might be replaced with multiplier 3, and the sixth multiplier with multipliers 2 or 3. This paper introduces a novel cost-performance formula with adjustable hyperparameters, enabling its calibration to meet specific accuracy and hardware requirements. The formula is defined as follows:

$$\begin{aligned} \mathrm{a} &= \frac{HO_o^x - HO_l^x}{HO_o^x - HO_{min}^x} \\ \mathrm{b} &= \frac{error_l^y - error_o^y}{error_{max}^y - error_o^y} \\ \mathrm{c_p} &= \left|\frac{a}{b}\right|, \end{aligned} \tag{1}$$

where HO_o represents the hardware overhead of the original design, HO_l is the hardware overhead after replacement, and HO_{min} denotes the hardware overhead of replacing the multiplier with the largest error. Similarly, $error_l$ refers to the error of the replaced design, while $error_o$ denotes the error of the original design, and $error_{max}$ corresponds to the error of the largest-error design. Notably, the designs represented by HO_{min} and $error_{max}$ are identical. Both a and b are all normalized, while $\mathrm{c_p}$ represents the cost-performance ratio.

Consider two scenarios where the error metrics are MSE and MED, respectively, while the hardware metrics remain constant. Due to the significant numerical differences between MSE and MED, an unmodified cost-performance formula may disproportionately favor multipliers with lower MSE, leading to slower convergence. Conversely, for MED, the formula may overly prioritize lower hardware overhead, potentially overlooking superior designs and yielding suboptimal outcomes. To address this, distinct hyperparameter settings are applied for different error metrics to ensure balanced cost-performance values. For instance, when the hardware metrics focus on area or power consumption and the error metrics use MSE, the hyperparameters $x = 3, y = 1/2$ are recommended. In contrast, for MED, selecting $x = 3, y = 1$ provides better results. Additionally, error metrics expressed in terms of Signal-to-Noise Ratio (SNR) are converted to MSE for consistency during DSE.

3) Approximate Design Generation,: In this approach, the FPAX [8] converter is utilized to facilitate the generation of approximate designs. Initially, two distinct designs are created by applying the two most cost-effective replacements, prioritizing them in alternating order. These two designs are then subjected to randomization processes inspired by genetic algorithms, incorporating crossover and mutation principles to produce additional designs. The cost-performance ratio for each design is computed using the formula presented in (1). The design with the highest cost-performance ratio is selected as the result of the current iteration.

4) Validation Against Abort Condition: The resulting design generated by II-C3 undergoes evaluation to determine whether its error margin complies with the predefined constraints. If the error margin violates these constraints, the termination condition is triggered, and the retraction and fine-tuning phase is initiated to refine the design. Conversely, if the

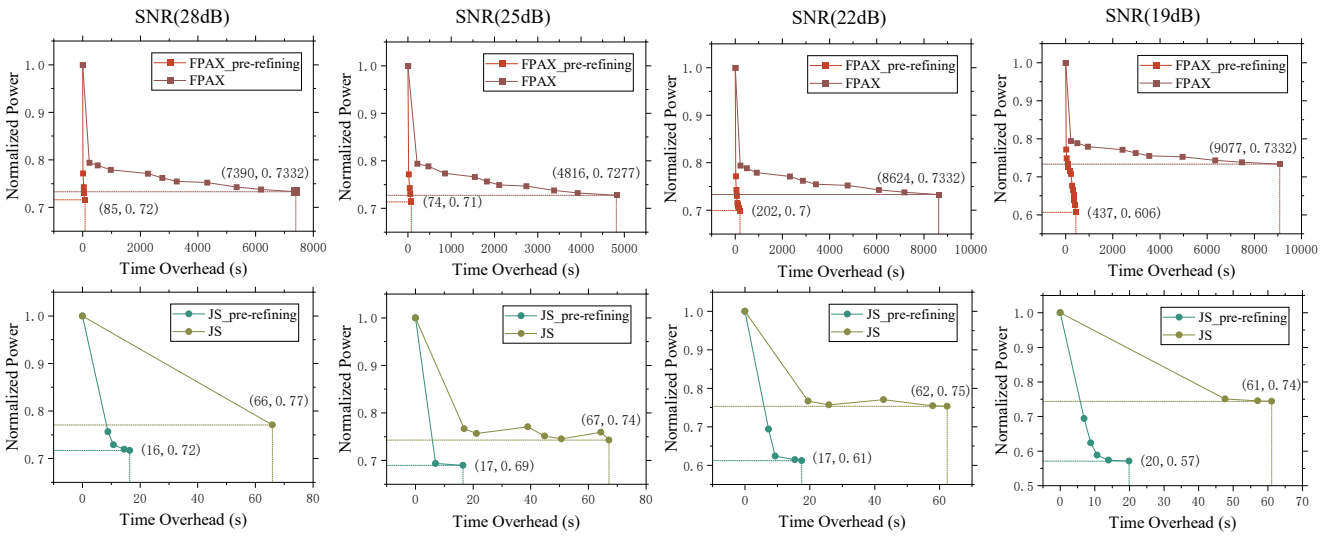


Fig. 3. Comparison between the DSE process with and without a pre-refining step under four different SNR constraints for power consumption optimization of 13-tap FIR, using FPAX and JS framework, respectively.

design satisfies the error constraints, the process continues to the next iteration.

5) Retraction and Fine-Tuning: When the error exceeds the prescribed constraint, a retraction step is applied to revert the candidate multiplier to the state from the preceding iteration. This process begins at the current depth and proceeds forward until the constraint is satisfied. Due to the inherent imprecision of retraction and the potential for further approximation in the resulting output, a subsequent fine-tuning step is introduced to optimize the design.

During fine-tuning, each candidate multiplier is incrementally approximated by one level, replacing it with the next higher-numbered approximate multiplier from the library. These replacements are performed individually for each candidate to generate multiple designs. Designs that satisfy the error constraint are then selected for evaluating their cost-performance ratios. Replacements are executed in descending order of cost-performance ratio until the error constraint is no longer met. At this point, the prior step is restored, and the corresponding design is finalized as the output. If all replacement strategies are applied without breaching the error constraint, the process undergoes a single additional iteration to further refine the design based on the revised configuration.

III. Evaluation and Analysis

To evaluate the effectiveness of the PreDAC framework, four widely used applications were selected for analysis: RGB2YCbCr, 3x3 Convolution, 13-tap FIR Filter, and Gaussian Blur Filter. These applications, as depicted in Fig. 2, are representative of image processing, artificial intelligence, and communication systems. While PreDAC can accommodate various error and hardware metrics, this study focuses on approximate computation gains in terms of area and power consumption. Accordingly, these two hardware metrics were chosen for evaluation.

For accuracy metrics, the MED and SNR were selected. MED evaluates whether the approximation error is positively or negatively biased, whereas SNR serves as a key performance metric in a broad range of applications, including image processing and communication systems. The comparative frameworks include JUMP search (JS) [9] and FPAX [8]. Both frameworks utilize a stepwise approximation approach instead of randomly generating designs within the design space, ensuring faster runtime compared to other methods. The evaluation process in this paper is divided into two main sections: (1) assessing the pre-refining step and (2) evaluating the enhanced DSE algorithm.

The PreDAC framework was implemented in Python, and experiments were conducted on a system with the following specifications: Intel(R) Xeon(R) Gold 6258R CPU (2.70 GHz) with 112 cores and 192 GB of RAM.

A. Pre-refining Step Evaluation

To evaluate the effectiveness of the proposed pre-refining step, experiments were conducted using the JS [9] and FPAX [8] frameworks, with and without the pre-refining step, respectively. The objective of hardware optimization was to minimize power consumption, and the design space exploration for the 13-tap FIR filter was performed under four SNR constraints: 28 dB, 25 dB, 22 dB, and 19 dB. The results are depicted in Fig. 3, where the horizontal axis represents runtime and the vertical axis represents power consumption, normalized to the exact design.

In the FPAX framework, the number of iterations is limited to 10 due to the excessive runtime when the pre-refining step is omitted. The FPAX algorithm computes the cost-performance ratio for each multiplier in the approximate library during every iteration. Notably, the number of multipliers before pre-refining is approximately five times larger than after pre-

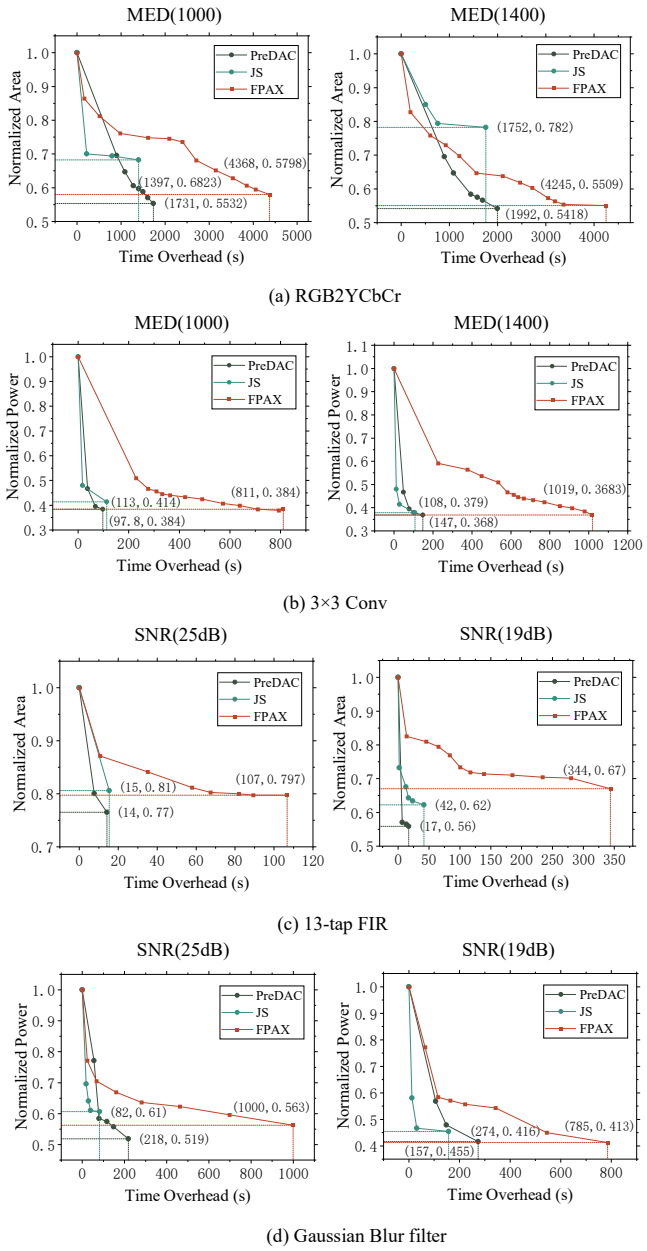


Fig. 4. Comparison of Exploration Efficiency between PreDAC, JS and FPAX using the same Pre-refining Step.

refining, leading to a speedup of up to 87× when the pre-refining step is applied, as shown in Fig. 3.

A similar runtime issue exists in the JS algorithm, although its error metric is calculated only once per iteration. As a result, the speedup for JS is approximately 4× on average when the pre-refining step is employed. Without pre-refining, the approximate library contains numerous non-Pareto multipliers with higher errors and worse power consumption, unnecessarily inflating the design space, slowing convergence, and yielding suboptimal outcomes. By implementing the pre-refining step, the JS algorithm achieves a further 23% reduction in power consumption compared to the scenario without pre-refinement. This demonstrates the pre-refining step's ability to significantly enhance both efficiency and optimization quality in the design space exploration process.

B. DSE Algorithm Evaluation

The proposed enhanced DSE algorithm was compared against the JS and FPAX frameworks under identical pre-refining conditions. The experiment incorporated two error metrics, two hardware metrics, and four fault-tolerant applications. The experimental results are shown in Fig. 4. The selected MED constraints are 1000 and 1400 in order, and the SNR are 25dB and 19dB.

As illustrated in Fig. 4, the proposed algorithm demonstrated comparable or superior performance relative to FPAX, achieving an average speedup of 7.7×. This improvement is attributed to the use of an application error model instead of Monte Carlo simulations, which significantly reduces the computational time required for error metric evaluation—a major component of the DSE algorithm runtime. Additionally, the enhanced cost-performance formula introduced in this study can be adaptively calibrated to balance runtime and design quality. This adaptability directs iteration outcomes towards maximizing gains and facilitates early termination, further minimizing runtime.

Compared to the JS algorithm, PreDAC achieved a similar runtime in most cases, with any runtime increases falling within acceptable limits. Furthermore, the enhanced algorithm delivered an average improvement of 8.8% in hardware overhead optimization. The precision of the application error model used in this study surpasses that of the Monte Carlo method, while the enhanced cost-performance formula enables the selection of appropriate hyperparameters to guide the DSE algorithm effectively. Finally, the retraction and fine-tuning steps proposed here exploit the remaining approximation space to optimize hardware overhead further while adhering to the specified error constraints.

IV. Conclusion

The PreDAC framework is proposed to streamline design space exploration by providing a library of customized approximation multipliers tailored to the application's input distribution, optimization objectives, and accuracy constraints. The framework incorporates an enhanced cost-performance formula and a fine-tuning step. This integration allows for an optimal balance between performance and design quality. The effectiveness of the pre-refining method and the improved DSE algorithm was evaluated independently. When combined with the pre-refining step, FPAX achieved the highest speedup, up to 87× compared to its original implementation. Meanwhile, the JS algorithm [9] demonstrated a 23% improvement in hardware overhead optimization. Under the condition of using the same pre-refining steps, the PreDAC framework achieved an additional average optimization of 8.8% in design quality while maintaining a runtime comparable to that of JS. Moreover, PreDAC delivered results that were comparable to or better than FPAX, achieving an average speedup of 7.7×.

REFERENCES

[1] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, "Roba multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 393–401, 2017.

[2] H. Jiang, L. Liu, P. P. Jonker, D. G. Elliott, F. Lombardi, and J. Han, "A high-performance and energy-efficient fir adaptive filter using approximate distributed arithmetic circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 1, pp. 313–326, 2019.

[3] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited: Cross-layer approximate computing: From logic to architectures," in *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6.

[4] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "autoax: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[5] M. Vaeztourshizi, M. Kamal, and M. Pedram, "Egan: A framework for exploring the accuracy vs. energy efficiency trade-off in hardware implementation of error resilient applications," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, 2020, pp. 438–443.

[6] Z. Zhang, T. Chen, J. Huang, and M. Zhang, "A fast parameter tuning framework via transfer learning and multi-objective bayesian optimization," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, ser. DAC '22. Association for Computing Machinery, 2022, p. 133–138. [Online]. Available: https://doi.org/10.1145/3489517.3530430

[7] L. Ferretti, J. Kwon, G. Ansaloni, G. D. Guglielmo, L. P. Carloni, and L. Pozzi, "Leveraging prior knowledge for effective design-space exploration in high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3736–3747, 2020.

[8] Y. Dou, C. Wang, H. Waris, R. Woods, and W. Liu, "Fpax: A fast prior knowledge-based framework for dse in approximate configurations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 6, pp. 1650–1662, 2024.

[9] L. Witschen, H. Ghasemzadeh Mohammadi, M. Artmann, and M. Platzner, "Jump search: A fast technique for the synthesis of approximate circuits," in *Proceedings of the 2019 Great Lakes Symposium on VLSI*, ser. GLSVLSI '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 153–158.

[10] S. Hashemi, R. I. Bahar, and S. Reda, "Drum: A dynamic range unbiased multiplier for approximate applications," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2015, pp. 418–425.

[11] K. Y. Kyaw, W. L. Goh, and K. S. Yeo, "Low-power high-speed multiplier for error-tolerant application," in *2010 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC)*, 2010, pp. 1–4.

[12] Z. Cui, K. Chen, B. Wu, C. Yan, Y. Gong, and W. Liu, "A time efficient comprehensive model of approximate multipliers for design space exploration," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*, 2024, pp. 116–123.

[13] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, 2017, pp. 258–261.

[14] E. Martin, O. Sentieys, H. Dubois, and J. Philippe, "Gaut: An architectural synthesis tool for dedicated signal processors," in *Proceedings of EURO-DAC 93 and EURO-VHDL 93- European Design Automation Conference*, 1993, pp. 14–19.

[15] S. Chen, K. Chen, Z. Cui, and W. Liu, "A high-accuracy time-efficient error metric model for approximate computing circuits," in *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2024, pp. 94–99.

[16] J. Castro-Godínez, J. Mateus-Vargas, M. Shafique, and J. Henkel, "Axhls: Design space exploration and high-level synthesis of approximate accelerators using approximate functional units and analytical models," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2020, pp. 1–9.

BlockPIM: Optimizing Memory Management for PIM-enabled Long-Context LLM Inference

Zhichun Li[1,2], Jun Zhou[1,2], Xueqi Li[1,*], Ninghui Sun[1,2]
[1]State Key Lab of Processors, Institute of Computing Technology, CAS [2]University of Chinese Academy of Sciences
{lizhichun23z, zhoujun23s, lixueqi}@ict.ac.cn, snh@ncic.ac.cn

***Abstract*—Processing-In-Memory (PIM) architectures alleviate the memory bottleneck in the decode phase of large language model (LLM) inference by performing operations like GEMV and Softmax in memory. However, the fragmented data layout in current PIM architectures limits end-to-end acceleration for long-context LLMs. In this paper, we propose BlockPIM, a cross-channel block memory layout strategy that maximizes memory utilization and eliminates the context length constraint. Additionally, we introduce a cross-channel attention computation scheme that is compatible with the current architecture to support distributed attention operations on BlockPIM. Experimental results demonstrate that our approach achieves a 62% average throughput increase compared to existing state-of-the-art PIM solutions, enabling efficient and scalable deployment of large language models on PIM architectures.**

I. Introduction

The large language models (LLMs) inference process consists of two stages: the *prefill stage* and the *decode stage*. The prefill stage processes the input prompt to generate initial context, while the decode stage follows, where the model generates tokens sequentially using the learned context and previous tokens to predict the next one. As demonstrated before, the main bottleneck is the decode stage and the performance is heavily constrained by memory bandwidth due to low compute density. While batching techniques can improve compute density for some operations by reusing parameters, the attention operation still suffers from extensive memory access between the processing units and the memory, making the memory a major performance bottleneck.

PIM (Processing-In-Memory) architectures is considered an ideal technological path [1]–[4], which integrates specialized processing elements (PEs) near memory banks or on buffer dies, enabling memory-intensive operators such as GEMV and Softmax—key components of attention—to be offloaded. However, to improve parallelism, fully utilize internal bandwidth (e.g., 242 TB/s in AttAcc [3]), and reduce communication overhead between PEs, these PEs are placed across memory channels for independent attention operands computations. Correspondingly, the memory-centric PIM design also leads to specific requirements on the physical layout of operands in memory, resulting in a unique data layout scheme, as shown in Fig.1. For instance, in the NeuPIMs [4] design, the entire attention computation for a single request must be completed within a single memory channel. Consequently, all KV cache associated with that request must be stored exclusively within the same channel, deviating from the conventional channel-interleaved memory storage approach used in standard memory architectures. While this design enhances computational efficiency within each channel, it introduces constraints on data placement that may affect overall system performance and flexibility.

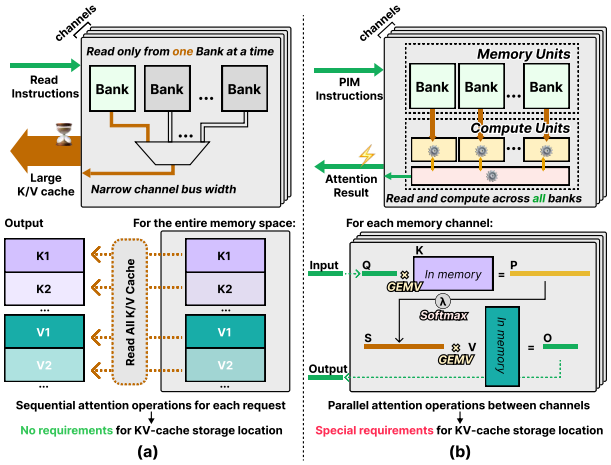


Fig. 1. Architectural changes cause data layout transformations. (a) Traditional memory architecture, (b) Processing-In-Memory architecture, and the respective data flows during attention operations.

We observed that in long-context inference scenarios, the existing **data layout scheme** presents two major issues. First, there exists memory fragmentation: each memory channel wastes space that is insufficient to store a complete KV cache, thereby reducing the number of requests that can be processed in parallel. Compared to the unified memory layout used in traditional GPUs, the current scheme leads to approximately 20% memory waste, resulting in an estimated 10% reduction in inference performance. The second issue is redundant computation, as channel-independent memory management prevents KV cache sharing across channels, undermining the effectiveness of the prefix caching mechanism. Furthermore, as context lengths continue to increase, the current layout inherently limits support for longer contexts. For example,

*Corresponding Author: Xueqi Li (lixueqi@ict.ac.cn). This work was supported in part by the National Natural Science Foundation of China under Grant No. 62488101 and 62202454, the National Key R&D Program of China under Grant 2022YFB4500403.

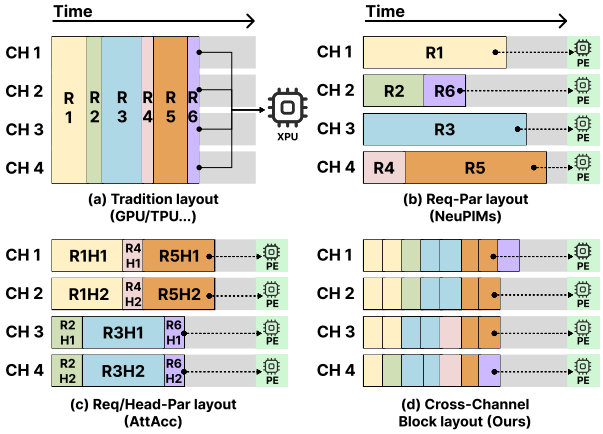


Fig. 2. Different memory layouts for distributing the KV cache of six request across four channels.

when running the Llama-3.1 70B model on a NeuPIM system with the existing layout, the maximum supported context length is only 25.6K tokens, significantly lower than the model's potential of 128K tokens.

Driven by the **goal** of optimizing memory management to overcome these challenges, we proposed BlockPIM, a cross-channel block memory layout scheme, that maximizes memory utilization and eliminates the context length constraint. Moreover, BlockPIM does not incur any redundant storage and computation when applying prefix caching. To support distributed attention operation on BlockPIM, we proposed a cross-channel attention computation mechanism that only requires minor modifications to the existing architecture. Experimental results demonstrate that our approach achieve 62% throughput increase on average compared with existing PIM solution, enabling efficient and scalable deployment of large language models on PIM architectures.

II. Background

A. Existing Memory Layout

During inference, multiple requests are typically grouped into a batch and processed in parallel. Each request stores intermediate values, known as the KV cache, in memory. Traditional XPUs and PIM-soluton adopt different strategies for organizing the KV caches in memory due to their differing approaches to computing attention.

1) Traditional Layout: In traditional memory, the KV cache of a request is evenly distributed across channels due to channel-level memory interleaving, as shown in Fig. 2(a). When performing attention on this request, XPU simultaneously reads different parts of the KV cache from different channels. Since each request has a unique KV cache, traditional XPU execute attention operations sequentially for each request when processing the attention layer. From a memory perspective, the XPU accesses the KV cache one at a time.

2) Req-Par Layout: In the NeuPIMs design, the KV cache for each request is stored within a single channel, while KV caches for different requests are distributed across channels. A greedy algorithm assigns a new request's KV cache to the channel with the lowest usage. For instance, as illustrated in Fig.2, assuming requests 1 through 6 are allocated sequentially, request 6 would be placed in the second channel, as it has the lowest usage after assigning the first five requests. During attention operations, channels perform computations in parallel, processing requests sequentially within each channel. We term this strategy **Req-Par**, as it exploits the independence of attention computations across requests, enabling parallel attention execution across channels.

3) Req/Head-Par Layout: The AttAcc memory layout strategy extends Req-Par layout by introducing parallelism across attention heads, as shown in Fig. 2(c). We use N_{kvhead}^{device} to represents the number of KV-heads allocated on this device under a tensor parallelism scheme. For each request, KV cache are stored across N_{kvhead}^{device} channels. When a new request is added to the batch, the KV cache are divided into N_{kvhead}^{device} segments along the attention head dimension and assigned to the lowest usage channels using a greedy algorithm. We term this strategy **Req/Head-Par** for exploiting independent computations across requests and attention heads. However, with the increasing adoption of *Grouped Query Attention*, the number of KV-heads has substantially decreased. Even the largest open-source model, Llama 3.1 405B, uses only 8 KV-heads, as shown in Table I. In an 8-way tensor parallelism setup, each device handles a single KV-head, reducing the Req/Head-Par layout effectively to a Req-Par layout. Therefore, for simplicity in discussion, all PIM layouts mentioned later are considered equivalent to the Req-Par layout.

B. Prefix Caching

Prefix Caching (also known as Prompt Caching) is a critical technique widely employed in modern inference frameworks such as vLLM [5], SGLang [6], TensorRT-LLM [7], LMDeploy [8]. By optimizing memory usage and reducing redundant computation, it effectively lowers response latency and enhances system throughput.

Inference systems often handle multiple requests that share the same prefix, leading to repeated computations for these prefixes. Additionally, storing multiple copies of identical prefix KV caches results in memory inefficiency. Based on this observation, Prefix Caching identifies and caches the attention computation results for these shared prefixes, enabling direct reuse in subsequent requests. This approach offers two main advantages: First, reusing cached KV caches reduces the computational load in the prefill stage, thereby decreasing response latency. Second, avoiding redundant storage of KV caches in memory allows the limited memory to accommodate more requests, increasing the batch size and further boosting system throughput.

III. Motivation

A. Memory Underutilization

In the existing PIM memory layout, each request must be allocated to one or more fixed channels. If a new request is unable to fit into any channel, it can only wait for other

requests to release memory, even if the total available memory across channels is far greater than the space required by this request. This caused memory fragmentation for each channel, thus reducing the number of requests that can be processed in parallel and ultimately degrading inference performance.

Assuming a configuration of 40 channels, each with a capacity of fitting 32K tokens' KV cache, we conduct a simulation excluding any memory overheads beyond the KV cache. For this simulation, we select requests from the Mooncake [9] trace dataset where the total length does not exceed 32K tokens. According to the simulation results, the traditional layout achieves a memory utilization rate of 99.55%, while the Req/Head-par layout reduces this rate to 79.57%, with the average batch size decreasing from 141.37 to 112.88. When estimating based on a GPU platform, such a reduction in batch size implies an 18.8% decrease in throughput.

B. Inefficiency with Prefix Caching Algorithm

Prefix Caching builds upon a paged memory strategy, dividing memory into fixed-size pages that store KV caches. Pages may be shared across requests or dedicated to individual ones. Once all associated requests are completed, a page is marked releasable but retains data to improve reuse. When memory is full, the scheduler overwrites the least-reusable releasable page for new KV caches.

The Req-Par and Req/Head-Par layouts essentially forgo traditional channel-level memory interleaving, instead establishing an independent memory space for each channel. This approach results in incompatibility with Prefix Caching, which is designed for a unified memory space.

A direct adaptation is to implement paging independently within each channel, limiting page reuse to within channels. However, due to the greedy allocation scheme, requests with the same prefix may not be assigned to the same channel, preventing cross-channel page sharing. Consequently, the KV cache for shared prefixes is redundantly stored across channels, reducing the effectiveness of Prefix Caching and introducing repeated computation overhead. To make matters worse, due to the general applicability of large models, shared prefixes in inference services are often more than one. Multiple prefixes within a limited channel space compete with each other, and when memory is insufficient, the KV cache of existing prefixes is overwritten by the new request's prefix. This greatly reduces the likelihood of prefix reuse.

C. Limited Context Length

In the Req-Par layout, all KV cache for a single request is stored in a single memory channel. However, the capacity of a single memory channel is quite limited. As recent model context length increases rapidly, the memory usage of the KV cache for a single request can easily exceed the capacity of one memory channel. For a request with a length of L tokens, the memory usage of its KV cache on a device in FP16 precision is:$\text{Mem}_{\text{request}} = 2 \times N_{\text{layers}} \times D_{\text{head}} \times N_{kvhead}^{device} \times L \times 2$, where N_{kvhead}^{device} represents the number of K/V heads allocated to this device after tensor parallelism. Given model parameters and channel capacity, we can estimate the maximum context length supported by a single channel in the Req-Par layout: $L_{\text{max}}^{\text{Req-Par}} = \text{Mem}_{\text{channel}}/(2 \times N_{\text{layers}} \times D_{\text{head}} \times N_{kvhead}^{device} \times 2)$.

In the Req/Head-Par layout, the KV cache corresponding to different KV heads of a request are distributed across different channels, reducing the memory pressure on a single channel. Thus, $L_{\text{max}}^{\text{Req/Head-Par}} = \text{Mem}_{\text{channel}}/(2 \times N_{\text{layers}} \times D_{\text{head}} \times 1 \times 2)$.

For a typical 8Gb HBM3 memory channel and various models, Table I below shows the maximum allowed context length under different Req-Par and Req/Head-Par layouts. It can be observed that, particularly for models supporting long contexts, the context length supported by these two memory layouts is far below the potential context length capability of the models. The context length of advanced large models has already surpassed 1M (e.g., Gemini Pro [10]), while the physical capacity of memory channels is challenging to increase proportionally, making it difficult for existing PIM architectures to support large models with long contexts under current memory layouts.

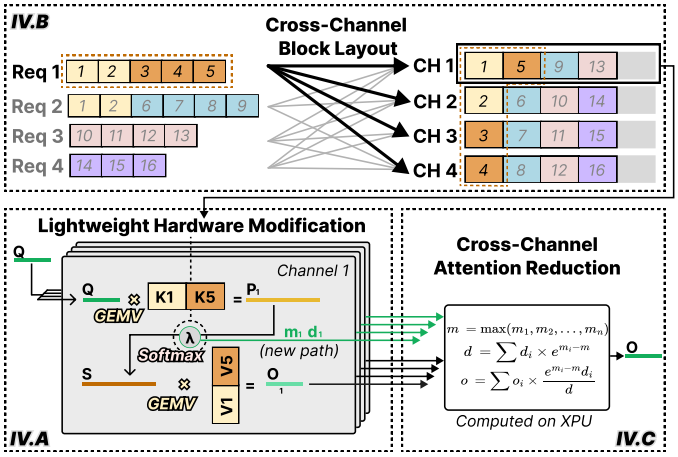


Fig. 3. BlockPIM design overview

IV. BlockPIM Design

As shown in Fig.3, BlockPIM consists of three key components: (1) Lightweight hardware modification, which gives architectural support for our cross-channel block layout scheme and cross-channel attention computation algorithm. (2) Cross-channel block layout, which reduces memory underutilization and redundancy in prefix caching, while overcoming context length limitations of traditional layouts with architectural and algorithmically support. (3) Cross-channel attention computation, ensuring correct attention results under the cross-channel block layout.

A. Lightweight Hardware Modification

We first briefly summarize the system architecture of the PIM-enabled computing system, then focusing on the lightweight architectural support introduced in our design.

System architecture. As illustrated in Fig.4(a)-(c), the PIM-device and XPU (e.g., GPU, TPU) are interconnected via

	Model Parameters				Contexts length	Maximum allowed contexts length				
	Name	D_{head}	$N_{kvheads}$	N_{layers}		Req-Par (No TP)	Req-Par (4-way TP)	Req-Par (8-way TP)	Req/Head-Par	BlockPIM
Short context Model	OPT 175B	128	96	96	2K	0.2K	0.9K	1.8K	21.3K	853.3K
	Gemma-2 27B	128	16	46	8K	2.8K	11.1K	22.3K	44.5K	1780.9K
Long context Model	Qwen-2 72B	128	8	80	128K	3.2K	12.8K	25.6K	25.6K	1024.0K
	Llama-3.1 70B	128	8	80	128K	3.2K	12.8K	25.6K	25.6K	1024.0K
	Mistral-Large-2 123B	128	8	88	128K	2.9K	11.6K	23.3K	23.3K	930.9K
	Llama-3.1 405B	128	8	126	128K	2.0K	8.1K	16.3K	16.3K	650.2K

TABLE I
CONTEXT LENGTH VARIATIONS FOR DIFFERENT MODELS UNDER VARIOUS MEMORY LAYOUT STRATEGIES

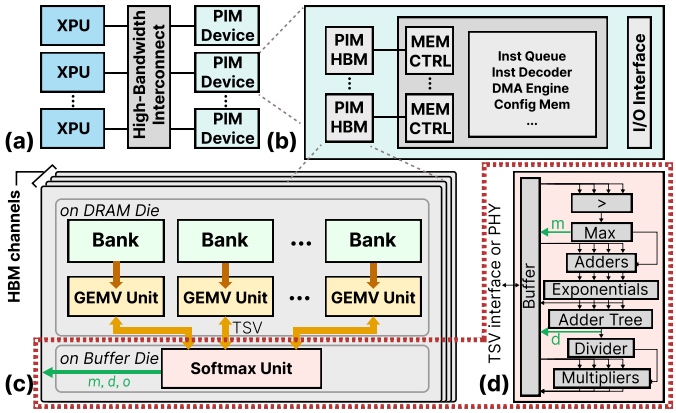


Fig. 4. Architecture of the PIM-enabled computing system in BlockPIM: (a) System hardware components, (b) PIM device structure, (c) Internal structure of PIM-enabled HBM channels, and (d) Softmax Unit with minor modifications (green arrow).

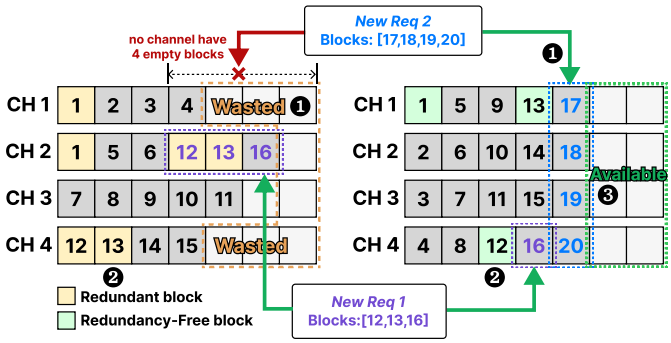


Fig. 5. Comparison of Req-Par block layout from NeuPIMs (left) and cross-channel block layout (right) highlighting the advantages of the latter.

high-bandwidth links, such as NVLink or PCIe. Each PIM-device consists of multiple HBM memory chips, managed by a centralized control chip responsible for issuing memory access or processing-in-memory commands. Within each HBM channel, dedicated computing units for operations like GEMV and Softmax are embedded within the DRAM die or buffer die to collaboratively perform attention computation.

Lightweight modification. In Fig.4(d), our modification targets the internal structure of the Softmax unit. When the maximum value m of the input vector and the denominator d are computed, instead of simply passing these values for subsequent calculations, we introduce additional connections (green arrow in the figure) to the buffer and modify the control logic, enabling m and d to be directly stored in the buffer. After completing the Softmax computation, m and d, along with the output vector o, are transferred to the XPU to continue cross-channel attention computation, as detailed in IV-C.

B. Cross-channel Block Memory Layout

Layout Strategy. We propose the cross-channel block layout to address limitations in existing memory layouts. In this design: (1) The KV cache generated for each new request is divided into fixed-size blocks, with each block representing a fixed number of tokens. (2) These blocks are distributed sequentially across memory channels using a greedy strategy, which allocates each block to the channel with the least current capacity utilization. (3) During inference, newly generated KV cache blocks are similarly assigned using the greedy strategy, ensuring that the KV caches of all requests remain approximately balanced across channels.

In contrast to NeuPIMs [4], which employs a block management strategy akin to PagedAttention [5], the cross-channel block layout optimizes block distribution across channels rather than confining them within individual channels. NeuPIMs' isolated block management perpetuates inefficiencies found in Req-Par and Req/Head-Par layouts. By balancing KV cache blocks across channels, the cross-channel block layout effectively addresses these inefficiencies, offering a more coordinated and efficient memory strategy.

Advantages. Our proposed layout implements a unified, cross-channel block management system, which brings three significant advantage, as shown in Fig. 5: ❶ The placement of new request's KV cache is no longer limited by the remaining space of a single channel, which improves the utilization of memory space. This avoids wasting space in each memory channel that cannot accommodate an entire KV cache. In other words, under our layout, these wasted spaces are used reasonably due to the finer-grained allocation method. ❷ No redundant computation or storage in Prefix Caching. Under BlockPIM's layout, a new request can directly reuse the KV cache of a shared prefix stored in any location, rather than

being limited to the space within a single channel. This means that the shared prefix only needs to be computed and stored once, eliminating redundant storage across channels. Moreover, the shared prefix KV cache no longer needs to compete for space within a single memory channel, significantly reducing KV cache invalidation and improving reuse rates. ❸ A request's KV cache can be distributed across multiple channels, maximizing the utilization of all available storage, akin to traditional layouts, which effectively mitigates the context length limitation in existing layouts. With this cross-channel layout, the capacity limit for a single request's KV cache extends beyond the constraint of a single channel, equating to the cumulative size of all available memory blocks within the system.

Block Size Considerations. To determine block size, it is essential to consider DRAM memory access characteristics, particularly the impact of row switching. Since accessing data from different rows incurs significant latency, it is beneficial to utilize all data within a single row before switching to another. Thus, consecutive K or V cache data is stored in a single row of each DRAM bank. However, we store the K and V cache in separate rows of each DRAM bank, as the Softmax computation between reading the K and V cache helps hide the row switching latency. As a result, the block size is twice the product of the total number of banks of a channel and the capacity of a single bank row.

C. Cross-channel Attention Computation

Computation Steps. For the attention computation process of a request, our cross-channel attention computation is divided into three steps: ❶ Within each channel, all KV cache blocks belonging to the request (denoted as K_i and V_i, where i represents the channel index) compute attention with the input Q vector, yielding the local attention result $O_i = \text{Softmax}\left(\frac{QK_i^T}{\sqrt{d_k}}\right)V_i$. Each channel's buffer simultaneously records two intermediate value m_i and d_i during the attention process. Here, m_i is the maximum value used in softmax, and d_i is the softmax normalization base value. After computation, O_i, m_i, and d_i are transmitted to the XPU. ❷ On the XPU, the global m and d values are computed sequentially using the following formulas:$m = \max(m_1, m_2, \ldots, m_n)$, $d = \sum d_i \times e^{m_i - m}$. ❸ On the XPU, based on the global m and d, the partial results from each channel are corrected and reduced to obtain the complete attention computation result: $O = \sum O_i \times \frac{e^{m_i - m} d_i}{d}$.

Our cross-channel attention computation enables the attention operation of a request to be decomposed and computed in parallel across multiple channels. As a result, this approach eliminates the need for the operands of the attention operation, the complete KV cache, to be stored entirely within a single channel.

Algorithmically correctness. Our approach ensures correct attention computation under a cross-channel block layout. The conventional attention operation, Attention(Q, K, V), is generally divided into three main steps: $S = \frac{QK^T}{\sqrt{d_k}}, P = \text{Softmax}(S), O = PV$. Under the cross-channel block layout, the K and V matrices are segmented along the sequence length dimension. For a given channel i, the partial output is computed as O_i. However, the overall output O does not simply equal $\sum O_i$; the partial results from each channel cannot be directly summed. This discrepancy arises because the original attention computation requires a softmax operation over the entire global S vector, while each channel i can only utilize the local S_i data. Thus, the attention outputs from each channel must undergo correction prior to summation, The step (2) of our algorithm calculates the correction coefficients, and the step (3) performs the correction and summation, thus ensuring the correctness of the final results.

Extra Overhead. In addition to transmitting the partial result O_i to the host, each softmax unit must also convey the parameters d_i and m_i. Fortunately, transmitting two additional values per channel has minimal impact on communication overhead. The computational cost of the extra operation is also minimal, capped by the correction and summation of d_{head}-length vectors across all channels, regardless of the request length. We extended the supported operators of LLMCompass [11] to simulate this overhead in experiments.

V. Evaluation

A. Experimental Setup

Simulation Settings. (1) Software simulation: An inference engine simulator was implemented which faithfully incorporated different memory layout strategies. The simulation employed a First-Come-First-Serve (FCFS) policy to handle incoming inference requests. The evaluation is conducted on the representative long-context model, Llama-3.1 70B. (2) Hardware simulation: The AttAcc simulator [3], built on the cycle-accurate memory simulator Ramulator [12], was utilized to model processing-in-memory (PIM) behavior. For the XPU component, we leveraged LLMCompass [11], a specialized simulator designed for evaluating hardware performance in LLM inference tasks. LLMCompass is calibrated against real hardware performance metrics to ensure accuracy.

Block Size Configuration. In our configuration, each HBM channel has 64 banks, and each bank row stores 1KB of data. This means the read/compute unit for each all-bank PIM operation is 64KB. With FP16 precision, each token requires 128×2B of memory for the K or V cache, allowing 64KB to store data for 256 tokens. Thus, the total block size, which stores both the K and V cache for 256 tokens, is 128KB.

Datasets. (1) ShareGPT [13]: Publicly shared ChatGPT logs, featuring short-input, short-output interactions. Since ChatGPT's system prompt is private, we use Claude 3.5 Sonnet's official system prompt [14] (2,658 tokens) as an approximate prefix for user requests. (2) Mooncake [9]: Real-world inference traces from Moonshot, with medium-to-long inputs and short outputs. The dataset itself includes details on prefix reuse between requests. (3) LongWriter [15]: A dataset for long-text generation, emphasizing short-input, long-output scenarios. There is no prefix reuse in this dataset. (4) LooGLE [16]: A benchmark dataset for long-context

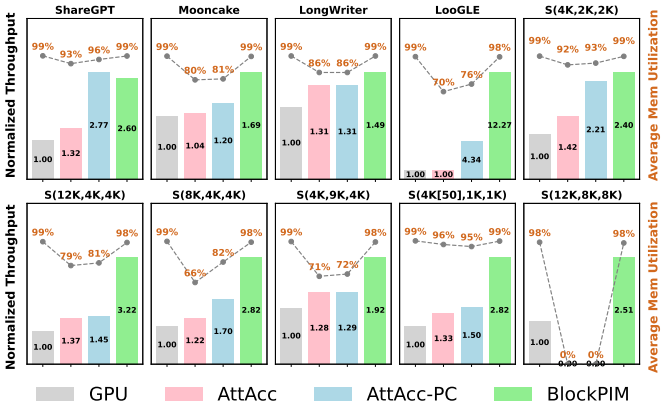


Fig. 6. Performance Comparison

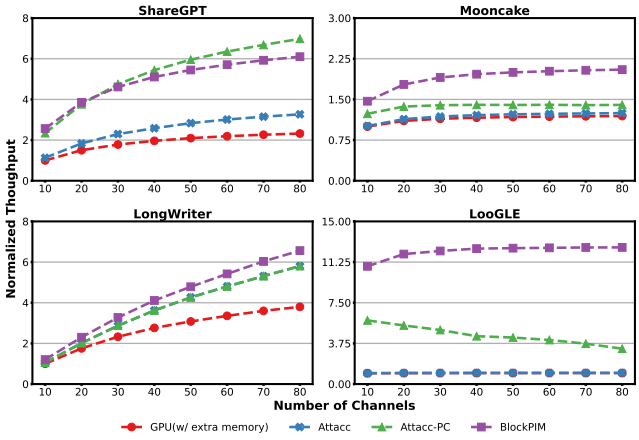


Fig. 7. Scalability Analysis

models, highlighting long-input, short-output cases. We only use the Q/A test samples for long context understanding, where each long text is reused for different Q/A pairs. (5) Synthetic datasets: Given the scarcity of public long-context inference trace datasets, we generated six synthetic datasets $S(prefix, input_{mean}, output_{mean})$ for more comprehensive comparisons across different data characteristics. Each synthetic dataset has 1,000 samples with input and output lengths normally distributed. In $S(4K[50], 1K, 1K)$, "50" denotes 50 distinct system prompts.

Baselines. (1) GPU: A standard DGX server configuration with 8×A100 GPUs. (2) AttAcc: Extends the GPU configuration with 8 PIM devices, each equipped with 40GB of HBM3 memory (1GB per channel) and memory timing aligned with AttAcc specifications. (3) AttAcc-PC: In addition to the AttAcc configuration, prefix caching are enabled per channel.

B. Performance Comparison

As shown in Fig. 6, we compare the acceleration achieved by our approach relative to the baseline across different datasets. Each set of data can also serve as an ablation study, reflecting the individual performance gains contributed by PIM, prefix caching, and our optimization. In the ShareGPT dataset, with its short system prompts, inputs, and outputs, memory fragmentation and redundant computation issues are minimal, resulting in comparable speedups for BlockPIM and AttAcc-PC, with BlockPIM being slightly lower (2.77x→2.60x) due to the extra overhead. However, across all other long-context datasets, BlockPIM consistently delivers performance improvements. In the LongWriter dataset, which does not use shared prefixes, the performance improvement of BlockPIM over AttAcc-PC (1.31x→1.48x) is entirely attributed to more efficient utilization of memory resources. In the LooGLE dataset, which highlights extreme cases of shared ultra-long prefixes, BlockPIM achieves a significant performance gain by avoiding redundant computation/storage, with the speedup increasing from 4.33x to 12.2x compared to AttAcc-PC. For the Mooncake dataset, the performance improvement (1.20x→1.68x) results from the combined effect of addressing both memory fragmentation and redundant computation issues. Our first five synthetic datasets demonstrate the acceleration of BlockPIM over AttAcc-PC under long input and output scenarios, with the average speedup improving from 1.62x to 2.63x. Due to AttAcc's limitations in context-length, it cannot process the final synthetic dataset, whereas our BlockPIM effectively achieves a 2.51x speedup.

C. Scalability Analysis

As shown in Fig.7, we evaluated the normalized throughput of AttAcc, AttAcc-PC, and BlockPIM as PIM memory channels increased, with the GPU (with extra memory) configuration serving as a baseline to reflect the performance improvement attributed solely to increased memory capacity. (1) ShareGPT dataset: AttAcc-PC and BlockPIM achieved similar scalability, showing that BlockPIM's optimization for long-context inference does not hinder short-context performance. (2) Mooncake dataset: Performance gains from additional PIM channels were limited due to XPU bottlenecks in long-input, short-output tasks. However, BlockPIM still demonstrated better scalability than AttAcc-PC. (3) LongWriter dataset: BlockPIM consistently outperformed AttAcc-PC by leveraging memory resources more efficiently. (4) LooGLE dataset: AttAcc-PC performance decreased with more channels, as prefix reuse efficiency dropped due to distribution across channels, leading to redundant computations. BlockPIM avoided this issue, maintaining stable performance.

VI. Conclusion

This paper proposed BlockPIM, a novel memory management scheme for PIM-enabled LLM inference system. BlockPIM maximizes memory utilization, eliminates redundant storage and computation in prefix caching and removes context length constraints. Results show that BlockPIM significantly outperforms the state-of-the-art PIM-solution in long-context scenarios.

REFERENCES

[1] M. He, C. Song, I. Kim, C. Jeong, S. Kim, I. Park, M. Thottethodi, and T. Vijaykumar, "Newton: A dram-maker's accelerator-in-memory (aim) architecture for machine learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 372–385.

[2] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin *et al.*, "Hardware architecture and software stack for pim based on commercial dram technology: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2021, pp. 43–56.

[3] J. Park, J. Choi, K. Kyung, M. J. Kim, Y. Kwon, N. S. Kim, and J. H. Ahn, "Attacc! unleashing the power of pim for batched transformer-based generative model inference," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 103–119.

[4] G. Heo, S. Lee, J. Cho, H. Choi, S. Lee, H. Ham, G. Kim, D. Mahajan, and J. Park, "Neupims: Npu-pim heterogeneous acceleration for batched llm inferencing," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, 2024, pp. 722–737.

[5] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient memory management for large language model serving with pagedattention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023, pp. 611–626.

[6] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez *et al.*, "Sglang: Efficient execution of structured language model programs," *arXiv preprint arXiv:2312.07104*, 2023.

[7] NVIDIA, "TensorRT-LLM," https://github.com/NVIDIA/TensorRT-LLM, 2024, version 0.14.0, Accessed: Nov. 17, 2024.

[8] InternLM, "LMDeploy," https://github.com/InternLM/lmdeploy, 2024, version 0.6.3, Accessed: Nov. 17, 2024.

[9] R. Qin, Z. Li, W. He, M. Zhang, Y. Wu, W. Zheng, and X. Xu, "Mooncake: Kimi's kvcache-centric architecture for llm serving," *arXiv e-prints*, pp. arXiv–2407, 2024.

[10] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang *et al.*, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv preprint arXiv:2403.05530*, 2024.

[11] H. Zhang, A. Ning, R. B. Prabhakar, and D. Wentzlaff, "Llmcompass: Enabling efficient hardware design for large language model inference," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 1080–1096.

[12] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer architecture letters*, vol. 15, no. 1, pp. 45–49, 2015.

[13] S. Team, "ShareGPT," https://sharegpt.com/, 2023.

[14] Anthropic, "System prompts," https://docs.anthropic.com/en/release-notes/system-prompts, claude 3.5 Sonnet, Accessed: Oct. 22, 2024.

[15] Y. Bai, J. Zhang, X. Lv, L. Zheng, S. Zhu, L. Hou, Y. Dong, J. Tang, and J. Li, "Longwriter: Unleashing 10,000+ word generation from long context llms," *URL https://arxiv. org/abs/2408.07055*, 2024.

[16] J. Li, M. Wang, Z. Zheng, and M. Zhang, "Loogle: Can long-context language models understand long contexts?" *arXiv preprint arXiv:2311.04939*, 2023.

Efficient Edge Vision Transformer Accelerator with Decoupled Chunk Attention and Hybrid Computing-In-Memory

Yi Li[1,5,*], Zijian Ye[1,*], Xiangqu Fu[2,4,*], Songqi Wang[1,5], Shucheng Du[6], Ning Lin[1], Dashan Shang[2], Jinshan Yue[2,†], Zhongrui Wang[3,†], Xiaojuan Qi[1,†], Feng Zhang[2,†], Han Wang[1,5]

[1]Department of Electrical and Electronic Engineering, The University of Hong Kong, Hong Kong, China; [2]State Key Lab of Fabrication Technologies for Integrated Circuits and Key Laboratory of Microelectronic Devices and Integrated Technology, Institute of Microelectronics of the Chinese Academy of Sciences, Beijing, China; [3]School of Microelectronics, Southern University of Science and Technology, Shenzhen, China; [4]The School of Integrated Circuits, University of Chinese Academy of Science

[5]Center for Advanced Semiconductor and Integrated Circuits, The University of Hong Kong, Hong Kong, China

[6]ACCESS – AI Chip Center for Emerging Smart Systems, InnoHK Centers, Hong Kong Science Park, Hong Kong, China

[*]Equal contribution to this work.

[†]Corresponding author: yuejinshan@ime.ac.cn; wangzr@sustech.edu.cn; xjqi@eee.hku.hk; zhangfeng_ime@ime.ac.cn.

***Abstract*—Vision Transformers (ViTs) are new foundation models for vision applications. Edge-deploying ViTs to realize energy-saving, low-latency, and high-performance dense predictions have wide applications, such as autonomous driving and surveillance image analysis. However, the quadratic complexity of the self-attention mechanism renders ViTs slow and resource-intensive, particularly for pixel-level dense predictions that involve long contexts. Additionally, the pyramid-like architecture of modern ViT variants leads to an unbalanced workload, further reducing hardware utilization and decreasing the throughput of conventional edge devices. To this end, we propose an algorithm-hardware co-optimized edge ViT accelerator tailored for efficient dense predictions. At the algorithm level, we propose a decoupled chunk attention (DCA) mechanism implemented in a pipelined manner to reduce off-chip memory access, thereby enabling efficient dense predictions within limited on-chip memory. At the architecture level, we introduce a hybrid architecture that combines SRAM-based computing-in-memory (CIM) and nonvolatile RRAM storage to eliminate extensive off-chip memory access, with a fusion scheduling to balance workloads and minimize intermediate on-chip memory access. At the circuit level, a bit/element two-way-reconfigurable CIM macro is proposed to improve hardware utilization across pyramidal ViT blocks with varied matrix sizes. The experimental results on object detection, semantic segmentation, and depth estimation tasks demonstrate that our design can efficiently process patch lengths up to 16384 with a speedup of 18.5×-217.1×, a reduction in memory accesses of 1.7×-7.4×, and an improvement in energy efficiency of 1.8×, under less than 1% performance degradation.**

***Index Terms*—Vision Transformers, compute-in-memory, edge AI accelerator, dense prediction**

I. Introduction

Recently, Vision Transformers (ViTs) [1] have emerged as foundation models in computer vision (CV), outperforming convolutional neural networks (CNNs) in various downstream tasks. By partitioning input images into sequences of patches and leveraging self-attention mechanisms, ViTs excel at capturing global image features and have demonstrated superior performance in dense prediction tasks [2], such as object detection, semantic segmentation, and depth estimation. These tasks involve pixel-level category prediction rather than image-level classification, making them critical for applications that require precise visual interpretation. Moreover, dense prediction tasks have a wide range of potential applications, including autonomous driving, robotics, and surveillance.

Despite their success, deploying ViTs on edge devices presents significant challenges. Unlike cloud servers with vast computational resources, edge devices operate under constraints in power, memory, and storage, while still requiring high accuracy and energy efficiency. The quadratic computational complexity of the self-attention mechanism in ViTs exacerbates these issues, particularly for pixel-level dense predictions involving long sequence lengths and extensive computations.

Several studies have been proposed to mitigate the hardware costs of vanilla Transformers and make them more suitable for resource-constrained edge devices [3]. Software-wise, researchers focus on reducing the computational burden of self-attention in Transformer inference by exploiting the inherent sparsity [4]. Since each token is usually correlated with only a few other tokens, the attention matrix is often sparse, therefore, many computations can be skipped to minimize energy and latency overhead. Hardware-wise, one solution is to optimize data flow scheduling to enhance the efficiency of traditional computing platforms. For instance, the FLASHATTENTION [5] method utilizes tiling techniques to reduce data shuffling between the low-speed, high-bandwidth memory (HBM) of the graphics processing unit (GPU) and the high-speed on-chip static random-access memory (SRAM), resulting in lower system power consumption and reduced latency. Another promising solution is the exploitation of the emerging compute-in-memory (CIM) architecture. With the computational functionality built into the memory block itself, the CIM paradigm significantly reduces the energy and time costs associated with off-chip memory access during Transformer's attention calculations on von Neumann architecture-based edge devices [6].

Nonetheless, deploying ViTs for dense predictions on edge devices remains a formidable challenge due to two key factors. First, pixel-level vision tasks often require significantly longer contexts compared to natural language processing (NLP) tasks, resulting in quadratic growth in memory and computational demands as the image resolution increases [7]. For example, when estimating depth

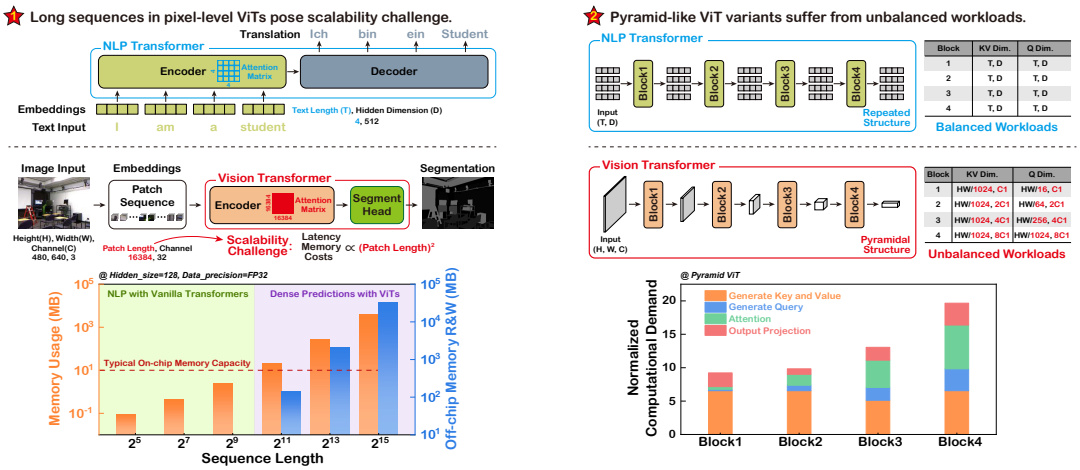


Fig. 1. Dense predictions with ViTs on edge devices face significant challenges, especially with scalability (left) and workload balance (right).

on the ADE20k dataset, the sequence length for the ViT encoder can reach up to 16,384 patches. Second, modern ViTs frequently employ pyramidal architectures to reduce computational costs while capturing multi-scale resolution features [8]. However, the varying feature sizes and matrix dimensions of pyramid ViT blocks not only challenge the optimal resource mapping due to the varying workload in each block but also incur hardware utilization concerns.

To address these challenges, we propose an algorithm-hardware co-optimized edge ViT accelerator tailored for efficient dense predictions. The contributions of this work can be summarized as follows:

- **Algorithmic Level:** We propose the decoupled chunk attention (DCA) mechanism, which rearranges patches to preserve spatial information and processes attention in pipelined chunks. This approach significantly reduces off-chip memory access, achieving 87.1% sparsity and 56.9% computational reduction during attention calculation, with less than 1% performance degradation compared to vanilla attention.
- **Architectural Level:** We introduce a hybrid architecture combining SRAM-based CIM and RRAM storage, where static weights are stored on-chip using RRAM to eliminate off-chip memory access, and a fusion scheduling strategy is proposed to balance workloads and minimize intermediate on-chip memory access. This reduces memory accesses by 1.7×-7.4×, and improves energy efficiency by 1.8× compared to using vanilla attention on SRAM-CIM with off-chip DRAM.
- **Circuit Level:** We develop a reconfigurable computing circuit that combines bit-level and element-level parallelism to enhance hardware utilization and throughput across pyramidal ViT blocks. This work achieves a total acceleration ratio of 18.5×-217.1×.

II. Preliminaries and Motivation

A. ViT

In 2020, Dosovitskiy et al. [1] introduced the ViT architecture, which allows for capturing global interactions between image patches without the constraints of local connectivity. Despite their good performance, vanilla ViTs use only single-scale low-resolution features and suffer from high computation costs on dense prediction tasks, such as semantic segmentation and depth estimation. To address these limitations, Wang et al. [8] leveraged the strengths of both CNNs and Transformers by substituting uniform encoder blocks with a progressively shrinking pyramid structure, thereby reducing the computational cost of large feature maps and enhancing the performance of many downstream CV tasks. Recent research on ViTs has increasingly focused on utilizing sparse techniques to further reduce the complexity of self-attention computations. Zheng et al. [9] collapsed the input or output dimensions based on the sparsity and skipped the zero-value attention map. Liu et al. [10] sparsely preserved the semantically relevant patches and condensed irrelevant ones into a representative patch.

B. CIM

Data movement consumes the majority of time and energy in the inference process, therefore reducing data movement is crucial for efficiently deploying Transformers on edge devices [6]. The CIM paradigm, which integrates computation within the memory itself, offers an energy-efficient and high-speed computing architecture well-suited for Transformer processing. Recent CIM-based Transformer accelerators predominantly employ either analog RRAM-CIM or digital SRAM-CIM architectures [11], [12]. Analog RRAM-CIM provides high-density, nonvolatile storage and facilitates matrix multiplications via inherent physical properties. However, RRAM's intrinsic stochasticity and expensive programming overhead may render it unsuitable for dynamically generated, high-precision attention computations. Conversely, digital SRAM-CIM architecture incurs minimal memory write overhead and supports lossless computations, ensuring efficient and high-accuracy Transformer processing. Nevertheless, the volatile nature of SRAM necessitates loading weights from off-chip memory, resulting in a slow wakeup-to-response time for edge devices. Moreover, SRAM cells

require more area compared to RRAM cells, limiting their memory capacity.

C. Motivation

Although sparse attention and the utilization of CIM hardware architectures have reduced data movement and decreased the time-energy overheads associated with self-attention. However, the computation of self-attention still relies on the entire sequence, failing to decouple the attention size from the sequence length, and continues to face scalability issues in pixel-level dense predictions with longer contexts. As illustrated in the left part of Fig. 1, when the sequence length exceeds 1024, the memory usage of the attention block surpasses the typical on-chip storage capacity of CIM systems, resulting in frequent reads and writes to off-chip memory. Moreover, with each quadrupling of the sequence length, the frequency of accesses to the off-chip storage increases by an order of magnitude. Additionally, the pyramid-like architecture of modern ViT variants leads to an unbalanced workload. As shown in the right part of Fig. 1, the computational demands of each block vary due to differences in block size and spatial reduction mechanisms. This variability renders the fixed data flow and scheduling in the NLP transformer inefficient, leading to suboptimal hardware utilization. To the best of our knowledge, no existing work has simultaneously addressed these issues. Therefore, a novel solution is required to deploy ViTs for dense predictions on edge devices efficiently.

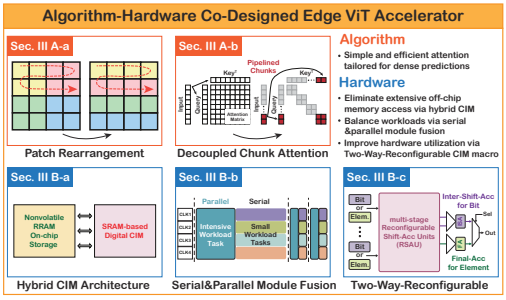


Fig. 2. Overview of the co-designed edge ViT accelerator.

III. Algorithm-Hardware Co-Designed Edge ViT Accelerator

Fig. 2 depicts the overview of the proposed algorithm-hardware co-designed edge ViT accelerator. Algorithmically, we propose a simple and efficient DCA mechanism that computes attention scores using fixed-size chunks in a pipelined manner, thereby minimizing the frequent access of query (Q), key (K), and value (V) elements. Hardware-wise, we introduce a hybrid architecture that combines SRAM-based CIM with nonvolatile on-chip RRAM storage to eliminate extensive off-chip memory access, with a fusion scheduling to minimize intermediate on-chip memory access for the varied workload ratios in each pyramid block. Further, we design reconfigurable CIM computing circuits that combine bit-level and element-level parallelism to improve resource utilization.

A. DCA Mechanism

a) Patch Rearrangement: In contrast to the long-range dependencies between tokens typically observed in NLP tasks, CV tasks generally exhibit stronger correlations between adjacent patches [13]. However, the sequential flattening adopted from NLP Transformers overlooks the two-dimensional spatial characteristics of images, causing adjacent patches to be scattered across widely spaced locations in the attention matrix (Fig. 3, upper part). Although the attention mechanism remains effective in capturing these long-range dependencies, the increased intervals necessitate the storage of longer patch sequences during attention computation, leading to costly off-chip memory accesses. To preserve the spatial information of adjacent patches, we divide the patch embeddings $\in \mathbb{R}^{H\times W}$ into windows $\in \mathbb{R}^{\frac{H}{4}\times\frac{W}{4}}$ and rearrange the patches akin to sequential flattening. This rearrangement effectively reduces the attention distance between adjacent patches, resulting in the clustering of high attention scores and forming a chunk-like attention pattern (Fig. 3, lower part). Therefore, we can still obtain sufficient information from the image even if we focus solely on the attention values within these chunks and ignore interactions with distant patches.

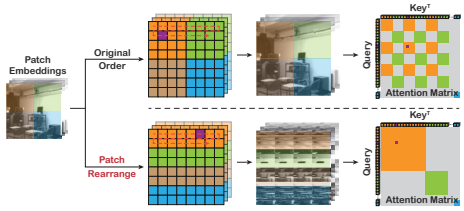


Fig. 3. Patch rearrangement diagram.

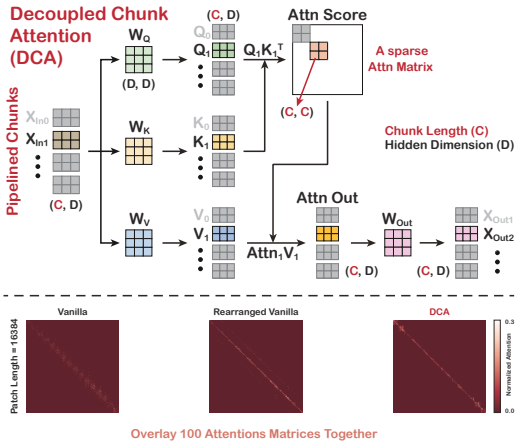


Fig. 4. Flowchart of the DCA. The bottom section displays the attention matrices of three different methods: vanilla attention (left), rearranged vanilla attention (middle), and the proposed decoupled chunk attention (right).

b) Hardware-friendly Chunk Attention: Inspired by the FLASHATTENTION, we propose a DCA algorithm tailored for dense predictions at the edge (Fig. 4, upper part), where the Q, K, and V matrices are partitioned into small chunks based on the on-chip memory capacity (e.g., a chunk length of 1024 corresponds to an on-chip

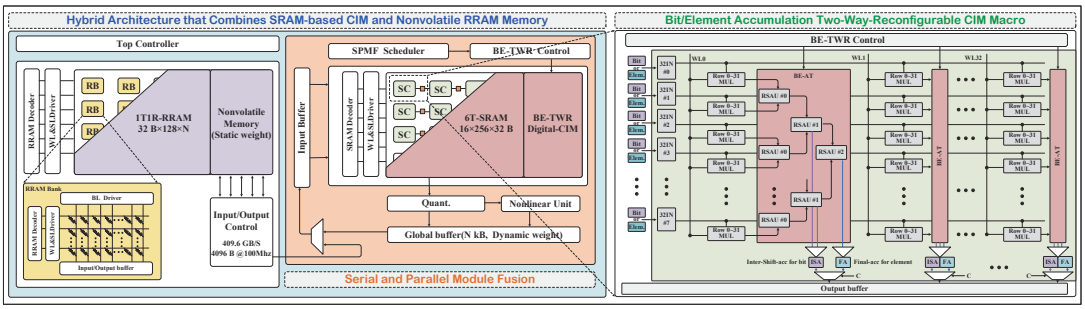


Fig. 5. Architecture overview of the proposed hybrid CIM ViT accelerator.

memory capacity of 128 KB), and the attention output is computed based on these chunks. Thanks to the pre-rearranged patches that concentrate important and relevant information in diagonal chunks, we avoid traversing the entire K and V matrices during the attention computation, in contrast to the FLASHATTENTION. Simultaneously, the pipelined architecture of the attention calculation minimizes the need for expensive off-chip memory accesses to store the complete input and intermediate attention results, thereby significantly reducing both the time and energy overhead associated with the attention computation.

To demonstrate the efficacy of our method, we visualize the overlay attention matrices of 100 randomly selected images, comparing the vanilla attention pattern with our DCA approach (Fig. 4, lower part). In the vanilla attention, bright spots indicating high correlation are scattered throughout the attention map. After patch rearrangement, most of these spots are concentrated on the diagonal strip, exhibiting an attentional pattern similar to our DCA method.

B. RRAM-storage SRAM-CIM Hybrid Hardware Architecture

a) Overall Hardware Architecture: As shown in Fig. 5, the proposed hybrid architecture consists of a top controller, 4 MByte 1T1R RRAM banks for static weights storage, 16 SRAM CIM (SC) macros supporting bit/element two-way reconfiguration, a non-linear unit, a small global buffer for dynamic weights, and a configurable serial and parallel module fusion (SPMF) scheduler. The high-density RRAM bank stores the static weights of each ViT block, and the reconfigurable SRAM-CIM macro efficiently performs matrix-matrix multiplications in DCA. The hybrid CIM architecture eliminates off-chip access through two means: first, the high-density RRAM banks allow for full on-chip storage of the lightweight Segformer-B0 model (3.4 MB) [7]; second, the proposed SPMF scheduling method eliminates off-chip access for dynamic weights and intermediate results.

The SPMF scheduler addresses the unbalanced workloads of the pyramidal ViTs depicted in Fig. 1. By decoupling the unbalanced workload in each ViT block using a combination of serial computation and parallel module fusion, the SPMF scheduler reduces intermediate memory access, ensuring efficient processing across the Segformer model. Additionally, we propose a bit/element two-way-reconfigurable CIM (BE-TWR-CIM) adder tree in the CIM macros to efficiently process matrix multiplications of different sizes (32, 64, 256, 1024, etc.). The BE-TWR controller can configure the adder tree into a high-input-parallelism single-bit accumulation mode for large-size matrix multiplication or a low-input-parallelism multi-bit accumulation mode for small-size matrix multiplication, thereby maximizing computation resource utilization.

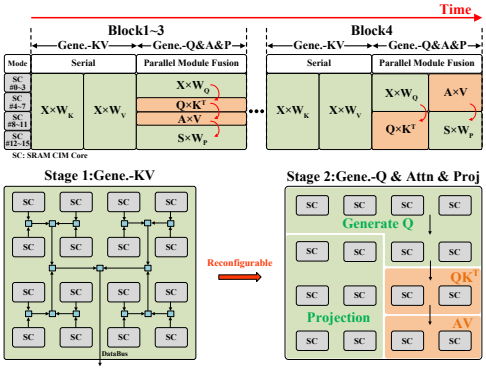


Fig. 6. Serial and parallel module fusion.

b) Serial and Parallel Module Fusion Scheduling: The fundamental principle of the SPMF scheduling method is to optimize computation resource utilization while minimizing memory access and storage requirements. As depicted in Fig. 4, the parallel execution of several operators (e.g., $Q \cdot K^T$ and $A \cdot V$) can eliminate intermediate memory access and storage space through direct data forwarding. However, the available computation resources may not suffice to support the parallel execution of all pipelineable operators. Consequently, the SPMF scheduling method is employed to determine the optimal parallel and serial workflows within each block, as illustrated in Fig. 6. In Blocks 1 to 3, the matrix dimensions of K^T and V are relatively small, allowing W_Q and W_p to be divided into smaller blocks without data dependency. This enables the full pipelining of the four operators ($X \cdot W_Q$, $Q \cdot K^T$, $A \cdot V$, and $S \cdot W_p$), thereby conserving memory access and storage space for the intermediate matrices Q, A, and S. In Block 4, the increased matrix dimensions of K^T and V preclude their mapping onto the limited CIM computation resources in a fully pipelined manner without

data dependency. Consequently, the pipeline is segmented into two sequential stages. The first stage involves the generation of large K and V matrices that cannot be efficiently integrated into a pipelined structure with adjacent operators, thereby executing serially. A reconfiguration schematic of the 16 SC macros between serial generation mode and parallel attention and projection mode is also provided in Fig. 6.

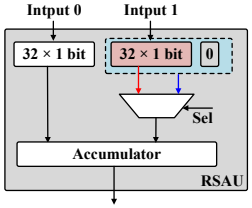


Fig. 7. Bit/element two-way-reconfigurable CIM adder tree.

c) Bit/element Two-Way-Reconfigurable CIM Macro: Each of the 16 SRAM-CIM macros comprises 256×256 SRAM cells and their associated peripherals. Each column of 256 SRAM cells is connected to a bit/element two-way reconfigurable adder tree (BE-AT), which is constructed from multi-stage reconfigurable shift accumulation units (RSAUs), as depicted in Fig. 7. In high-input-parallelism single-bit (HP-SB) accumulation mode, configured by the BE-TWR controller for large-scale matrix multiplication (e.g., 256×256×32, 1 bit), the RSAU utilizes a 2-1 multiplexer to load Input1 (32×1 bit) into the accumulator directly. By leveraging multi-stage direct accumulation, the RSAU efficiently performs single-bit accumulation for large-size matrix multiplications. In low-input-parallelism multi-bit (LP-MB) accumulation mode, also configured by the BE-TWR controller for smaller-scale matrix multiplications (e.g., 32×32×32, 4 bit), the RSAU employs the 2-1 multiplexer to load the shifted Input1 into the Accumulator. Through multi-stage shift accumulation, the BE-AT efficiently performs multi-bit accumulation for smaller-scale matrix multiplications.

IV. Evaluation

A. Experiment Setup

a) Networks and Dataset: To demonstrate the efficacy and generality of our design, we evaluate the performance of DCA using SegFormer-B0 as the backbone across three representative dense prediction tasks: object detection on the COCO dataset [14], semantic segmentation on the ADE20k dataset [15], and depth estimation on the NYUDV2 dataset [16]. SegFormer is a state-of-the-art pyramidal ViT framework that uses a series of mixed Transformer encoders to extract hierarchical feature representations across multiple scales. During the experiments, input images are resized to a resolution of 512×512 and embedded into 128×128 patches, corresponding to a sequence length of 16384. The hierarchical feature maps for the four encoder blocks in SegFormer-B0 are (128, 128, 32), (64, 64, 64), (32, 32, 160), and (16, 16, 256). Subsequently, a lightweight full multilayer perceptron decoder aggregates the information from different layers and generates pixel-wise predictions. The simple yet powerful SegFormer backbone is ideally suited for dense predictions on resource-constrained edge devices.

b) Hardware Baselines: To better illustrate the enhancements introduced by each proposed technique, we utilize three hardware baselines for comparative analysis. Baseline0 maintains the same SRAM CIM computational performance (excluding the BE-TWR adder tree) and uses off-chip DRAM for intermediate data storage while executing the original Segformer model. Baseline1 incorporates the DCA algorithm optimization based on Baseline0. OP0 refers to deploying DCA on the proposed RRAM-storage SRAM-CIM hybrid architecture, whereas OP1 further integrates the SPMF scheduling and bit/element two-way-reconfigurable accumulation technique. Additionally, a GPU baseline is evaluated with the RTX A30.

c) Hardware Components and Power/Area Breakdown: We implement the proposed hybrid CIM ViT accelerator using Verilog and synthesize the design using Synopsys Design Compiler under a 28 nm technology at 250 MHz. Table I shows the hardware components and power/area breakdown. It contains 16 SRAM CIM macros, each consisting of 256×256 SRAM cells and 256 BE-AT adder trees. The BE-AT unit only increases power consumption by 2.3% compared to a normal adder tree. The global SRAM buffer and fixed RRAM banks are set to 640 KB and 4 MB, respectively. The total area is 10.54 mm^2 with 834 mW peak power.

TABLE I
Hardware Components and Area/Power Breakdown

Module	Description	Area (mm^2)	Power (mW)
SRAM-CIM: cells	16×256×256	4.30	141
SRAM-CIM: BE-AT	16 × 256	1.98	414
Top control	-	0.32	3.64
Nonlinear unit	-	0.50	24.0
Global buffer	640 KB	1.28	10.1
RRAM	4 MB	2.16	241
Total	-	10.54	834

B. Experiment Results

a) DCA algorithm: Table II presents the performance results of our DCA design in comparison with the baseline SegFormer-B0 model across three different tasks. Our DCA approach achieves an 87.1% sparsity in the attention matrix and reduces attention FLOPs by 56.9% compared to the baseline model. Despite the simplicity of the DCA mechanism, it proves to be highly effective, achieving a mIoU of 37.1% on the ADE20K, a RMSE of 0.62 on the NYUDV2, and a box mAP of 36.6% and a mask mAP of 34.4% on the COCO, with an average performance degradation of only 0.7% compared to the conventional attention method. The visualization result of three tasks in Fig. 8 further demonstrates the high performance. For object recognition tasks, the DCA method shows results close to the baseline, albeit with slightly lower confidence scores for certain objects. For semantic segmentation tasks, the DCA method yields similar results consistent with the baseline, particularly for objects occupying large areas, with only a minor reduction in accuracy at the boundaries

between different classes. In the depth estimation task, the depth prediction results are comparable to the baseline prediction, with a small decrease in accuracy in delineating object boundaries.

b) Evaluation of Acceleration Ratio: Fig. 9(a) presents the acceleration ratio compared to the GPU and hardware baselines. Among the four ViT blocks in Segformer, barely using CIM shows 8.5×-16.1× acceleration. Utilizing the DCA Segformer algorithm optimization, the acceleration reaches 10.9×-20.9×, which varies depending on the algorithm-level computation compression on each ViT block. Note that the hardware resource cannot be fully utilized due to the unbalanced workload ratios among operators in each block and also the varying matrix dimensions. Utilizing the SPMF scheduling and BE-TWR CIM adder tree, the total acceleration ratio finally reaches 18.5×-217.1×.

TABLE II
PERFORMANCE COMPARISON BETWEEN VANILLA ATTENTION AND DCA ON DENSE PREDICTION TASKS

SegFormer B0 (3.4 M)	**Attn Sparsity** (% ↑[a])	**Attn FLOPs** (M ↓)	**COCO** (mAP ↑)	**ADE20k** (mIoU ↑)	**NYUDV2** (RMSE ↓)	**Average Loss** (% ↓)
Vanilla Attention	/	1.938	36.9 (Box) 34.7 (Mask)	37.4	0.618	/
DCA	87.1	0.836	36.6 (Box) 34.4 (Mask)	37.1	0.62	0.7

[a]Arrow direction indicates a better result.

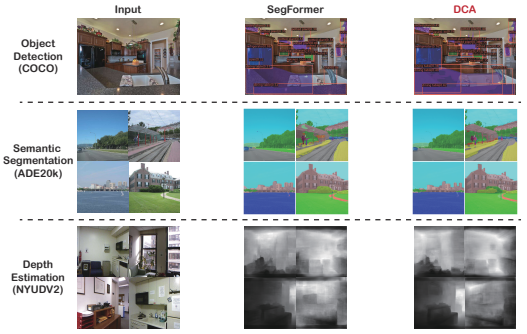


Fig. 8. Dense prediction results.

c) Evaluation of Memory Access: Fig. 9(b) shows that the proposed optimizations, OP1 and OP2, significantly reduce both on-chip and off-chip memory access compared to baseline0 and baseline1. For on-chip memory access, this work integrates SPMF scheduling and the BE-TWR accumulation techniques to reduce memory access by 1.7×-7.4× across four blocks compared to Baseline0. For off-chip memory access, this work utilizes the DCA mechanism and the heterogeneous RRAM-storage SRAM-CIM architecture to reduce memory access across four blocks to zero compared to Baseline0.

d) Evaluation of Performance and Energy Efficiency: Table III presents the comparison with the state-of-the-art ViT accelerators. The proposed hybrid CIM ViT accelerator achieves 1.6× and 1.8× energy efficiency improvement compared to the recent TranCIM [17] and P^{3}ViT [18], respectively. Utilizing the high-density 4 MB RRAM for fixed weight and 640 KB SRAM for intermediate data, this work avoids off-chip access compared to previous CIM chips. Furthermore, thanks to the DCA mechanism,

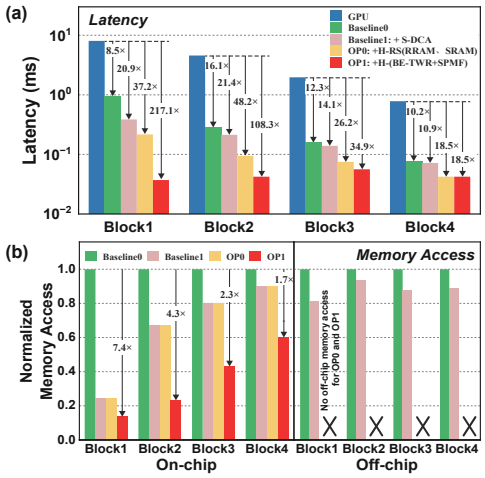


Fig. 9. Evaluation of (a) acceleration ratio and (b) memory access.

this work enables a pixel-level dense prediction algorithm with a maximum sequence length of 16384 within limited hardware resources.

TABLE III
COMPARISON TO THE STATE-OF-THE-ART VIT ACCELERATORS

	P^{3}ViT [18]	**TranCIM** [17]	**This Work**
MAC Implementation	Digital CIM	Digital CIM	Digital CIM
Technology (nm)	28 nm	28 nm	28 nm
On-chip memory	100 KB	192 KB	**4736 KB**
Need off-chip access	Yes	Yes	**No**
Frequency (MHz)	50-200 MHz	80-240 MHz	250 MHz
Die Area (mm^2)	2.0	6.83	9.46
Precision	INT4/8/16	INT8/16	INT8
Performance (TOPS)	1.6(INT8)	1.48(INT8)	6.55(INT8)
Energy Efficiency (TOPS/W)	23.2(INT8)	20.5(INT8)	**36.9(INT8)**
Prediction Level	Image-Level	Token-Level	**Pixel-Level**
Max Sequence Length	197	4096	**16384**

V. CONCLUSION

In this study, we implement an efficient yet powerful DCA on a hybrid CIM architecture, incorporating fusion scheduling and reconfigurable computation circuits to reduce both on- and off-chip memory accesses. Tailored for dense predictions, our proposed design supports a patch length of up to 16384, with a speedup of 18.5×-217.1×, a reduction in memory accesses of 1.7×-7.4×, and an improvement in energy efficiency of 1.8× under less than 1% performance loss across three representative pixel-level tasks. This ViT accelerator offers a promising solution for various dense prediction applications at the edge.

ACKNOWLEDGMENT

This work was supported in part by National Key Research and Development Program of China under Grant 2023YFB4402400; National Natural Science Foundation of China under Grants 92464201, U2341218, 92464203, 62374181, and 62204256; Shenzhen Science and Technology Innovation Commission (Grant No.SGDX20220530111405040); Beijing Natural Science Foundation (Grant No.Z210006); Hong Kong Research Grant Council (Grant Nos.27209621, 17205922, 17212923); RGC YCRG Grant no.C7003-24Y and RGC GRF Grant no.17205922; Joint Laboratory of Microelectronics (JLFS/E-601/24); This research was partially supported by ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK funding, Hong Kong SAR.

References

[1] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[2] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," in *ICCV*, 2021, pp. 12 179–12 188.

[3] Y. Chen, L. Zhang, J. Shang, Z. Zhang, T. Liu, S. Wang, and Y. Sun, "DHA: learning decoupled-head attention from transformer checkpoints via adaptive heads fusion," in *NIPS*, 2024.

[4] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022.

[5] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, "Flashattention: Fast and memory-efficient exact attention with io-awareness," *Adv. Neural Inform. Process. Syst.*, vol. 35, pp. 16 344–16 359, 2022.

[6] M. Zhou, W. Xu, J. Kang, and T. Rosing, "Transpim: A memory-based acceleration via software-hardware co-design for transformer," in *HPCA*, 2022, pp. 1071–1085.

[7] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers," *Adv. Neural Inform. Process. Syst.*, vol. 34, pp. 12 077–12 090, 2021.

[8] W. Wang, E. Xie, X. Li, D.-P. Fan, K. Song, D. Liang, T. Lu, P. Luo, and L. Shao, "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions," in *ICCV*, 2021, pp. 568–578.

[9] Q. Zheng, S. Li, Y. Wang, Z. Li, Y. Chen, and H. H. Li, "Accelerating sparse attention with a reconfigurable non-volatile processing-in-memory architecture," in *DAC*, 2023, pp. 1–6.

[10] T. Liu, X. Liu, S. Huang, L. Shi, Z. Xu, Y. Xin, Q. Yin, and X. Liu, "Sparse-tuning: Adapting vision transformers with efficient fine-tuning and inference," *arXiv preprint arXiv:2405.14700*, 2024.

[11] X. Yang, B. Yan, H. Li, and Y. Chen, "Retransformer: Reram-based processing-in-memory architecture for transformer acceleration," in *ICCAD*, 2020, pp. 1–9.

[12] F. Tu, Z. Wu, Y. Wang, W. Wu, L. Liu, Y. Hu, S. Wei, and S. Yin, "Multcim: Digital computing-in-memory-based multimodal transformer accelerator with attention-token-bit hybrid sparsity," *IEEE J. Solid-State Circuits*, 2023.

[13] A. Khan, Z. Rauf, A. Sohail, A. Rehman, H. Asif, A. Asif, and U. Farooq, "A survey of the vision transformers and its cnn-transformer based variants. arxiv 2023," *arXiv preprint arXiv:2305.09880*, 2023.

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *ECCV*, pp. 740–755.

[15] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *CVPR*, 2017, pp. 633–641.

[16] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*, 2012, pp. 746–760.

[17] F. Tu, Z. Wu, Y. Wang, L. Liang, L. Liu, Y. Ding, L. Liu, S. Wei, Y. Xie, and S. Yin, "Trancim: Full-digital bitline-transpose cim-based sparse transformer accelerator with pipeline/parallel reconfigurable modes," *IEEE J. Solid-State Circuits*, vol. 58, no. 6, pp. 1798–1809, 2022.

[18] X. Fu, Q. Ren, H. Wu, F. Xiang, Q. Luo, J. Yue, Y. Chen, and F. Zhang, "P^{3}vit: A cim-based high-utilization architecture with dynamic pruning and two-way ping-pong macro for vision transformer," *IEEE Trans. Circuits Syst. I Regul. Pap*, 2023.

LVM-MO: A Large Vision Model Pioneer on Full-Chip Mask Optimization

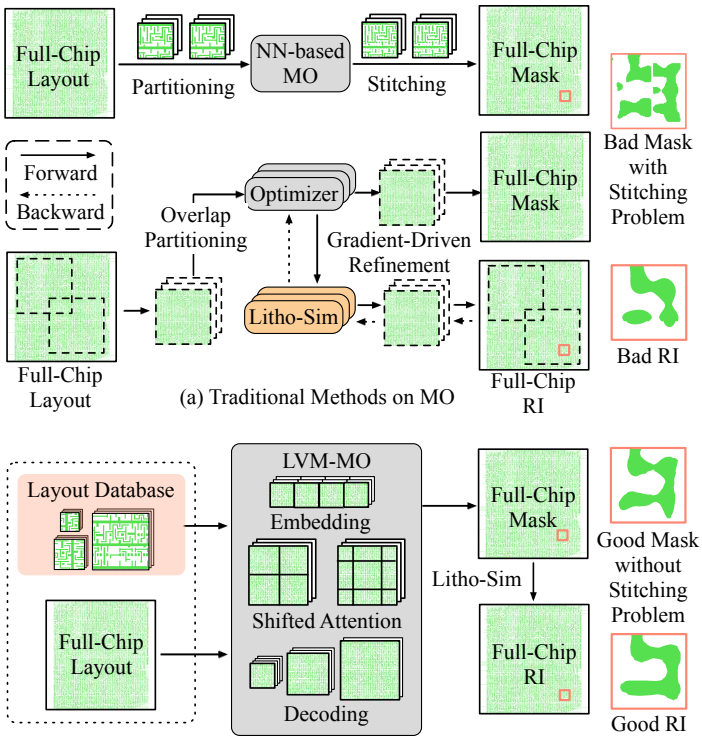
Yiwen Wu*[1] Yuyang Chen*[1] Shuo Yin[2] Nan Wang[1] Tao Wu[1] Xuming He[1] Hao Geng†[1] Jingyi Yu†[1]
[1]ShanghaiTech University [2]Dept. CSE, CUHK

***Abstract*—Moving toward the post-Moore era, full-chip mask optimization (MO) has become a pivotal step for semiconductor designers and manufacturers in extending current resolution enhancement techniques. The majority of recent research efforts have focused on clip-level restoration, employing a divide-and-conquer approach to mitigate the impacts of optical proximity and process bias across entire chips. Nevertheless, when confronted with industrial full-chip mask optimization challenges, these works exhibit limited correction capabilities, struggle with generalization, and are time-inefficient. In this paper, we propose a novel full-chip mask optimization paradigm based on a massive lithography data-driven large vision model. Our approach features a foundation layout feature extractor, which is aware of the mutual influence of polygons in long-range pattern perception as well as optical physics and chemical characteristics of lithography, matters. Compared with state-of-the-art (SOTA) works, our work demonstrates significant advantages in terms of resolution fidelity, correction speed, and the ability to handle full-chip scale layouts.**

I. Introduction

As lithography advances into the low-k1 regime, mask optimization (MO) has become essential for achieving production-ready patterning solutions. While full-chip scale MO was identified over a decade ago as a critical development in lithography, its practical implementation has remained largely unrealized. This stagnation is attributed to the intense computational demands and high complexity associated with full-chip MO. In response, considerable efforts [1]–[5] have been made to tackle this issue. Neural network (NN)-based approaches to optical proximity correction (OPC) and inverse lithography technology (ILT) primarily focusing on smaller layout clips rather than full-chip scales have emerged as two viable solutions. Current NN-based OPC techniques often apply generic neural network architectures without fully leveraging the specific characteristics of optical proximity effects and consequently struggle to adapt to varying layout sizes, let alone scale to full-chip implementations. ILT methods, in contrast, aim to conduct global optimizations and therefore are potentially capable of handling layouts of various sizes. However, such optimizations are computationally expensive due to the inefficiency of pixel-based mask optimization and are not readily applicable to full-chip MO.

We observe that lithography is fundamentally limited by optical physics and resist imaging, necessitating full-chip MO methods that integrate global layout patterns, circuit functionality, lithography settings, and imaging factors including optical diameter and coherence length. Instead of directly designing an optimization scheme such as NN or ILT, we set out to first design **a foundational layout feature extractor** that is capable of capturing polygon positions, long-range pattern interactions, light diffraction effects, etc. This extractor adapts to varying layout sizes, enabling consistent integration of full-chip-scale inputs into the correction process, a task that is too time-intensive for traditional ILT at such scalability. Our approach is inspired by recent advances in vision foundation models such as Vision Transformers [6] (ViT) and DINOv2 [7] that offer a robust framework for building such extractors, leveraging their proven ability to handle pattern comprehension and long-term dependencies efficiently. Once trained, they provide versatile, high-quality visual feature representations that can be directly adapted to diverse downstream tasks like classification, detection [8], segmentation [9], and retrieval without extensive retraining.



(b) LVM-MO on Full-Chip Scale by Zero-Shot

Fig. 1 Illustration of our LVM-MO on full-chip design empowered by millions of layout/mask/resist data vs. the traditional clip-focus and divide-and-conquer full-chip MO methods. The previous works suffer from the mask stitching issue and low-fidelity resulting resist images.

In this paper, we introduce LVM-MO, a large vision model tailored for full-chip mask optimization (MO), as illustrated in Fig. 1. By integrating layout- and circuit-specific designs into a transformer-based, autoencoder framework, we develop a customized layout feature extractor. Lithography simulation back-propagation further embeds exposure and development insights into the feature extraction process. For full-chip inputs, LVM-MO employs a larger patch embedding module to handle low polygon density and wide pixel relationships in layout images. Additionally, we impose the optical coherence length constraint to enable the use of shifted window attention, simplifying feature extraction and focusing on relative pattern positions. This combined approach effectively captures both local and global layout information, avoiding the stitching issues found in traditional divide-and-conquer methods. With our layout-aware patch embedding and

*Both authors contributed equally to this research
†Corresponding author

litho-aware attention mechanisms, the feature extractor supports efficient and precise full-chip MO. Once trained, LVM-MO can directly convert the layout features into mask outputs using a simple decoder, eliminating the need for iterative division operations.

A key highlight of our work is that large model-based mask optimizers like our LVM-MO can incorporate optical physics, resist chemistry, and lithography principles through extensive training with massive datasets. To achieve this, we generate 5 million of layout samples from open-source circuit designs and train the model in three stages: (1) layout geometry understanding, (2) lithography basics learning via ILT, and (3) generalization for varying layout scales. It is critical to note that, compared to full-chip ILT methods, which require significant computation time for each layout, our approach shifts the time-intensive process to model training. Once trained, the model operates solely in the inference phase for mask optimization, making the overall process far more efficient. This "left-shift" in time consumption to the training phase ensures that inference, not iterative computation, dominates mask optimization. For example, LVM-MO directly outputs a full-chip mask image in seconds, bypassing traditional computational bottlenecks. Compared to state-of-the-art (SOTA) full-chip ILT methods, LVM-MO also achieves significant improvement in fidelity while enabling the inference in seconds, a significant advancement in full-chip MO.

The main contributions of this paper are as follows:

- To the best of our knowledge, this is the first work of a large vision model (LVM) tailored for lithography and layout information extraction, integrating these features into a decoder for downstream mask optimization.
- We meticulously craft the LVM architecture around lithography parameters, employing a shifted litho-window transformer for multidimensional layout analysis.
- Using open-source circuit structures, we build a million-scale layout database mirroring practical industrial settings, enabling a scalable and efficient inference process.
- Our approach markedly surpasses state-of-the-art works, delivering over **50%** boost in fidelity and achieving **1000**$\times$ speedup.

II. Preliminaries

In this section, we introduce the bases of lithography and the vision transformer. The necessary definitions of evaluation metrics and the problem formulation are provided.

A. Lithography Basis

Forward lithography projects a mask pattern $M \in \mathbb{R}^{H\times W}$ to a wafer image $Z \in \mathbb{R}^{H\times W}$ in two processes. The first optical process transfers M to aerial image intensity I, which can be approximated by the Hopkins' model [10] as shown in Equation (1):

$$I(x,y) \approx \sum_{i=1}^{N_i} \sigma_i |M(x,y) \otimes h_i(x,y)|^2, \tag{1}$$

where $\otimes$ denotes the convolution operation, h_i is the ith optical kernel, σ_i is the corresponding weight of the system,N_i denotes the total number of kernels used for simulation, and (x,y) is the notation of M. The second photo-resist formation process converts the aerial image intensity I to the printed wafer image Z. A compact model using a constant intensity threshold I_{th} is applied:

$$Z(x,y) = \begin{cases} 1, & \text{if } I(x,y) > I_{th}, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

$f(\cdot)$ can be used to represent the whole computing above from mask image M to wafer image Z, as shown in:

$$f(M) = Z. \tag{3}$$

The inverse problem can be regarded as finding the optimal mask M^* given the target wafer image Z_t. After the forward process $f(\cdot)$, the wafer image Z should be aligned with Z_t as closely as possible. Thus, mask optimization can be defined as:

$$M^* = f^{-1}(Z_t). \tag{4}$$

Optical coherence, characterized by the coherence length and range, is a key factor in lithography resolution and fidelity. Coherence length reflects the temporal correlation of light waves and depends on the source's spectral bandwidth, with shorter lengths improving imaging stability. Coherence range, influenced by the source's angular divergence, defines spatial wavefront correlation, ensuring a balance between resolution and contrast. Optical diameter, determined by the illumination aperture size, governs the angular divergence, influencing coherence and feature resolution. These parameters also impact the Optical Transfer Function (OTF), attenuating higher spatial frequencies. The maximum period in MO is typically $2\mu m$. Periods beyond this contribute minimal high-frequency content to the OTF and exhibit negligible proximity effects, ensuring efficient correction of critical lithographic distortions. Building on the characteristics of these parameters in lithography, we specifically designed network architecture parameters to make the network more efficient in learning layout pattern correlations and lithographic characteristics. This part will be discussed in more detail in the Section III-B.

B. Vision Transformer

The Vision Transformer (ViT) is a deep learning architecture adapted from Transformer [11], originally used in natural language processing to handle visual data. Unlike Convolution Neural Networks (CNNs), which rely on convolution layers to extract local features progressively across layers, the ViT splits an input image into fixed-size patches, which are then linearly embedded and processed as tokens. This enables the model to capture long-range dependencies in the image through self-attention mechanisms, offering flexibility in capturing global context without a localized receptive field. Some versions of vision transformer, such as Swin Transformer [12], can adapt to different input sizes through patch resizing and hierarchical token structures, allowing the model to process images of varying resolutions more effectively. This flexibility and capability to handle larger receptive fields give Vision Transformers an advantage in handling complex, global features in high-resolution images.

C. Evaluation Metrics and Problem Formulation

This paper uses squared L_2 error, process variation band (PVB), and edge placement error (EPE) as our evaluation metrics. The optimization problem is also defined.

Definition 1 (Squared L_2 Loss)**.** *L_2 calculates the difference between the nominal condition wafer image Z_{norm} and the target image Z_t, denoted as:*

$$L_2 = \|Z_{norm} - Z_t\|_2^2. \tag{5}$$

Definition 2 (PV Band)**.** *Process variation band (PVB) evaluates the robustness of the mask between the maximum and minimum process conditions, denoted as $\|Z_{out} - Z_{in}\|_2^2$. Z_{out} is the printed image under maximum process condition with focus and $+3\%$ dose, Z_{in} is the printed image under minimum process condition with defocus and -3% dose.*

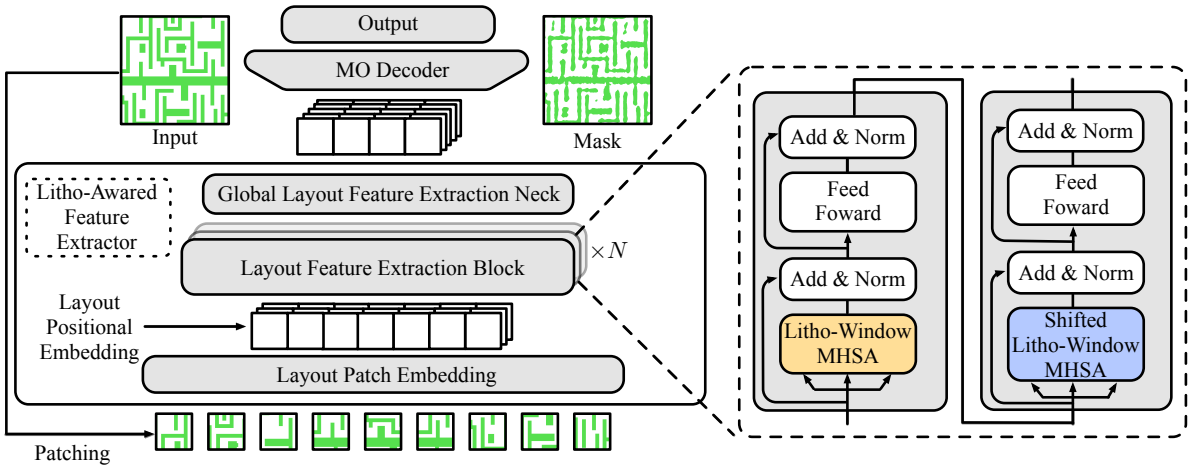


Fig. 2 Architecture of our LVM-MO.

Definition 3 (Edge Placement Error)**.** *Edge placement error (EPE) assesses the distortion of the wafer/resist image relative to the target. Measurement points are uniformly placed along the shape's horizontal and vertical contours on the target image. Unlike the ICCAD 2013 contest, which counted EPE violations above a threshold, our work adopts calculating the average value across all points, in line with industry standards.*

Problem 1 (Full-Chip Mask Optimization)**.** *Given the full-chip scale layout Z_{ft}, our objective is to efficiently obtain the optimized full-chip scale mask M_f such that the resulting resist pattern Z_f after the lithography processes closely matches Z_{ft}, while minimizing the squared L_2 error, EPE, PV band area.*

III. Proposed Framework

In this section, we introduce the overall architecture of the proposed framework. First, Section III-A outlines an overview of our LVM-MO structure, a litho-aware autoencoder, emphasizing its feature-extracting ability. Next, we explain the usage of litho-aware shifted window attention applied with our model in Section III-B. This design considers lithography parameters like optical diameter and coherence length, reducing attention computation while preserving critical lithography information between adjacent patches. Finally, we describe the training-inference flow of our proposed model in Section III-C. Our training involves three stages: layout polygon geometry understanding, lithography basics learning using ILT, and generalization calibration of small scale to full-chip size. Once adequately trained, LVM-MO rapidly generates full-chip scale masks during inference within seconds.

A. LVM-MO Architecture

An autoencoder is an artificial neural network designed to learn and efficiently encode input data, primarily for tasks like dimensionality reduction, denoising, or feature extraction. We customized a litho-aware autoencoder to extract features of layout geometry and lithography basics, as shown in Fig. 2. Our litho-aware autoencoder consists of two main parts: litho-aware feature extractor and MO-decoder, which will be introduced in Section III-A and Section III-A.

Litho-Aware Feature Extractor As shown in Fig. 2, our litho-aware feature extractor consists of layout patch embedding, layout positional embedding, layout feature extraction block, and global layout feature extraction neck.

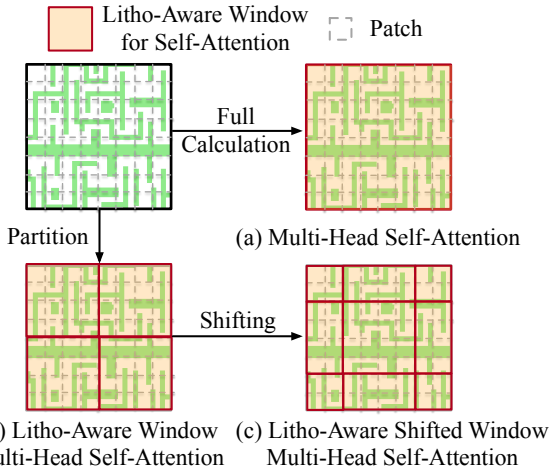


Fig. 3 Comparison of litho-aware shifted window attention concerning litho-aware window attention and ordinary attention.

A layout image $X \in \mathbb{R}^{H\times W\times C}$ first goes through a patch embedding process. For patch size of P, and a desired latent feature dimension D_1, X is transferred into a latent space dimension representation $L_1 \in \mathbb{R}^{N\times D_1}$, where $N = \frac{H}{P} \times \frac{W}{P}$. The feature dimensionality increases in this process. This is essential to partition the image and raise its dimensions for subsequent processing. Then, layout positional embedding infuses the latent feature L with position information $E_{pos} \in \mathbb{R}^{N\times D_1}$. This operation doesn't change the whole dimension of latent feature L_1. It should be noticed that E_{pos} is learnable during the training process, and for a varied layout, the E_{pos} can be interpolated to a suitable dimension. The latent feature is then fed into layout feature extraction blocks. Since the layout is divided into patches before input into layout feature extraction blocks, our model does not limit the length of patches. Within each block, litho-aware shifted window and window multi-head self-attention are used to extract features comprehensively from the input patches. It should be noted that the dimension remains unchanged after the latent feature goes through these blocks. Detailed discussion on the attention mechanism will be provided in Section III-B. Finally, L_1 passes through the global layout feature extraction neck and transfers to $L_2 \in \mathbb{R}^{N\times D_2}$, consists of two convolutional layers to integrate global information further and compress the patch information. D_2 is the desired global feature dimension. Both the geometrical and lithographical features are learned in the layout feature extractor.

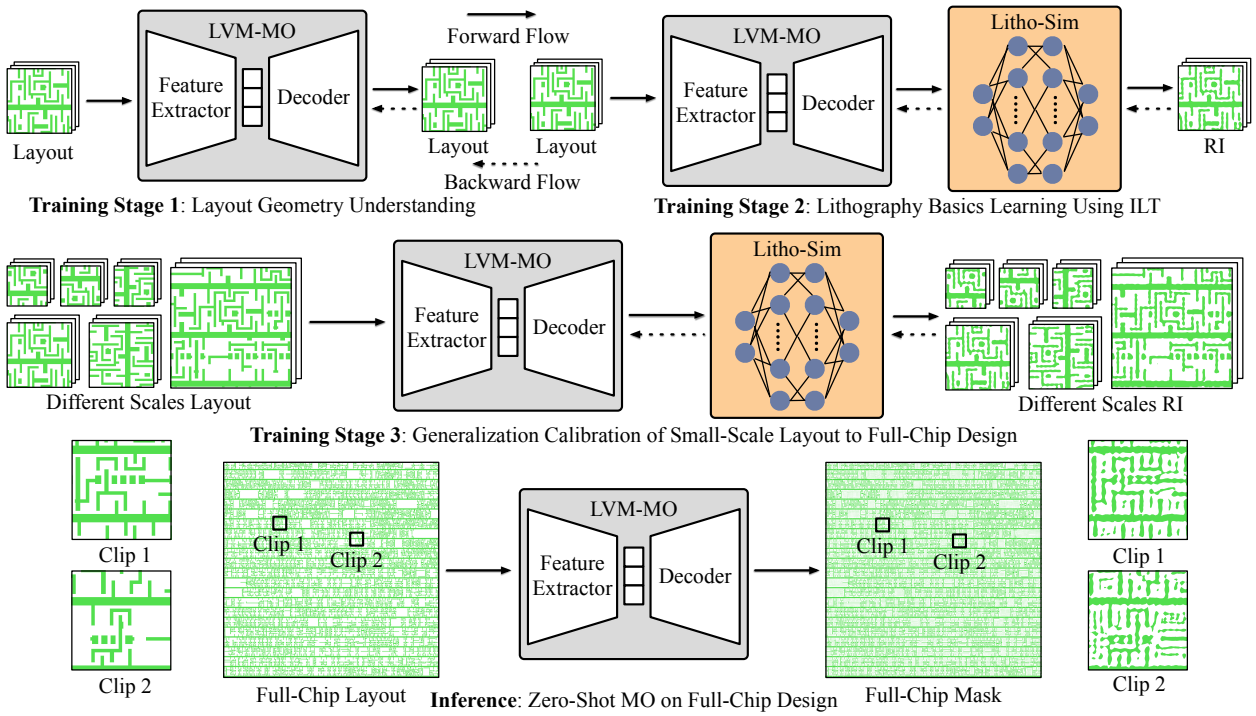


Fig. 4 Training & Inference flow of our LVM-MO.

Mask Optimization Decoder After passing through the layout feature extractor, the layout becomes a vector of dimensions $N \times D_2$. We use a transposed convolutional decoder that efficiently reconstructs the lithography information extracted by the encoder into a mask image while effectively recovering the fine details, as illustrated in Fig. 2.

B. Litho-Aware Window and Shifted Window Multi-Head Self-Attention

Technique Insights This part elaborates on the vision transformer architecture. While the vision transformer can support images of various sizes, naive ViT networks use global attention, which, while advantageous in enabling a large exchange of information between global patches, results in massive computational costs for high-resolution layout image inputs. Experiments show that for a $30\mu m$ side-length image (1 nm = 1 pixel), the GPU memory required reaches approximately 300 GB within just one transformer block. Fortunately, as introduced in Section II-A, the affected range of light diffraction in mask optimization is approximately $2\mu m$. This means we only need to compute feature correlations in a limited range. Inspired by the above nature of mask optimization, we adopt litho-aware window and shifted window multi-head self-attention, limiting attention computation to a single window with respect to light diffraction. The illustration is shown in Fig. 3.

Multi-Head Self-Attention For layout feature $X \in \mathbb{R}^{N\times D}$, queues $Q = X \cdot W_Q$, keys $K = X \cdot W_K$, and values $V = X \cdot W_V$, $W_Q, W_K, W_V \in \mathbb{R}^{D\times D_k}$. D_k is a specified dimension. The self-attention Z_h and corresponding multi-head self-attention Z_{mhsa} with H heads are calculated as:

$$Z_h = \text{Softmax}(\frac{QK^T}{\sqrt{D_k}}) \cdot V, h = 1, 2, ..., H, \tag{6}$$

$$Z_{mhsa} = \text{Concat}(Z_1, Z_2, ..., Z_H) \cdot W_O, \tag{7}$$

where $Z_{mhsa} \in \mathbb{R}^{N\times D}$, $Z_h \in \mathbb{R}^{N\times \frac{D_k}{H}}$, $W_O \in \mathbb{R}^{D_k\times D}$ is the projection weight. The computational complexity is $\mathcal{O}(N^2D)$, which gains fast as N gets larger.

Litho-Aware Window and Shifted Window Multi-Head Self-Attention The litho-aware window multi-head self-attention limits attention computing in the litho-aware windows. For litho-aware window size of $W \times W$, there exists $k = \lceil \frac{N}{W^2} \rceil$ number of windows. Self-attention within each litho-aware window X^i_{window} is calculated independently and concatenated together:

$$Z^i_{\text{window}} = mhsa(X^i_{\text{window}}), i = 1, ..., k, \tag{8}$$

$$Z_{\text{window}-mhsa} = \text{Concat}(Z_1, ..., Z_k), \tag{9}$$

where $Z^i_{\text{window}} \in \mathbb{R}^{W^2\times D}$, $Z_{\text{window}-mhsa} \in \mathbb{R}^{N\times D}$. The computational complexity of window attention is $\mathcal{O}(NWD)$, which is affected by the window size W. When $W \ll \sqrt{N}$, the computing is greatly reduced.

Window Attention effectively addresses the GPU memory limitation but reduces inter-window interaction, which can result in stitching issues at the feature level. We employ the litho-aware shifted window multi-head self-attention approach to mitigate this, as illustrated in Fig. 3. In even-numbered layout feature extraction blocks, the windows are shifted by half the window size, moving points that were originally at the edge to the center of the window. This significantly reduces the local effects introduced by windowing. Moreover, given that each point's lithography effect typically extends over a $2\mu m$ distance, this approach comprehensively captures the optical proximity effect. For our specific task, litho-aware shifted window multi-head self-attention can be approximated as multi-head self-attention, which calculates global layout attention.

C. Training & Inference Flow

Training Flow The training process is a crucial part of our framework. For the proposed LVM-MO model, training directly on datasets

TABLE I Comparison with state of the arts.

Bench	Area(μm^2)	EGAN-OPC [13] + DC			Neural-ILT [14] + DC			Multi-ILT [15] + DC			Ful-ILT [5]			Ours		
		L_2	PVB	EPE	L_2	PVB	EPE	L_2	PVB	EPE	L_2	PVB	EPE	L_2	PVB	EPE
ICCAD13-S	2.0×2.0	41387.2	48917.8	7.6	37515.3	50963.9	7.9	28704.6	42132.0	3.4	38402.8	48673.0	9.2	25933.5	46595.7	2.8
ICCAD13-L	2.0×2.0	89381.4	11230.4	9.3	71570.7	108161.8	7.5	51028.2	88022.1	2.9	61643.0	106551.0	8.5	48547.4	96877.7	2.0
CHIP-L1	$\sim 2.0 \times 2.0$	316408.3	550175.3	10.2	273920.1	482547.6	7.8	326912.9	490897.3	4.2	289827.9	367969.3	8.1	168960.3	355424.4	3.6
CHIP-L2	$\sim 4.0 \times 4.0$	1217940.2	1718228.1	14.8	973280.2	1269664.5	12.2	898304.6	1124776.6	6.4	1304628.0	1062168.2	15.3	702371.1	1116672.3	5.1
CHIP-L3	$\sim 8.2 \times 8.1$	4145163.7	5303460.4	10.5	4038772.3	4876767.2	9.9	2970626.2	2809734.8	5.4	4262425.1	3765231.4	11.9	1983827.1	3179608.2	4.0
CHIP-L4	$\sim 16.4 \times 16.5$	17648155.9	19294115.7	11.2	16736065.0	18399437.1	9.2	9585152.2	10926350.3	5.7	13877534.8	12689778.6	12.8	7155456.3	11004928.3	4.4
CHIP-L5	$\sim 31.8 \times 30.7$	61365838.8	66454134.5	11.3	59821497.5	65938924.3	9.4	45410688.8	50972876.5	5.4	60826060.8	41903554.0	12.1	32347392.3	41923968.3	4.3
CHIP-L6	$\sim 61.4 \times 59.3$	252992234.3	354638847.0	11.5	245571621.0	284301498.5	9.2	167319584.5	186691222.8	5.8	236791188.5	158583664.3	12.8	153277888.5	166623940.5	4.4
CHIP-L7	$\sim 90.9 \times 90.8$	560892534.3	674771451.5	11.5	522921540.5	609585455.5	9.1	331406436.3	384556344.5	5.6	444236360.0	351000531.0	12.7	295354338.3	378805482.0	4.5
CHIP-L8	153.5×152.9	1282191579	1336770725	11.2	1216226387	1268302370	9.2	951155909	1097085108	5.2	1056216511	828287165	12.0	721514949	1008921116	4.3
Average on $4\mu m^2$ Area		273360.6	355519.4	11.5	252270.4	312550.9	9.5	198366.6	239216.1	5.5	252134.0	216539.0	12.2	**144126.5**	**215180.6**	**4.3**
Ratio		1.897	1.652	2.671	1.750	1.453	2.199	1.376	1.112	1.749	1.575	1.006	2.826	1	1	1

TABLE II Benchmarks Statistics.

Benchmarks	Training Set	Testing Set	Size/Layout (μm^2)	Lithography Simulator
ICCAD2013	20807	20	2.0×2.0	ICCAD2013
CHIP-L1	5061523	20	$\sim 2.0 \times 2.0$	Industry Simulator
CHIP-L2		20	$\sim 4.0 \times 4.0$	
CHIP-L3		20	$\sim 8.2 \times 8.1$	
CHIP-L4		20	$\sim 16.4 \times 16.5$	
CHIP-L5	-	10	$\sim 31.8 \times 30.7$	
CHIP-L6	-	4	$\sim 61.4 \times 59.3$	
CHIP-L7	-	4	$\sim 90.9 \times 90.8$	
CHIP-L8	-	1	153.5×152.9	

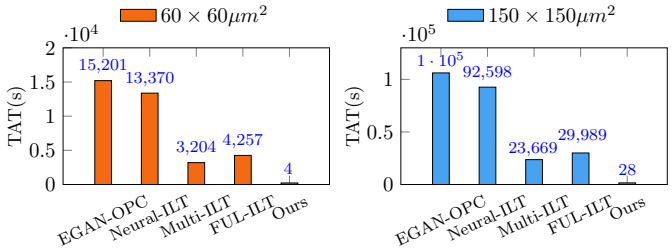


Fig. 5 Turn around time comparison with state of the arts.

ranging from small academic examples to full-chip data would lead to poor convergence. Therefore, we divide the training into three stages.

Stage 1: Layout Polygon Geometry Understanding In the first stage, we focus on reconstruction training, using the LVM-MO to reconstruct the target patterns, as shown in Fig. 4. The goal is for the network to learn to understand the spatial arrangement of patterns, which contains the geometry information. Given the original layout patterns $Z \in \mathbb{R}^{H \times W \times C}$, a reconstruction loss $L_{reconstruct}$is applied:

$$L_{reconstruct} = \frac{1}{HWC} \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{c=1}^{C} (Z_{h,w,c} - \hat{Z}_{h,w,c})^2. \quad (10)$$

$\hat{Z}$ is the reconstructed layout patterns.

Stage 2: Lithography Basics Learning Using ILT We incorporate inverse lithography technology (ILT) components in the second stage, shown in Fig. 4. The aerial image I obtained through ILT is compared to the target image Z_t using the $L_2 = \|I - Z_t\|_2^2$ loss function, and the gradient is back-propagated through the network. LVM-MO understands lithography basics at this stage. The decoder learns the inverse lithography optimization in this process.

Stage 3: Generalization Calibration from Small-Scale Layout to Full-Chip Design The approach in the first two stages is similar to traditional ILT. Still, further optimization is required for full-chip layout inputs. In this stage, we train the network using chips of varying sizes to enhance its generalization ability to a full-chip layout, illustrated in Fig. 4. Different scale layout patterns are applied as inputs to the LVM-MO, and MO in different layout scales is calibrated.

Inference Flow As shown in Fig. 4, after adequate training in geometry, lithography basics, and scale generalization on layout patterns, the LVM-MO is ready to make direct inferences on full-chip scale layout, called zero-shot. It should be noticed that besides full-chip design, different scales of layout patterns can be zero-shot with our LVM-MO.

IV. Experiments

A. Experimental Setup and Benchmark Details

We implement our work with the Pytorch-Geometric toolkit [16]. We train our model on the platform that possesses the Xeon Platinum 8358P CPU and 8× NVIDIA Tesla A100 (80GB) Graphics Cards with NV-Switch and test on 1× A100 Graphic Card.

The complete information of our database is shown in TABLE II. We use the academic ICCAD2013 dataset and our generated CHIP-L dataset from real design. For ICCAD2013, the training dataset is generated by data augmentation. ICCAD2013 simulator is applied to training to make testing fair. As for CHIP-L, both the training and testing sets are from some popular open-source digital designs like [17]–[19], which we extracted from GDSII files generated by commercial layout generation tool, Design Compiler and Innovus, and open-source FreePDK45 [20] process design kit. We use over 5 million layouts of different scales as our training datasets. Layer metal-1 is extracted as our layout. Limited by the memory of GPU, only scales not bigger than $16.4 \times 16.4 \mu m^2$ are used during training. It should be noted on real design training that both the state-of-the-art (SOTA) and our LVM-MO use the same industry-level simulator as [5].

For lithography simulator settings, the photoresist intensity threshold for lithography settings is set at 0.225, and sigmoid steepness is 50. The lithography wavelength is $193nm$ with a defocus range of $\pm 30nm$ and a dose range of $\pm 3\%$. The resolution of the mask and corresponding target pattern/on-wafer image are all $1nm$/pixel. For our model's configuration, LVM-MO applied a 128 patch size, 24 attention heads, 12 layout feature extraction layers, and 16 window size in latent space, corresponding to a $2um$ window size in real space. Our LVM-MO model consists of 2.7 **billion trainable parameters**.

We compared our LVM-MO with the following state of the arts on the two benchmarks to evaluate its performance.

- EGAN-OPC [13] + DC A generative adversarial network (GAN) on MO powered by target-mask mapping and ILT with divide-and-conquer method.
- Neural-ILT [14] + DC An end-to-end learning-based OPC framework using a single neural network with the divide-and-conquer

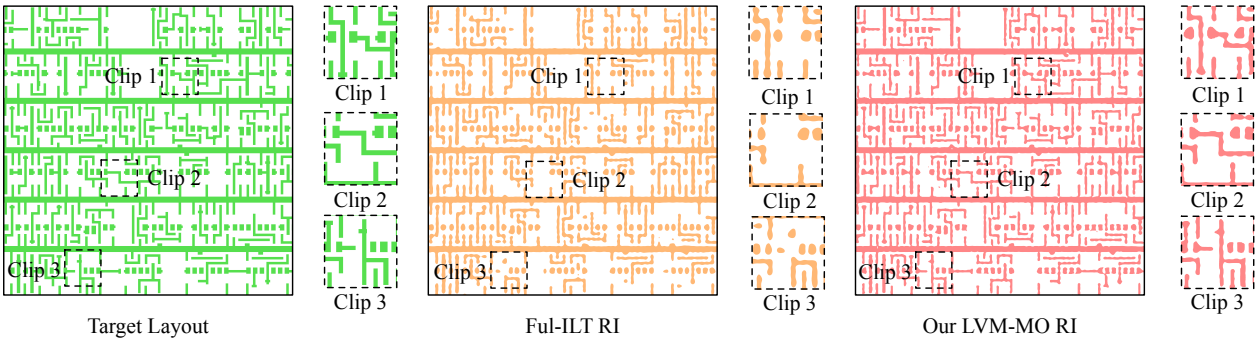


Fig. 6 Visualization on partial full-chip resist images of Ful-ILT and our LVM-MO.

method.

- Multi-ILT [15] + DC An efficient ILT framework leveraging multi-level resolution method with divide-and-conquer method.
- Ful-ILT [5] A full-chip ILT-based mask optimization flow enabled by divide-and-conquer method.

It's worth noting that we obtain the source code of prior works from the repository on GitHub and execute them on the same platform.

B. Full-chip Mask Optimization Performance against SOTA

In TABLE I, `ICCAD13-S` is the 10 small cases of `ICCAD2013` testcases, while `ICCAD-L` is the 10 large cases of `ICCAD2013` testcases. The average three metrics of 10 cases are calculated. Real design cases from `CHIP-L1` to `CHIP-L8` are of various layout scales, and all the metrics are calculated averagely. As EGAN-OPC, Neural-ILT, and Multi-ILT focus on local mask optimization, for MO on large scales of layout like 8192×8192 pixels which exceed the memory of GPU, layouts are divided into the maximum size for these works to do mask optimization and stitched together directly to go through simulator to get corresponding wafer image and metrics.

Our LVM-MO shows good results on L_2 leverage on academic datasets with the corresponding simulator on both `ICCAD13-S` and `ICCAD13-L`, with acceptable PVB. We achieve 25993.5, 48547.4 on L_2 averagely in `ICCAD13-S` and `ICCAD2013-L` compared with Multi-ILT's 28704.6 and 51028.2, which demonstrates the fidelity of our LVM-MO on local mask optimization.

On the real design test cases we generated, locally focusing MO works show their weaknesses. As layout patterns are much more complex in real design, leveraging L_2, PVB, and EPE is quite challenging. On $\sim 2.0 \times 2.0$ scales, as shown in TABLE I, [13]–[15] have poor performance. As these works ignore mask optimization on the large layout scale, their performance gain worse. For `Ful-ILT`, as it applied the divide-and-conquer method to do mask optimization in full-chip scale, it performs better on larger layout scale compared with these works. However, the divide-and-conquer method is rough on the full-chip level mask optimization, even leading to false optimization as shown in Fig. 6. LVM-MO flawlessly achieves perfect full-chip scale mask optimization from nature, which is illustrated Section III-B. On metrics, we achieve 144126.5, 4.3 average $4\mu m^2$ on L_2 and EPE compared with `Ful-ILT`'s 252134.0, 12.2, which is 58%, 183% improvement, with comparable PVB. The visualization result is shown Fig. 6.

The turn around time (TAT) comparison with the state of the arts is shown Fig. 5. As we applied a revolutionary method, using training and inference flow, LVM-MO can zero-shot full-chip scale masks within seconds. compared with the ILT-based optimization method that optimizes the layout from scratch, on layout scales of $60 \times 60 \mu m^2$ and $150 \times 150 \mu m^2$, our mask optimization time is 4 and 28 seconds, which is more than $1000\times$ speed leverage compared with the fast method, with remarkable fidelity as shown TABLE II. It is noteworthy that our model achieved zero-shot performance on the full-chip scale layout using just a single GPU, whereas other ILT-based methods require multiple GPUs to operate, leading to a significant increase in computational demands as the layout scales up. This underscores the remarkable efficiency of LVM-MO.

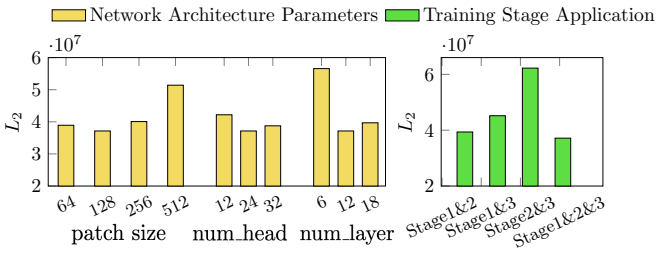


Fig. 7 Ablation studies on network architecture and training stage application.

C. Ablation Studies

As illustrated in Section III-A, Section III-B, and Section III-C, both the architecture parameters and the training stages affect our model's performance. We applied different settings of parameters and training stages, the results are shown in Fig. 7. L_2 is regarded as a metric for simplification, and the results are tested on $30 \times 30 \mu m^2$ layout size. As illustrated in Fig. 7, we research network architecture parameter settings, which are patch size, head number of self-attention, and layer number of feature extractor block. It can be noticed that patch size and layer number affect the model's performance heavily. For the training stage usage, the results state the performance during the same period. It can be witnessed that the layout reconstructing stage greatly enhances the model's geometry understanding ability, making the model converge to faster lithography basics learning.

V. Conclusion

In this paper, for the first time, we have proposed a novel full-chip mask optimization paradigm based on a large vision model driven by massive lithography data. Rather than relying on a crude application of various neural networks as mask optimizers, we have developed a foundation layout feature extractor capable of comprehending the interactions between layout polygons during the exposure as well as the principles of imaging physics and chemical processes. Our method demonstrates significant advantages over SOTA works in terms of lithographical performance, time efficiency, and the ability to combat full-chip layouts. We hope this work offers valuable insights to industry professionals on the utilization of large vision models and layout feature extractors in computational lithography tasks.

ACKNOWLEDGEMENT

This work was supported in part by the National Natural Science Foundation of China under Grants W2431046, and by MoE Key Lab of Intelligent Perception and Human-Machine Collaboration (ShanghaiTech University), the Shanghai Frontiers Science Center of Human-centered Artificial Intelligence, and the HPC Platform of ShanghaiTech University.

REFERENCES

[1] V. Singh, B. Hu, K. K. V. Toh, S. B. Bollepalli, S. Wagner, and Y. A. Borodovsky, "Making a trillion pixels dance," in *Proc. SPIE*, 2008.

[2] J. L. Digaum, G. Chiang, L. Wang, K. Braam, D. Gerold, Y. Wang, and T. H. Dam, "Full chip inverse lithography technology mask synthesis for advanced memory manufacturing," in *SPIE Advanced Lithography*, 2023.

[3] J. Zhu, L. Wang, B. Falch, T. H. Dam, Y. Wang, P. Panaite, M. Ji, and M. Su, "Scalable hierarchy extraction of repeating structures to enhance full chip mask synthesis," in *SPIE Advanced Lithography*, 2023.

[4] L. L. Pang, P. J. Ungar, A. Bouaricha, L. Sha, M. Pomerantsev, M. Niewczas, K. Wang, B. Su, R. Pearman, and A. Fujimura, "TrueMask ILT MWCO: full-chip curvilinear ILT in a day and full mask multi-beam and VSB writing in 12 hrs for 193i," in *Proc. SPIE*, 2020.

[5] S. Yin, W. Zhao, L. Xie, H. Chen, Y. Ma, T.-Y. Ho, and B. Yu, "Fuilt: Full chip ilt system with boundary healing," in *Proc. ISPD*, 2024.

[6] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[7] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.

[8] S. Damm, M. Laszkiewicz, J. Lederer, and A. Fischer, "Anomalydino: Boosting patch-based few-shot anomaly detection with dinov2," *arXiv preprint arXiv:2405.14529*, 2024.

[9] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo *et al.*, "Segment anything," in *Proc. ICCV*, 2023, pp. 4015–4026.

[10] H. H. Hopkins, "The concept of partial coherence in optics," Aug 1951.

[11] A. Vaswani, "Attention is all you need," 2017.

[12] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proc. ICCV*, 2021, pp. 10 012–10 022.

[13] H. Yang, S. Li, Z. Deng, Y. Ma, B. Yu, and E. F. Y. Young, "Gan-opc: Mask optimization with lithography-guided generative adversarial nets," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[14] B. Jiang, L. Liu, Y. Ma, B. Yu, and E. F. Y. Young, "Neural-ilt 2.0: Migrating ilt to domain-specific and multitask-enabled neural network," *IEEE TCAD*, 2022.

[15] S. Sun, F. Yang, B. Yu, L. Shang, and X. Zeng, "Efficient ilt via multi-level lithography simulation," in *Proc. DAC*, 2023.

[16] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *CoRR*, 2019.

[17] YosysHQ, "Picorv32 -a size-optimized risc-v cpu." [Online]. Available: https://github.com/YosysHQ/picorv32/?tab=readme-ov-file

[18] agalimberti, "Nocrouter - rtl router design in systemverilog." [Online]. Available: https://github.com/agalimberti/NoCRouter

[19] darklife, "Darkriscv." [Online]. Available: https://github.com/darklife/darkriscv

[20] "Freepdk45." [Online]. Available: https://eda.ncsu.edu/freepdk/freepdk45/

Efficient Rectification Signal Validation for Optimal Functional ECO Patch Generation

Tzu-Yu Tung*†, Yu-Ling Hsu*‡, Shao-Lun Huang‡ and Chung-Yang Ric Huang*†

†National Taiwan University, ‡Cadence Design Systems

tzuyutung0403@gmail.com, {yulingh, shaolun}@cadence.com, cyhuang@ntu.edu.tw

***Abstract*—Synthesis-based functional Engineering Change Order (ECO) algorithms, as classified in [1], are particularly effective for addressing functional bugs. These algorithms typically involve two primary steps: (1) identifying rectification signals to address functional mismatches, and (2) generating patch circuits based on these signals. While much of the existing research focuses on enhancing step (2), step (1) often remains somewhat ad hoc and inefficient.**

In this paper, we propose a novel approach for systematically collecting and validating all possible sets of rectification signals from a given set of candidates. Leveraging a heuristic for grouping and ranking rectification signals, our ECO flow efficiently identifies minimal patches while achieving a highly competitive runtime. Our contributions include three key innovations: a heuristic for identifying high-quality rectification candidates, an efficient algorithm for validating all feasible sets of rectification signals, and a signal grouping and ranking technique that ensures minimal patch size. When integrated with an open-source patch generation tool, our method demonstrates an average reduction of 44% in patch sizes compared to a leading commercial ECO tool on benchmark circuits.

Keywords— ECO, rectification point, quantifier elimination, SAT, QBF

I. Introduction

In modern VLSI design practices, functional Engineering Change Order (ECO) plays a vital role in addressing design errors and accommodating specification changes. As outlined in [1], ECO algorithms can be broadly classified into three categories: fault-model-based, sweeping-based, and synthesis-based.

For modifications involving common design errors—such as gate-type changes, missing or extra inverters, or rewiring—fault-model-based ECO algorithms [2]-[4] are the most effective, achieving minimal patch costs as noted in [1]. For more complex modifications that extend beyond fault models but remain confined to specific modules or sub-circuits, sweeping-based ECO algorithms [5]-[7] are more efficient. These algorithms minimize patches by merging functionally equivalent portions between the original and corrected (golden) circuits.

However, when modifications affect the design globally —such as altering the logical expression in a "case" statement, thereby impacting the satisfiability and observability don't-cares—rectifications are no longer local. In such scenarios, neither fault-model-based nor sweeping-based algorithms are effective. Synthesis-based ECO algorithms [8]-[12], by contrast, excel in addressing such functional changes [1]. These algorithms employ error-diagnosis techniques to identify potential rectification signals and then use synthesis techniques to generate patches that resolve the functional discrepancies.

Synthesis-based ECO algorithms can be formalized as follows. Let $F(\mathbf{x})$ and $G(\mathbf{x})$ represent the functions of the old and golden multiple-output circuits, respectively, where $\mathbf{x}$ is the set of primary inputs (PIs). Let $\mathbf{r}$ denote a set of internal signals in F. The function $F(\mathbf{x})$ is multi-rectifiable at signals $\mathbf{r}$ with respect to $G(\mathbf{x})$ if and only if:

$$\forall \mathbf{x} \,.\, \exists \mathbf{r} \,.\, F(\mathbf{x}, \mathbf{r}) \equiv G(\mathbf{x}) \qquad (1)\ [12]$$

The above single alternation ∀∃-formula is a 2QBF (Quantified Boolean Formula) problem. Solving this specific 2QBF problem is classified as a Π_2^P-complete problem [12] [13]. Earlier researches employed symbolic techniques such as Binary Decision Diagrams (BDDs) and truth tables to fix erroneous designs [14][15]. However, these approaches often suffered from runtime or memory explosion as design complexity increased.

To address this inefficiency, the authors in [9] proposed a SAT-based technique that assumes errors can be rectified on a single signal. This method computes patches through functional interpolation and leverages the scalability of modern SAT solvers. While this single-signal rectification assumption is generally valid—since rectifications can often revert to fixing one primary output (PO) at a time—it is not inherently optimal.

The authors in [11] then enhanced the above algorithm by incrementally generating partial patches on the rectification signals. This work naturally extended the SAT-based functional ECO technique to multi-error rectifications. Nevertheless, the order of the incremental fixes is somewhat arbitrary, and a poorly chosen sequence may lead to non-convergence on the patch generation. To address this limitation, the same authors later devised a multi-patch generation approach by interpolation with cofactor reduction [12]. Given a set of rectification signals, this new algorithm could fix multi-errors on the circuit simultaneously.

While these synthesis-based functional ECO algorithms appear to provide optimal patch solutions, their optimality is constrained by the pre-determined set of rectification signals. Since rectification signals are typically identified through error-diagnosis techniques, they often address only the error-triggering input vectors rather than the full scope of differences between the old and golden circuits. Consequently, these ECO algorithms frequently need to enumerate various combinations of rectification candidates and settle on the first set of signals that satisfies Equation (1).

Note that for clarification, we call the signals suggested by the error-diagnosis techniques the "rectification candidates", denoted as $\mathbf{r}_F$, and any of its subsets that satisfies Equation (1) the "rectification signals", denoted as $\mathbf{r}$. Therefore, to find the optimal ECO patch, we should enumerate all the possible $\mathbf{r}$ in $\mathbf{r}_F$ and compute:

$$\min_{\mathbf{r}:\forall \mathbf{r} \subseteq \mathbf{r}_F} \text{cost}(\mathbf{r}), \text{ in which } \forall \mathbf{x} \,.\, \exists \mathbf{r} \,.\, F(\mathbf{x}, \mathbf{r}) \equiv G(\mathbf{x}) \qquad (2)$$

where cost() is the cost function for the optimal patch based on $\mathbf{r}$.

However, given that generating an optimal patch is already computationally expensive, it seems impractical to

This work is supported by the National Science and Technology Council, Taiwan. Project No.: NSTC 113-2640-E-002-001

XXX-X-XXXX-XXXX-X/XX/$XX.00 ©20XX IEEE

iterate through all the possible subsets **r**'s to further optimize the patch.

To cope with this challenge, in this paper, we propose an novel rectification signal validation technique so that we can identify all the possible sets of rectification signals very efficiently. This is achieved by checking an approximated function, instead of Equation (1), for the rectifiability of a given set of candidate signals. The approximated function is constructed based on quantifier elimination and functional unateness techniques.

In addition, we further devise several heuristic algorithms so that we can obtain the optimal patch circuit with the fewest possible number of calls to the expensive patch generation routine. The experimental results show that our rectification signal validation and collection algorithm, when combined with an open-source ECO engine, can outperform a commercial ECO tool by an average of 44% in patch sizes on benchmark circuits.

The rest of the paper is organized as follows. In Section II, we first introduce our three-stage function ECO flow. The most important technique, the rectification signal validation algorithm, is presented in Section III. Section IV demonstrates the experimental results and Section V concludes the paper.

II. Our Functional ECO Flow

With the efficient rectification signal validation technique, our functional ECO algorithm can be outlined in the following three stages: (1) Preparation Stage, where we select the rectification candidates based on error diagnosis techniques [14][15]; (2) Validation Stage, where we apply an approximated function to collect all the sets of rectification signals; and (3) Patch Generation Stage, where we rank the sets of rectification signals and obtain the optimal patch with very few calls to the patch generation tool.

A. Preparation Stage

The choice of the rectification signals greatly determines the final quality of the patch. Intuitively speaking, if we can select a candidate closer to the bug location, it is more likely that the patch to fix the difference will be smaller. Therefore, we propose a *SuspiciousScore* function to evaluate the likelihood of a signal to be the bug location in the circuit.

The fundamental concept of this approach is to quantify a signal's ability to correct counterexamples. As illustrated in the example of Figure 1, the circuit is presented in an And-Inverter-Graph (AIG) format where each circle is an AND gate and dot and solid lines correspond to inverted and non-inverted fanins, respectively. With the counterexample "1101", flipping the value of gate(9) is able to fix the bug while flipping that of gate(7) will not. Therefore, only gate(9)'s suspicious score will be increased in this case.

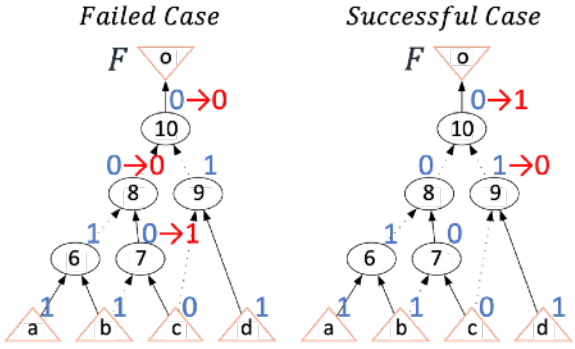


Fig. 1. Correcting a counterexample by flipping a signal's simulation value

In this preparation stage, we first perform random simulations and collect the set of counterexamples **cex** if there is a discrepancy on any output of the old (F) and golden (G) circuits. For pairs of internal signals that have the same responses on all the simulation patterns, we use the SAT solver to prove their equivalence. If asserted, they will be merged and form a new input frontier [16]. Otherwise, the patterns to distinguish the pairs will be tested if they are indeed counterexamples. In addition, we manage to somewhat balance the numbers of counterexamples to witness the non-equivalence among circuit outputs. For those outputs with relatively few counterexamples, we will resort to the SAT solver to generate more counterexamples.

Once we have collected the counterexamples, we will compute the *SuspiciousScore* of a signal s based on three factors:

- $N_{\mathbf{cex}}$: Number of counterexamples it can fix when flipping its simulation value.
- S_{cex}: For each counterexample *cex* in **cex**, the number of signals that can rectify this counterexample. This indicates the rarity of this signal's capability in fixing this counterexample.
- D_{max}: The longest distance from this signal to the circuit outputs. By introducing this factor, we favor rectification signals that are closer to circuit inputs.

Given a signal s, its *SuspiciousScore* is calculated as:

$$SuspiciousScore(s) = \left(\sum_{N_{cex}} \frac{1}{S_{cex}}\right) \cdot (1 + D_{max}) \qquad (3)$$

Note that we only need to compute the *SuspiciousScore*'s for the signals in F, and as we calculate them in the topological order, the time complexity is $O(S \cdot N)$, where S is the number of signals in F and N is the total number of counterexamples.

Lastly, we select the rectification candidates with the top n *SuspiciousScore*, where n is a programmable variable so that the selected candidates should cover all the non-equivalent POs.

B. Validation Stage

As indicated in Equation (2), there are two runtime bottlenecks in computing the optimal patch circuit: The first is to solve the 2QBF equation for every set of rectification signals, and the second is to call the patch generation process many times to seek the minimal patch.

To alleviate the second issue, we create this "validation stage" to collect all the possible sets of rectification signals first, leaving the expensive patch generation to the next stage. This enables the grouping and ranking on the sets of rectification signals, and if we can ensure that the first few sets of rectification signals can indeed render the optimal patch, we can early terminate the search process and largely reduce the number of calls to the patch generation routine.

For the runtime issue in solving the enormous amount of 2QBF problems, we devise an efficient rectification signal validation algorithm as illustrated in Algorithm I below. Note that instead of solving the 2QBF problem in Equation (1), we first invert it to:

$$\exists \mathbf{x} \,.\, \forall \mathbf{r} \,.\, F(\mathbf{x}, \mathbf{r}) \neq G(\mathbf{x}) \qquad (4)$$

and then construct an approximated quantifier-free function $H(\mathbf{x}, s)$ to remodel $\forall \mathbf{r} \,.\, F(\mathbf{x}, \mathbf{r}) \neq G(\mathbf{x})$ (details will be presented in the next section). Therefore, the 2QBF can be reduced to 1QBF (i.e. $\exists \mathbf{x} \,.\, H(\mathbf{x}, s)$) and be solved by SAT.

In Algorithm I, we first introduce a set of auxiliary selection variables $\mathbf{s}_F$ (Line 4) in order to enumerate all the

Algorithm I. Rectification Signal Validation

1. Function *RectificationSignalValidation*($\mathbf{x}, \mathbf{r}_F$):
2. // $\mathbf{x}$: set of circuit inputs
3. // $\mathbf{r}_F$: set of rectification candidates
4. Define $\mathbf{s}_F$: set of selection variables // Note: $|\mathbf{s}_F| = |\mathbf{r}_F|$
5. Define $H(\mathbf{x}, s)$: an approximated function with respect to a selection variable s in $\mathbf{s}_F$
6. Let $F'(\mathbf{x}) = F(\mathbf{x})$ // to be remodeled by $H(\mathbf{x}, s)$
7. for each signal r_i in $\mathbf{r}_F$ and s_i in $\mathbf{s}_F$
8. $H(\mathbf{x}, s_i) \leftarrow$ *ConstructProofModel*($\mathbf{x}, r_i$)
9. endfor
10. // The proof model for $\mathbf{s}_F$ is denoted as $H(\mathbf{x}, \mathbf{s}_F)$
11. Let $S = \bigvee_{s_i \text{ in } \mathbf{s}_F} s_i$ // to characterize the search on $\mathbf{s}_F$
12. Let $R = \varnothing$ // to record the sets of rectification signals
13. while (($\overrightarrow{\mathbf{s}_F} \leftarrow$ *SAT*$_1(S)$) == SATISFIABLE)
14. // $\overrightarrow{\mathbf{s}_F}$ is an assignment on variables in $\mathbf{s}_F$
15. $H'(\mathbf{x}, \mathbf{s}_F) \leftarrow$ *ApplyValue*($\overrightarrow{\mathbf{s}_F}, H(\mathbf{x}, \mathbf{s}_F)$)
16. If (($\overrightarrow{\mathbf{x}} \leftarrow$ *SAT*$_2(H'(\mathbf{x}, \mathbf{s}_F))$ == UNSATISFIABLE)
17. // $\overrightarrow{\mathbf{x}}$ is an assignment on variables in $\mathbf{x}$
18. $\mathbf{r} \leftarrow$ *UnsatAnalysis*($\mathbf{r}_F$, *SAT*$_2$)
19. $R \leftarrow$ *AddRecSignalSet*($\mathbf{r}, R$)
20. *SAT*$_1 \leftarrow$ *AddBlockingCls*($\mathbf{r}, \mathbf{s}_F$, *SAT*$_1$)
21. else
22. $\overrightarrow{\mathbf{r}} \leftarrow$ SatGen($\mathbf{r}_F$, *SAT*$_2$)
23. *SAT*$_1 \leftarrow$ *AddBlockingCls*($\overrightarrow{\mathbf{r}}, \overrightarrow{\mathbf{s}_F}$, *SAT*$_1$)
24. endif
25. endwhile
26. Return R

combinations of the rectification candidates. Each selection variable s_i in $\mathbf{s}_F$ is to determine whether the corresponding rectification candidate r_i is selected into the set of rectification signals. If yes, then the circuit $F(\boldsymbol{x})$ is remodeled with the approximated function $H(\mathbf{x}, s_i)$ (Lines 5-10). Otherwise, $F(\boldsymbol{x})$ remain the same with respect to r_i.

To collect all the possible sets of rectification signals, we employ two satisfiability solvers *SAT*$_1$ and *SAT*$_2$. *SAT*$_1$ is to enumerate the assignments on the selection variables $\mathbf{s}_F$. Initially, we define a Boolean function S as the disjunction of all the selection variables s_i's (Line 11). Obviously, it is satisfiable. Note that the decision variable order in *SAT*$_1$ follows the rank of the suspicious scores of variables in $\mathbf{s}_F$ so that the one with higher score will be picked first. Once we get an assignment $\overrightarrow{\mathbf{s}_F}$, we apply it to simplify the remodeled circuit $H(\mathbf{x}, s)$ and call *SAT*$_2$ to check the satisfiability (Lines 13 - 16).

If *SAT*$_2$ returns UNSATISFIABLE, we find a set of potential[1] rectification signals $\mathbf{r}$ and we call the UNSAT analysis routine to minimize the number of necessary variables in $\mathbf{r}$. This can reduce the set of rectification signals and thus minimize the patch (Lines 16-18). We then record $\mathbf{r}$ in R and add a blocking clause of $\mathbf{s}_F$ based on $\mathbf{r}$ to *SAT*$_1$.

Otherwise, if *SAT*$_2$ returns SATISFIABLE, it implies that the current assignment cannot lead to a valid rectification. Therefore, we preform the SAT generalization to find the minimal assignment of $\overrightarrow{\mathbf{r}}$ and add the corresponding blocking clause by $\overrightarrow{\mathbf{s}_F}$ to *SAT*$_1$ (Lines 22-23).

The above procedure (Lines 13-25) will repeat until all the combinations of the rectification candidates $\mathbf{r}_F$ have been tested.

C. Patch Generation Stage

Once we have collected all the sets of potential rectification signals, we can call the patch generation tool for each set of signals and obtain the optimal patch circuit. However, considering the large number of the sets and the long runtime of the patch generation routine, we should minimize the number of patch generation calls and hopefully the optimized patch can be found in the first few calls.

We observed that many sets of the rectification signals are quite similar and empirically it shows that their patch costs are almost the same. Therefore, we group the sets of the rectification signals by their similarities and pick the one that is firstly obtained in Algorithm I as the representative. The grouping criteria we adopt here is simple: if there are more than 50% of common signals among two sets, we will place them in the same group.

We also noticed a trend that the closer the rectification signals are to the PIs, the smaller the patch circuit it may be. Therefore, for each rectification signal, we define its "level" as the largest distance from the PIs to this signal and then sort the sets by the summation of the levels of the rectification signals in the set. Note that by picking the sets with the smaller summations of signal levels, we not only select the ones which are closer to PIs, but also favor the sets with smaller numbers of rectification signals.

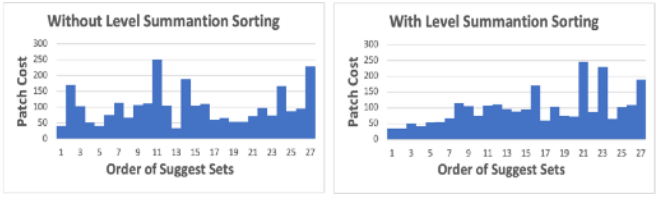


Fig. 2. Patch costs for different representative sets of rectification signals. Left figure are in the original order, and right one is sorted by the summation of the levels of the rectification signals.

As we generate the patches from the ordered sets, the next question is whether we have found the minimum patch and when we should stop the search. To answer this question, we implement a simulation annealing procedure where the search will be discontinued once there are accumulatively a few unsuccessful attempts to improve the patch cost. As demonstrated in the experimental results, our ECO flow can indeed obtain the minimal patches within just a few calls.

In addition, we also implement a simple heuristic to switch the patch generation from all-POs-at-a-time, group-by-group-POs, and one-PO-at-a-time. This is particularly helpful when the ECO problem involve multiple errors.

[1] We state it "potential" here because we remodel the circuit $F(\mathbf{x})$ with an approximated function $H(\mathbf{x}, s)$. It is likely that the suggested rectification signals may not render any valid patch. However, according to our experimental results, this rarely happens. We will cover the details in later sections.

III. Modeling and Implementation of the Rectification Signal Validation Algorithm

As mentioned in the previous section, given a set of rectification candidates $\mathbf{r}_F$, we extract the sets of valid rectification signals by constructing an approximated quantifier-free function $H(\mathbf{x}, s)$ for the formula $\forall \mathbf{r} \,.\, F(\mathbf{x}, \mathbf{r}) \neq G(\mathbf{x})$ and using SAT solver to collect the sets of rectification signals from its unsatisfiability cores. We will explain in details about the modeling and implementation of the algorithm.

A. Quantifier Elimination for a Rectification Candidate

Given an universally quantified formula $\forall r \,.\, F(\mathbf{x}, r)$, we can transform it into a quantifier-free formula as the conjunction of its cofactors on the quantifier variable r : $F(\mathbf{x}, r = 0) \wedge F(\mathbf{x}, r = 1)$. However, if there are multiple quantifier variables $\mathbf{r}$, there will be exponential terms in the conjunctive formula with respect to the number of variables in $\mathbf{r}$.

Consider the case that r has only one fanout $u(r)$ which is unate to F. We can then describe F as $A \cdot u(r)$, where A is certain function that is independent of r. Therefore, the universal quantification of F on r can be rewritten as:

$$\forall r \,.\, F = \forall r \,.\, (A \cdot u(r)) = A \cdot (\forall r \,.\, u(r)) \qquad (5)$$

Note that if the fanout function $u(r)$ is not too complicated, the elimination of the single quantifier variable r is relatively simple. For example:

$$\forall r \,.\, \mathrm{AND(r, a)} = 0$$

$$\forall r \,.\, \mathrm{OR(r, a)} = \mathrm{a}$$

$$\forall r \,.\, (ra + \bar{r}b + c) = (a + c)(b + c)$$

For the case that r has multiple fanouts which are unate to F, F can be represented as $A \cdot u_1(r)u_2(r) \ldots u_n(r)$ and the universal quantification can be rewritten as:

$$\forall r \,.\, F = A \cdot (\forall r \,.\, u_1(r))(\forall r \,.\, u_2(r)) \ldots (\forall r \,.\, u_n(r)) \quad (6)$$

B. Constructing the Approximated Function $H(\mathbf{x}, \mathbf{s}_F)$

Before we introduce our algorithm for constructing the approximated function $H(\mathbf{x}, s)$ to remodel Equation (4), we should first answer the following questions:

- How to apply quantifier elimination to the miter circuit $F(\mathbf{x}, r) \neq G(\mathbf{x})$

The miter function $F(\mathbf{x}, r) \neq G(\mathbf{x})$ can be rewritten as $F(\mathbf{x}, r) \vee G(\mathbf{x})) \wedge (\overline{F}(\mathbf{x}, r) \vee \overline{G}(\mathbf{x})$. As universal quantifier $\forall r$ can be distributed over the conjunctive operator '$\wedge$' and G(x) is independent of the signals r in F, the universal quantification over the miter function can be computed by:

$$\forall r \,.\, ((F(\mathbf{x}, r) \vee G(\mathbf{x}) \wedge (\overline{F}(\mathbf{x}, r) \vee \overline{G}(\mathbf{x})))$$
$$= (\forall r \,.\, F(\mathbf{x}, r)) \vee G(\mathbf{x})) \wedge (\forall r \,.\, (\overline{F}(\mathbf{x}, r)) \vee \overline{G}(\mathbf{x})) \quad (7)$$

- Given a set of rectification candidates $\mathbf{r}_F$, how to construct a unified circuit (proof model) such that different combinations of rectification candidates can be tested for Equation (4) using SAT engines

For each rectification candidate r_i in $\mathbf{r}_F$, we define an auxiliary selection variable s_i to act as a switch to include r_i into the set of rectification signals. Combining Equation (6) and (7), we can construct a proof model $H(\mathbf{x}, s_i)$ as Figure 3.

Note that since each u_i is unate to F and is applicable to Equation (5), when modeling multi-fanout cases, we actually need not synthesize the function 'A' nor represent $\forall r \,.\, F$ as the form in Equation (6). The proof model in Figure 3 acts as an implicit representation of Equation (6).

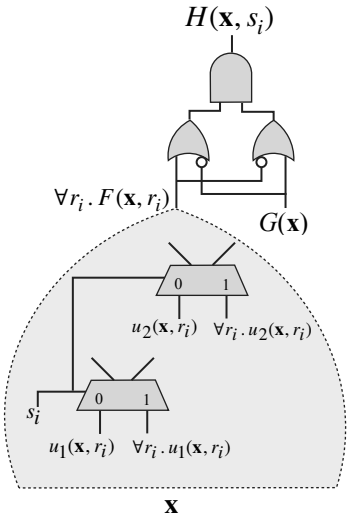


Fig. 3. Using selection variable s_i to construct the proof model

- How to extend Equation (7) for multiple POs

Up to this point, the modeling $H(\mathbf{x}, s_i)$ is considering single PO and is an exact model for checking the valid rectification signals.

For multiple PO cases, if we would like to find a set of rectification signals that can fix all the POs at one shot, we need to revise Equation (1) as:

$$\forall \mathbf{x} \,.\, \exists \mathbf{r} \,.\, (F_1(\mathbf{x}, \mathbf{r}) \equiv G_1(\mathbf{x})) \wedge (F_2(\mathbf{x}, \mathbf{r}) \equiv G_2(\mathbf{x})) \wedge \ldots \quad (1')$$

Taking the inversion as in Equation (4), we have:

$$\exists \mathbf{x} \,.\, \forall \mathbf{r} \,.\, (F_1(\mathbf{x}, \mathbf{r}) \neq G_1(\mathbf{x})) \vee (F_2(\mathbf{x}, \mathbf{r}) \neq G_2(\mathbf{x})) \vee \ldots \quad (4')$$

Let the proof model $H(\mathbf{x}, \mathbf{s}_F)$ for the multiple PO cases be the disjunction of the individual model $H(\mathbf{x}, s_i)$'s. That is:

$$H(\mathbf{x}, \mathbf{s}_F) = \bigvee_i H(\mathbf{x}, s_i) \qquad (8)$$

As shown in Algorithm I, we use this proof model to check if the set of rectification candidates is a valid set of rectification signals. That is, if the proof "SAT($H(\mathbf{x}, \mathbf{s}_F)$)" returns UNSATISFIABLE, we will save the corresponding set of signals $\mathbf{r}$ as a valid set of rectification signals.

However, since the universal quantifier $\forall r$ cannot be distributed over the disjunctive operator '$\vee$', $H(\mathbf{x}, \mathbf{s}_F)$ is NOT an exact proof model for Equation (4').

C. Application of $H(\mathbf{x}, \mathbf{s}_F)$ in the ECO Algorithm

Since $H(\mathbf{x}, \mathbf{s}_F)$ is NOT an exact proof model for Equation (4'), we need to check whether using $H(\mathbf{x}, \mathbf{s}_F)$ in the ECO Algorithm is sound and complete.

Let's first look at these propositions.

Proposition 1: $H(\mathbf{x}, \mathbf{s}_F)$ is an under-approximated proof model of Equation (4')

Proof:

By definition,

$$H(\mathbf{x}, \mathbf{s}_F) = \bigvee_i H(\mathbf{x}, s_i) = \bigvee_i \forall \mathbf{r} \,.\, (F_i(\mathbf{x}, \mathbf{r}) \neq G_i(\mathbf{x})) \quad [1]$$

Since

$$\bigvee_i \forall \mathbf{r} \,.\, (F_i(\mathbf{x}, \mathbf{r}) \neq G_i(\mathbf{x})) \implies \forall \mathbf{r} \,.\, \bigvee_i (F_i(\mathbf{x}, \mathbf{r}) \neq G_i(\mathbf{x})) \quad [2]$$

Let's denote the proof model for Equation (4') as:

$$E(\mathbf{x}, \mathbf{s}_F) = \forall \mathbf{r} \,.\, \bigvee_i (F_i(\mathbf{x}, \mathbf{r}) \neq G_i(\mathbf{x})) \quad [3]$$

From [1], [2] and [3], we have --

$$H(\mathbf{x}, \mathbf{s}_F) \implies E(\mathbf{x}, \mathbf{s}_F)$$

We can conclude that $H(\mathbf{x}, \mathbf{s}_F)$ is an under-approximated proof model of Equation (4') [QED]

Proposition 2: If we use the approximated function $H(\mathbf{x}, \mathbf{s}_F)$ and identify a set of rectification signals **r**, it is possible that we cannot generate a patch based on **r**.

Proof:

If we use the approximated function $H(\mathbf{x}, \mathbf{s}_F)$ and identify a set of rectification signals **r**, from Algorithm I, SAT($H(\mathbf{x}, \mathbf{s}_F)$) must return UNSATISFIABILITY. That is, $H(\mathbf{x}, \mathbf{s}_F)$ = false.

However, since $H(\mathbf{x}, \mathbf{s}_F) \implies E(\mathbf{x}, \mathbf{s}_F)$, it is possible that $E(\mathbf{x}, \mathbf{s}_F)$ is not 'false' and the proof SAT($E(\mathbf{x}, \mathbf{s}_F)$) may return SATISFIABILITY. Therefore, it is possible that we cannot generate a patch. [QED]

Proposition 3: For a set of rectification candidates $\mathbf{r}_F$, if there exists a patch that can be generated from certain subset of $\mathbf{r}_F$, say **r**, our approximated proof model $H(\mathbf{x}, \mathbf{s}_F)$ and thus our rectification signal validation algorithm (i.e. Algorithm I) will be able to guarantee such patch exists and identify this set of rectification signals **r**.

Proof:

This can be easily derived from Proposition 2. Due to the space limit, we omit the proof here. [QED]

In conclusion, our proposed rectification signal validation algorithm can efficiently collect all the sets of valid rectification signals and thus obtain the optimal path. Although it is possible that we may suggest certain sets of rectification signals which are not valid for patch generation, experiments show that this rarely happens and thus the runtime overhead is very small.

IV. Experimental Results

A. Experimental Settings

We present the experimental results of our proposed algorithm here. All experiments were conducted on a Linux workstation with 128GB RAM and a 2.2GHz Intel Xeon CPU. The testcases are from the 2017 CAD Contest [17].

To evaluate the quality of patches, we adopt the patch cost function from the 2021 CAD contest [18] as our criteria. We compare all our experimental results with the Conformal LEC tool by Cadence, using the Conformal ECO GXL 21.2 license, with the command *analyze eco* [19].

We leverage the open-source tool "*runeco*" [20] in Berkeley ABC [21] to generate the patches.

B. Comparison on Patch Costs and Runtime

Table I presents the experimental results on patch costs. The third column indicates the index for the occurrence of the optimal solution ("Opt Sol @") with respect to the number of sets of the rectification signals we have tested ("#Sets tested"). It can be seen that with the rectification signal grouping and ranking heuristic and the simulation annealing technique, we can obtain the optimal patch with no more than 4 calls to the patch generation tool.

The 4th column reports the number of rectification signals for the optimal patch. For this ICCAD 2017 benchmark, most cases can be fixed with less than four rectification signals.

We also implemented a baseline version in which the major difference is the heuristic grouping and ranking algorithm for the sets of the rectification signals during the patch generation stage. The results show that our full version has 12% advantages.

When compared with Conformal ECO, we can achieve 44% patch reductions on average. This asserts that with the exploration of more sets of rectification signals, we can consistently improve the patch size.

Nevertheless, we are unable to solve test06, 10, 11, and 19. The original *runeco* paper [20] specifically mentioned that these four cases require structural rewriting techniques to be resolved. Therefore, *runeco* is incapable of generating patches for them.

Table II is the runtime comparison. It is worth noting that 98% of our runtime is on the patch generation stage. This

TABLE I. PATCH COST COMPARISON

Testcase	Design Size	Opt Sol @ / #Sets tested	#sigs for Opt Sol	Patch Cost			Patch Ratio	
				Baseline	Ours	Conformal ECO	Ours / Baseline	Ours / Comformal
test01	12	1/1	1	**8**	**8**	**8**	1	1
test02	1348	1/4	1	31	**18**	73	0.58	0.25
test03	2617	1/4	1	7	**5**	17	0.71	0.29
test04	95	1/3	1	**3**	**3**	**3**	1	1
test05	25092	4/4	3	180	**159**	191	0.88	0.83
test06	14064	-	-	-	-	**367**	-	-
test07	3180	4/4	3	86	**63**	132	0.73	0.48
test08	2759	2/3	4	108	108	**86**	1	1.26
test09	6353	3/4	20	132	**123**	157	0.93	0.78
test10	2216	-	-	-	-	**353**	-	-
test11	2278	-	-	-	-	**95**	-	-
test12	13804	3/4	1	**1**	**1**	**1**	1	1
test13	369	1/1	1	**18**	**18**	66	1	0.27
test14	1981	1/1	7	**132**	**132**	619	1	0.21
test15	1886	1/4	1	**5**	**5**	99	1	0.05
test16	2371	1/3	2	61	**19**	1076	0.31	0.02
test17	2910	2/3	1	150	**145**	196	0.96	0.74
test18	4860	1/4	2	**19**	**16**	245	0.84	0.07
test19	13349	-	-	-	-	**169**	-	-
test20	30816	1/1	6	**59**	**59**	87	1	0.68
Average				62.5	55.1	202	0.88	0.56

TABLE II. RUNTIME COMPARISON

Testcase	Our Runtime (sec)				Conformal Runtime (sec)	Our Runtime / Conformal Runtime
	Preparation Stage	Validation Stage	Patch Gen Stage	Total		
test01	0.01	0.01	1.24	1.26	31	0.04
test02	0.09	0.1	48.3	48.49	36.5	1.33
test03	0.12	0.09	8.2	8.41	31.4	0.27
test04	0.01	0.02	13.51	13.54	27.7	0.49
test05	1.11	0.12	415	416.23	51	8.36
test06	-	-	-	-	662	-
test07	2.62	22.12	600	624.74	57.8	10.81
test08	0.2	0.46	828	828.66	42	19.74
test09	1.01	1.23	145	147.24	36.5	4.03
test10	-	-	-	-	70.7	-
test11	-	-	-	-	64.7	-
test12	0.98	6.82	330	337.8	39.6	8.51
test13	0.02	0.03	1.87	1.92	32.9	0.06
test14	0.11	0.31	608	608.42	50.4	12.1
test15	0.12	0.46	36.5	37.08	42.4	0.88
test16	0.51	5.14	334	339.65	186	1.82
test17	1.01	4.78	155	160.79	37.7	4.29
test18	0.72	5.09	148	153.81	235	0.65
test19	-	-	-	-	1750	-
test20	0.88	1.22	253	255.1	49.8	5.13
Average	0.6	3.63	245.4	249.63	81.75	4.9

implies that by transforming the problem from 2QBF to 1QBF, the runtime for collecting all the sets rectification signals is drastically reduced.

For the solved testcases, our runtime is about 4.9X when compared to Conformal ECO. On the one hand, this is due to the facts that we place more calls to patch generation (i.e. 3.44 times on average, from Table I). On the other hand, it says that there is room to improve the runtime for the open-source tool *runeco*.

C. Comparison and Analysis

To better evaluate the quality of the rectification signals and the optimality of the patch generation, we take the rectification signals collected by Conformal ECO and feed them to the *runeco* tool ((2) in Table III). When comparing it with our results (1), we can see that our ECO engine can optimize the patch for 20%. This indicates that our rectification signal collection and validation algorithm indeed contributes to the optimization of the patches.

On the other hand, if we compare (2) to (3) (i.e. to Conformal ECO alone), the *runeco* engine still outperforms Conformal ECO for 26%. It is not clear why the open-source tool performs better than commercial tool in these cases. However, the improvement (26%) is still less than the 44% improvement (ours vs. Conformal), indicating that our rectification signal collection and validation algorithm again plays an important role in optimizing the patches.

Ideally, we should have an experiment that collects our identified rectification signals and asks Conformal ECO to generate the patches. However, there is no such option in Conformal ECO and thus we cannot perform such comparison.

TABLE III. DIFFERENT RECTIFICATION SIGNAL + PATCH GEN

Testcase	Design Size	Patch Cost (1) Ours alone	(2) Conformal RS + runeco patch gen	(3) Conformal alone	(1) / (2)	(2) / (3)
test01	12	**8**	**8**	**8**	1	1
test02	1348	**18**	39	73	0.46	0.53
test03	2617	**5**	**5**	17	1	0.29
test04	95	**3**	**3**	**3**	1	1
test07	3180	**63**	70	132	0.9	0.53
test09	6353	**123**	142	157	0.87	0.9
test12	13804	**1**	**1**	**1**	1	1
test17	2910	**145**	178	196	0.82	0.91
test18	4860	**16**	129	245	0.12	0.53
Average		36.9	54.7	182	0.80	0.74

V. CONCLUSION

We propose a novel rectification signal validation algorithm to efficiently collect all possible sets of rectification signals for optimal patch generation in functional ECO problems. Additionally, we introduce a grouping and ranking strategy for these rectification signal sets, allowing us to achieve optimal patch generation with minimal calls to the patch generation tool.

Experimental results indicate that by exploring multiple sets of rectification signals, our approach reduces patch costs by 44% on average compared to a commercial tool. However, some test cases involving structural changes fail when processed by the third-party patch generation tool (*runeco*). In the future, we plan to integrate our rectification signal validation technique with sweeping-based and fault-model-based ECO engines to handle a broader range of testcases.

REFERENCES

1. S.-L. Huang, W.-H. Lin, P.-K. Huang and C.-Y. Huang, "Match and Replace: A Functional ECO Engine for Multierror Circuit Rectification," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 32, no. 3, pp. 467-478, March 2013, doi: 10.1109/TCAD.2012.2226456. https://doi.org/10.1109/TCAD.2012.2226456
2. K.-H. Chang, I. L. Markov and V. Bertacco, "Fixing Design Errors with Counterexamples and Resynthesis," 2007 Asia and South Pacific Design Automation Conference, Yokohama, Japan, 2007, pp. 944-949, doi: 10.1109/ASPDAC.2007.358111. Test in Europe Conference Exhibition (DATE). 1036–1041.
3. A. Veneris and I. Hajj, “Design error diagnosis and correction via test vector simulation,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 18, no. 12, pp. 1803–1816, Dec. 1999.
4. M. Abadir, J. Ferguson, and T. Kirkland, “Logic design verification via test generation,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 7, no. 1, pp. 138–148, Jan. 1988.
5. S. Krishnaswamy, H. Ren, N. Modi and R. Puri, "DeltaSyn: An efficient logic difference optimizer for ECO synthesis," 2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers, San Jose, CA, USA, 2009, pp. 789-796, doi: 10.1145/1687399.1687546.
6. H. Ren, R. Puri, L. Reddy, S. Krishnaswamy, C. Washburn, J. Earl, and J. Keinert, “Intuitive ECO Synthesis for High Performance Circuits,” in Proc. Design Autom. Test in Europe Conf. Exhib. (DATE), pp. 1002–1007, 2013.
7. D. Hoffmann and T. Kropf, “Efficient design error correction of digital circuits,” in Proc. ICCD, Sep. 2000, pp. 465–472.
8. C.-C. Lin, K.-C. Chen, and M. Marek-Sadowska, “Logic synthesis for engineering change,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. , vol. 18, no. 3, pp. 282–292, Mar. 1999.
9. B.-H. Wu, C.-J. Yang, C.-Y. Huang, and J.-H. Jiang, “A robust functional ECO engine by SAT proof minimization and interpola- tion techniques,” in Proc. IEEE/ACM Int. Conf. ICCAD, Nov. 2010, pp. 729–734.
10. A. Ling, S. Brown, S. Safarpour, and J. Zhu, “Toward automated ECOs in FPGAs,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 30, no. 1, pp. 18–30, Jan. 2011.
11. K. -F. Tang, C. -A. Wu, P. -K. Huang and C. -Y. Huang, "Interpolation-based incremental ECO synthesis for multi-error logic rectification," 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), San Diego, CA, USA, 2011, pp. 146-151. Design Automation Conference. 1–6.
12. K. -F. Tang, P. -K. Huang, C. -N. Chou and C. -Y. Huang, "Multi-patch generation for multi-error logic rectification by interpolation with cofactor reduction," 2012 Design, Automation \& Test in Europe Conference \& Exhibition (DATE), Dresden, Germany, 2012, pp. 1567-1572, doi: 10.1109/DATE.2012.6176722.
13. Celia Wrathall. Complete sets and the polynomial-time hierarchy. Theoretical Computer Science, 3(1):23–33, 1976.
14. A. M. Wahba, "BDD/AOG-based algorithm for multiple error rectification in combinational circuits," ICECS 2000. 7th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.00EX445), Jounieh, Lebanon, 2000, pp. 658-661 vol.2, doi: 10.1109/ICECS.2000.912964.
15. Shi-Yu Huang, Kuang-Chien Chen and Kwang-Ting Cheng, "AutoFix: a hybrid tool for automatic logic rectification," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 9, pp. 1376-1384, Sept. 1999, doi: 10.1109/43.784128.
16. A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification". ERL Technical Report, EECS Dept., UC Berkeley, March 2005..
17. 2017 CAD Contest, https://www.iccad-contest.org/2017.
18. 2021 CAD Contest, https://www.iccad-contest.org/2021/.
19. Conformal ECO Designer, https://www.cadence.com/en_US/home/tools/digital-design-and-signoff/functional-eco/conformal-eco-designer.html.
20. A. Q. Dao, N-Z Lee, L-C Chen, M. P-H Lin, J-H R. Jiang, A. Mishchenko, and R. Brayton, "Efficient Computation of ECO Patch Functions," 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 2018, pp. 1-6, doi: 10.1109/DAC.2018.8465898.
21. Berkeley ABC: A System for Sequential Synthesis and Verification, https://people.eecs.berkeley.edu/~alanmi/abc/.

KLiNQ: Knowledge Distillation-Assisted Lightweight Neural Network for Qubit Readout on FPGA

Xiaorang Guo*, [1], Tigran Bunarjyan*, [1], Dai Liu[1], Benjamin Lienhard[2, 3] and Martin Schulz[1]
[1]Chair of Computer Architecture and Parallel Systems, Technical University of Munich, Garching, Germany
[2]Department of Chemistry, Princeton University, Princeton, NJ 08544, USA
[3]Department of Electrical and Computer Engineering, Princeton University, Princeton, NJ 08544, USA
Email: {xiaorang.guo, tigran.bunarjyan, dai.liu}@tum.de, blienhard@princeton.edu, schulzm@in.tum.de

***Abstract*—Superconducting qubits are among the most promising candidates for building quantum information processors. Yet, they are often limited by slow and error-prone qubit readout—a critical factor in achieving high-fidelity operations. While current methods, including deep neural networks, enhance readout accuracy, they typically lack support for mid-circuit measurements essential for quantum error correction, and they usually rely on large, resource-intensive network models. This paper presents KLiNQ, a novel qubit readout architecture leveraging lightweight neural networks optimized via knowledge distillation. Our approach achieves around a 99% reduction in model size compared to the baseline while maintaining a qubit-state discrimination accuracy of 91%. KLiNQ facilitates rapid, independent qubit-state readouts that enable mid-circuit measurements by assigning a dedicated, compact neural network for each qubit. Implemented on the Xilinx UltraScale+ FPGA, our design can perform the discrimination within 32 ns. The results demonstrate that compressed neural networks can maintain high-fidelity independent readout while enabling efficient hardware implementation, advancing practical quantum computing.**

***Index Terms*—Superconducting Qubits, Qubit Readout, Knowledge Distillation, FPGA**

I. Introduction

Quantum computers are expected to tackle specific complex problems much faster than classical computers by leveraging quantum-mechanical properties. Among various quantum bit (qubit) technologies, superconducting qubits stand out for their fast gate operations and high fidelity. However, within the control pipeline of superconducting quantum computers, qubit readout remains one of the slowest and most error-prone processes [1], [2]. In this case, improving the qubit readout accuracy helps to enable high-fidelity quantum computers. Accelerating the readout process is essential for supporting the operation of large quantum circuits within the limited coherence times.

Qubit readout is a complex process that involves both analog and digital operations. In this work, we focus on the digital aspect, processing the in-phase (I) and quadrature (Q) signals obtained from Analog-to-Digital Converters (ADCs) to discriminate between the qubit states ('0' and '1'). This process is also known as single-shot qubit-state discrimination [3], [4]. In order to achieve high-fidelity qubit readout, a robust and accurate discrimination algorithm is required. Previous works have applied simple machine learning or statistical techniques, such as Support Vector Machiness (SVMs) [5], hidden Markov models [6], and Match Filters (MFs) [7], to improve the readout fidelity. However, the methods still fall short of the requirements for high-fidelity quantum computing. Thus, Benjamin et al. [3] proposes a deep neural network-based approach that significantly enhances readout accuracy. This method utilizes the Feed Forward Neural Network (FNN) architecture and directly takes time-sequenced digitized signals from the ADCs as input, without requiring post-processing. Once trained, it classifies the states of all qubits with high accuracy.

Although the FNN effectively discriminates the qubit states, the overhead of software-based classification poses a significant challenge for superconducting quantum computers. Since the readout signals must be transferred repeatedly between the control and readout hardware and software, the resulting latency often exceeds the qubit coherence time [8]. Therefore, *readout discriminators must be implemented on dedicated hardware*.

For this reason, current designs are migrating neural networks onto Field Programmable Gate Arrays (FPGAs) to reduce both communication and discrimination latency. The state-of-the-art work named HERQULES [9] introduces an architecture that integrates specific MFs tailored to each qubit into the FNN proposed by work [3] but with a reduced network size benefiting from their selected features. This architecture improves the readout accuracy; however, several critical challenges remain unresolved: 1) the current FNN architecture requires simultaneous readout of all qubits, which limits its applicability in mid-circuit measurement scenarios

* Equal Contribution.

This work was funded by the German Federal Ministry of Education and Research (BMBF) under the funding program Quantum Technologies - From Basic Research to Market under contract number 13N16087, as well as from the Munich Quantum Valley (MQV), which is supported by the Bavarian State Government with funds from the Hightech Agenda Bayern.

where qubits need to be measured potentially at arbitrary times and their outcomes influence subsequent operations; 2) the design requires an additional digital demodulation process, hindering real-time processing performance; and 3) existing designs are either too heavy to scale to large systems or are feature-limited for independent qubit processing. For instance, when HERQULES is adapted for single-qubit readout, its performance significantly degrades compared to its original configuration optimized for a five-qubit system. Another approach [10] applies quantization techniques to reduce the FNN size from work [3]. While this avoids demodulation, it sacrifices accuracy and fails to support mid-circuit measurements.

To address these challenges, we propose *KliNQ: Knowledge Distillation-Assisted Lightweight Neural Network for Qubit Readout on FPGA*. KliNQ leverages knowledge distillation, where a large, feature-rich teacher network transfers its expertise to a compact student network optimized for FPGA implementation. By utilizing only the distilled knowledge, we reduce the network size by 99% from the teacher model while maintaining a comparable discrimination accuracy of around 91%. This significantly lowers FPGA resource utilization, making KliNQ ideal for realizing Fault-tolerant Quantum Computing (FTQC) applications. In addition to shrinking the network, KliNQ employs an independent architecture with lightweight neural networks tailored to individual qubits. This design enables independent qubit discrimination without requiring simultaneous readouts, making KliNQ particularly well-suited for mid-circuit measurement and real-time feedback control. Overall, our main contributions are:

- We introduce KliNQ, a qubit-state discriminator that leverages knowledge distillation to achieve around 99% reduction in network size without compromising readout accuracy.
- We design KliNQ with distilled independent neural networks for each qubit, enabling support for mid-circuit measurements while achieving a comparably high readout accuracy of 0.906 (average for five qubits).
- We implement this design on an FPGA platform, utilizing a highly parallelized architecture to achieve readout times of 32 ns with low resource utilization.

II. Backgrounds

In this section, we discuss the principles of qubit readout for superconducting qubits and introduce knowledge distillation as a method for network compression.

A. Qubit Readout

In quantum computing, the qubit is the fundamental unit of storing information, capable of representing both states '0' and '1' in superposition, which is expressed by the formula $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $|\alpha|^2 + |\beta|^2 = 1$. During measurement, the qubit collapses to either '0' or '1', losing its superposition property. This process is known as qubit readout. For superconducting qubits, the readout is typically performed by sending microwave pulses into the qubit resonators for a few hundred nanoseconds to microseconds [11], [12]. Depending on the qubit state, the microwaves passing through the resonators undergo qubit-state-specific phase shifts, which are then down-sampled and eventually digitized by ADCs into I and Q values. These I/Q values fluctuate over time during the readout, and their trace follows different convergence paths based on the qubit state, as introduced by Maurya et al. [9]. Here, we focus on using neural networks to analyze these traces to discriminate the qubit state with high accuracy and as early as possible.

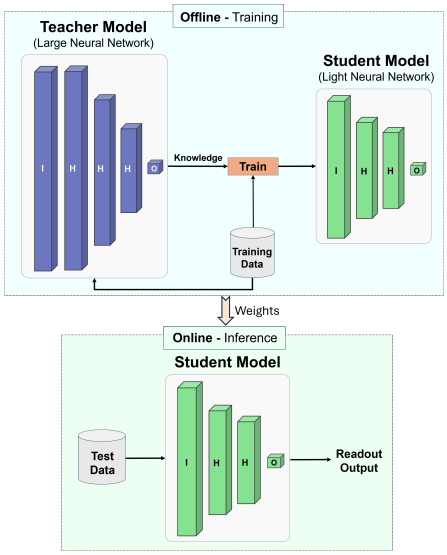


Fig. 1. System Architecture of Knowledge Distillation-Assisted Discriminator: Offline Training and Efficient FPGA-Based Online Inference

B. Knowledge Distillation

Knowledge distillation is a model compression technique that transfers knowledge from a large, complex neural network (the "teacher") to a smaller, more efficient "student" network, enabling the creation of lightweight models that retain the predictive power of their larger counterparts while significantly reducing computational and memory demands [13], [14]. The training process combines standard supervised learning with a distillation objective, aligning the student's outputs with the teacher's softened probabilities, which carry nuanced class relationship information [15]. Since only lightweight student networks are needed during inference, this method proves highly effective in resource-constrained environments like FPGAs, where real-time performance and efficiency are critical. Additionally, knowledge distillation is advantageous for tasks requiring complex pattern recognition, such as qubit-state readout in quantum computing, facilitating models that maintain high accuracy while optimizing for reduced resource consumption and scalability.

III. Methodology

Fig. 1 illustrates the proposed system architecture of this work, which incorporates an offline training and distillation scheme alongside an online inference process. This section will discuss how knowledge distillation is generally performed

in the system and the optimization strategies employed through hardware-software co-design to enhance performance and lead to the optimal model for KLiNQ.

A. Teacher Model: Experimental Design

To develop a robust teacher model for knowledge distillation, we conduct extensive experiments to identify an architecture that effectively balances accuracy, generalizability, and efficiency. Prior research [3], [9], [10] has shown that FNN outperforms Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) for qubit-state readout tasks. Compared to more complex architectures, the simplicity and flexibility of FNN enable efficient processing of I/Q traces and ensure robust generalization, making it the ideal choice for this application.

The teacher FNN was trained on 1 μs qubit measurement data using multiplexed I/Q signal traces flattened into 1000 inputs. The network architecture includes three hidden layers with 1000, 500, and 250 neurons, followed by an output layer for binary classification to distinguish between qubit states ('0' or '1'), optimizing fidelity scores across all qubits. We used the geometric mean of individual qubit assignment fidelities to evaluate the network models as the primary metric for discrimination accuracy. This geometric mean fidelity, F_{GM}, is defined as:

$$F_{\text{GM}} = \left(\prod_{i=1}^{N} F_i \right)^{1/N},$$

where N represents the total number of qubits and F_i is the readout accuracy for each qubit. This metric emphasizes consistent performance by penalizing outliers with low accuracy, ensuring reliable model performance across all qubits.

B. Data Preprocessing and Input Optimization

To ensure that the student networks achieve high fidelity and computational efficiency from a hardware-software codesign perspective, we need to optimize the input representation as small as possible. This involves two key steps: (1) averaging measurements over carefully selected intervals and (2) incorporating matched filters to enhance signal quality.

1) Averaging Interval Values: We perform a series of steps to preprocess the qubit-specific traces to achieve a compact input size while maintaining classification accuracy. We experimentally determine the optimal measurement interval for averaging I and Q components for each trace, effectively reducing the trace duration and dimensionality. Averaging compresses the raw data into a lower-dimensional representation by summarizing temporal features while preserving the most salient characteristics essential for classification.

2) Matched Filters: While averaging significantly reduced the input size, experiments show that averaged trace alone cannot achieve a high classification fidelity, especially for qubits with subtle qubit-state-readout signal differences. To address this, we incorporate MFs to provide an additional feature that enhances the discrimination capability of the

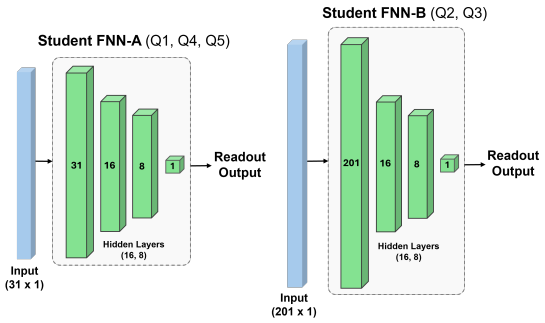


Fig. 2. KLiNQ Student Network architectures for qubit groups of (Q1, Q4, Q5) and (Q2, Q3).

student models. The MF envelope for each qubit was trained by maximizing the separation between ground-state ('0') and excited-state ('1') traces. The envelope is computed as:

$$\text{MF Envelope} = \frac{\text{mean}(T_0 - T_1)}{\text{var}(T_0 - T_1)},$$

where T_0 and T_1 represent the readout traces for the ground and excited states, respectively. During inference, the MF output is calculated by applying the trained envelope to the respective trace via a dot product operation, resulting in a single scalar feature. This scalar value, along with averaged I and Q components, is used as an additional input feature to form a representative input for the student models. This has been proven to significantly improve the performance of the student models by enhancing the signal-to-noise ratio (SNR) and capturing qubit-specific dynamics.

C. Distillation Training and Process

Knowledge distillation is implemented to transfer the predictive capabilities of the teacher network to smaller, more efficient student networks. The primary objective of this process is to ensure that the student models maintain high classification accuracy while achieving a compact architecture, making them suitable for resource-constrained environments.

Distillation Training: The training process involves teaching the student network to replicate the behavior of the teacher network by minimizing a composite loss function. This loss combines two components: (1) a supervised learning objective, ensuring the student learns from the dataset's ground truth labels (hard labels) and (2) a distillation objective, aligning the student's outputs with the teacher's predictions (soft labels). The composite loss function, $\mathcal{L}_{\text{distill}}$, is defined as:

$$\mathcal{L}_{\text{distill}} = \alpha \mathcal{L}_{\text{CE}} + (1 - \alpha)\mathcal{L}_{\text{KD}},$$

where $\mathcal{L}_{\text{CE}}$ represents the binary cross-entropy loss, calculated between the student's predictions and the ground truth (hard labels).$\mathcal{L}_{\text{KD}}$ stands for the distillation loss, computed as the Mean Squared Error (MSE) between the softened logits of the teacher and the student (soft labels), and α represents a weighting factor that balances the supervised learning and distillation components.

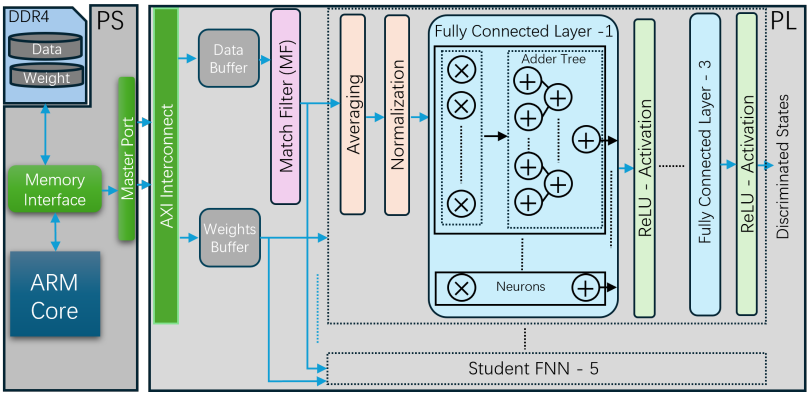


Fig. 3. FPGA-based FNN architecture.

Distillation Process and Inference: As mentioned earlier in Fig. 1, distillation happens during the training phase, where the teacher model is pre-trained on the dataset and subsequently used to guide the student model. The pre-trained teacher network provides softened probabilities (soft labels) as supplementary supervisory signals for the student network's distillation process. The student network, initialized with random weights, is trained using gradient descent to minimize a composite $\mathcal{L}_{\text{distill}}$.

For inference, only the student models are deployed, having inherited the predictive capabilities of the teacher network through offline distillation. The compact design of the student models supports efficient, low-overhead inference, making them ideal for qubit-state discrimination in subsequent FPGA implementations.

D. Student Network Architectures

We now introduce two distinct student network architectures, as illustrated in Fig. 2, based on the experimental observations: a small FNN-A for qubits 1, 4, and 5, which exhibit high SNR and are easier to discriminate, and a relatively larger FNN-B for qubits 2 and 3, where the SNR is suboptimal, particularly for qubit 2. Note that the input size of the networks is fixed, and when the trace length changes, we dynamically adjust the number of samples to be averaged to match the required output size. In this section, we use 1 μs-long traces as an example for description.

- **Student FNN-A (Q1, Q4, Q5):** This architecture is optimized for qubits whose qubit-state discrimination performance is adequately captured by the 64-ns (32 samples) averaging interval. The input consists of the 30-dimensional averaged I/Q data plus the MF feature, forming a 31-dimensional vector. The network comprises two hidden layers with 16 and 8 neurons, respectively, followed by a single output layer for binary classification.
- **Student FNN-B (Q2, Q3):** The state discrimination of qubits 2 and 3 requires larger input sizes due to increased noise levels, crosstalk effects, and greater variability in the readout traces. For these qubits, a 10-ns (5 samples) averaging interval has been identified as optimal, resulting in 200 averaged data points from the I/Q traces. The input size was expanded to 201 dimensions when combined with the MF feature. This network also features two hidden layers with 16 and 8 neurons, followed by a single output layer.

These architectures have been selected as the most optimal configurations in terms of balancing fidelity, computational efficiency, and input sizes.

IV. Lightweight Neural Networks on FPGA

In this section, we describe the deployment of our lightweight student neural network on FPGA and discuss its architecture. As mentioned in Sec. III, due to the different properties of qubits, we design different sizes of neural networks to balance accuracy and efficiency. In this case, the network we develop on FPGA is also re-configurable in size.

Fig. 3 illustrates the overall structure of the FPGA-based FNN, which comprises DDR memory, the Processing System (PS) part, and the Programmable Logic (PL) component. The DDR memory is responsible for storing the test data of the qubit traces as well as the trained weights of the neural network. To be noted, in actual quantum systems, KliNQ can process the raw data directly from the ADCs to perform state discrimination. However, this work focuses on the algorithm and FPGA implementation, so we use the DDR memory to store the traces as a substitute. PS serves as a data exchange interface between the memory and our customized logic to offload the qubit traces, necessary parameters, and weights to PL through the AXI bus protocol.

In the PL, we adopt a modular design approach to deploy each FNN layer sequentially, enabling pipelined functionality. As discussed in Sec. III-D, two network architectures are developed for different qubits to optimize performance. Although these models differ in configurations, they share the same overall architecture. Therefore, rather than introducing them separately, we provide a unified overview. We select a 32-bit fixed-point format for the data representation, allocating 16 bits for the integer and 16 bits for the fractional part to maintain discrimination accuracy.

Following the architectures outlined in Sec. III-D, the data pre-processing stage begins with an average layer that computes the average value of a group of qubits, reducing

the input size to the neural network and minimizing resource utilization. This is followed by a normalization layer, which optimizes the data distribution, improves algorithm efficiency, and mitigates the risk of overflow in the fully connected layers, where multiple multiplication and addition operations are performed. To perform normalization, the expression $\frac{x-x_{\min}}{\sigma_x}$ is used in this work, where σ_x represents the standard deviation of the observed qubit traces. The original formula typically requires division, which consumes significant resources and time on the FPGA, limiting real-time performance. However, as the $x_{\min}$ and σ_x are prepared during training, the σ_x is also approximated as a power of 2. Thus, we replace the division with shift operations and can get the results within only two clock cycles. Simultaneously, another feature is extracted from the MF layer. Given the identical operations and data flow (Multiply-Accumulate (MAC)), this layer reuses the same design as a fully connected layer, which will be briefly introduced later.

After normalization, we concatenate the I and Q values together with the MF feature into a single input vector, which is then fed into the fully connected layer. Within each layer, neuron operations are executed in parallel, so the latency of a layer equals that of a single neuron. Neurons perform input-weight multiplications in a 4-stage pipeline, utilizing time-multiplexed DSP resources on the FPGA, which are exclusively responsible for executing these multiplication operations. Each multiplication is implemented using fully combinational logic, achieving a latency of one clock cycle, so the overall latency of multiplication is four cycles. The outputs are summed along with the bias through an adder tree structure (shown in Fig. 3), achieving a latency with $\lceil \log_2(\mathrm{n}) \rceil + 1$, where n is the number of inputs. However, the latency varies slightly with the number of inputs due to the ceiling operations. Each fully connected layer is followed by a ReLU activation function, which introduces the non-linearity between layers. The sign bit is checked to determine whether the output remains unchanged or is set to zero. Additionally, overflow conditions are managed in the activation layer to ensure correct functionality. Meanwhile, we handle overflows in the activation layer to ensure the correct functionality. Subsequently, the whole process is repeated twice more, culminating in a final single value that serves as the discrimination result.

V. Evaluations

A. Experimental Setup

Datasets and baseline: In this work, we train and evaluate our method using real-measured experimental data from a superconducting quantum information processor comprising five qubits [3]. The dataset comprises measurements from all 32 possible qubit-state permutations, where each qubit's state is characterized by a sequentially recorded Q and I value. While the full recorded trace spans 2 μs, only the first 1 μs is utilized in our analysis, as this duration provides nearly identical discrimination accuracy compared to the full sequence. For network evaluation, we adopt a setup similar to previous works [3], [9], leveraging 15,000 traces per qubit-state configuration for training and 35,000 traces for testing. In addition, we use work [3] as the baseline model in the comparison.

FPGA Hardware: The hardware implementation of this design is programmed in Verilog and synthesized using Xilinx Vivado 2024. The complete project is deployed on a Xilinx Zynq RFSoC (ZCU216) [16], operating at 100 MHz.

B. Discrimination Accuracy

We begin by evaluating one key metric for discriminators: readout fidelity. Tab. I presents the readout fidelities of KLiNQ compared with related works with the readout-trace length of 1 μs. The comparison includes the individual readout accuracies and their geometric means of the readout fidelities for five qubits. To provide a comprehensive comparison, we compute two geometric means: one for all five qubits and another excluding qubit 2, as noise significantly impacts its performance. The original papers of the baseline FNN and HERQULES are designed for a five-qubit synchronous readout system. However, in this comparison, we reproduce their design and test their architecture on independent readouts. The result shows that KLiNQ achieves a competitive fidelity compared to the large baseline model and outperforms HERQULES with more than 1%. Here, we exclude work [10] as it does not explicitly report accuracy.

TABLE I
Qubit-Readout Fidelity Comparison Tested on Independent Readout Scenario

Design	Qubit 1	Qubit 2	Qubit 3	Qubit 4	Qubit 5	F_{5Q}	F_{4Q}
Baseline FNN[1]	0.969	0.748	0.940	0.946	0.970	0.910	0.956
HERQULES[2]	0.965	0.730	0.908	0.934	0.953	0.893	0.940
KLiNQ	0.968	0.748	0.929	0.934	0.959	0.904	0.947

[1] Baseline FNN [3] reports a geometric mean (F_{5Q}) of 0.912 for the five-qubit readout system.
[2] HERQULES [9] reports a geometric mean (F_{5Q}) of 0.927 for the five-qubit readout system.

We further evaluate the robustness of KLiNQ's readout performance across varying trace durations. Tab. II summarizes the readout fidelities as trace durations decrease from 1 μs to 500 ns. While the geometric mean fidelity across all five qubits declines with shorter traces, certain qubits achieve optimal performance at specific shorter durations, highlighted in green. By utilizing the optimal trace durations for each qubit, KLiNQ achieves an improved five-qubit fidelity (F_{5Q}) of 0.906. The behavior of the fidelities across trace lengths is also depicted in Fig. 4(a), where all qubits except Qubit 2 demonstrate consistent and robust behavior. Moreover, Fig. 4(b) shows that KLiNQ maintains superior readout fidelity and robustness compared to the state-of-the-art HERQULES, particularly excelling at shorter readout-trace durations.

TABLE II
Readout Fidelity of KLiNQ vs. Readout-Trace Duration

Design	Duration	Qubit 1	Qubit 2	Qubit 3	Qubit 4	Qubit 5	F_{5Q}
KLiNQ NNs	1μs	0.968	0.748	0.929	0.934	0.959	0.904
	950ns	0.967	0.744	0.925	0.934	0.956	0.901
	750ns	0.962	0.736	0.927	0.932	0.963	0.900
	550ns	0.944	0.720	0.930	0.921	0.967	0.891
	500ns	0.935	0.717	0.929	0.917	0.966	0.887

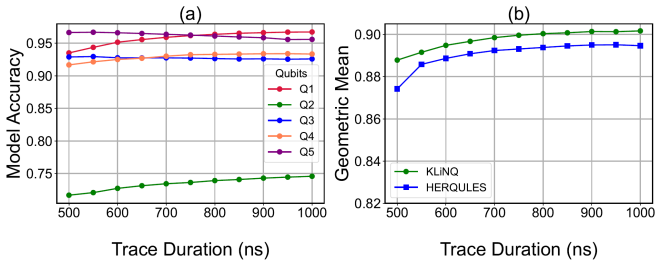


Fig. 4. (a) Accuracy of individual qubit-state discriminator when varying the readout-trace duration. (b) Geometric mean comparison of readout fidelity between KLiNQ and HERQULES across varying readout-trace lengths.

C. Network Compression Rate

Network Compression Rate (NCR) is a key metric in evaluating knowledge distillation. To assess this, we compare the number of parameters in KLiNQ's five individual qubit discrimination system to those in the teacher NN architectures. As illustrated in Fig. 5, the teacher NNs collectively comprise a substantial 8.14 million parameters. In contrast, KLiNQ's student NNs are remarkably compact, with the larger FNN (used for qubits 2 and 3) containing just 6,754 parameters, and the smaller FNN (used for qubits 1, 4, and 5) comprising only 1,971 parameters. This corresponds to an NCR of 99.89% relative to the teacher neural network. Furthermore, compared to the baseline model with 1.63 million parameters, KLiNQ also achieves a remarkable 98.93% reduction in network size, highlighting the exceptional efficiency of its knowledge distillation strategy.

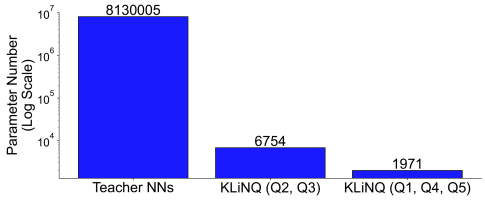


Fig. 5. Comparison of the number of network parameters for the teacher model and two distilled KLiNQ Student NNs (the plot is in log scale).

D. Latency and Resource Utilization

Tab. III summarizes the FPGA resource utilization and model latency for the subcomponents of the proposed design, including the MF for both I/Q components, the average and normalization module (AVG&NORM) and the neural network. The MF is time-multiplexed across all qubits, so its resource usage is counted only once for the entire design. In contrast, the AVG&NORM module and neural networks are instantiated separately for each qubit, with either the small or large configuration selected based on the setup requirements.

The overall latency is calculated by summing the latencies of the pipelined components. Since the logarithmic values of the samples for both longer traces (1 μs) and shorter acceptable traces (550 ns) are the same, the latency remains constant across all readout traces. Additionally, while the larger model requires more time for inference, it benefits from shorter averaging groups, leading to a reduced average processing time. As a result, both modules coincidentally produce the same execution latency of 32 ns at 100 MHz.

Resource utilization can be further optimized, as the current design prioritizes minimizing discrimination latency, sacrificing time-multiplexing to achieve faster classification. Therefore, trade-offs can be made to balance resource efficiency and performance depending on the specific latency requirements.

TABLE III
RESOURCE UTILIZATION AND LATENCY FOR INDIVIDUAL COMPONENTS

Components	LUT	FF	DSP	Latency (ns)
Shared Resources				
MF	27180 (6.39%)	24052 (2.83%)	375 (8.78%)	11
Per-Qubit Resources: Qubits 1, 4, 5				
AVG&NORM	17770 (4.2%)	11415 (1.35%)	0 (0%)	9
Network	8840 (2.08%)	6020 (0.71%)	55 (1.28%)	12
Per-Qubit Resources: Qubits 2, 3				
AVG&NORM	19600 (5%)	17500 (2%)	0 (0%)	6
Network	25882 (8.44%)	23172 (2.72%)	226 (5.29%)	15

VI. Discussion

This work proposes an independent readout scheme to enable mid-circuit measurements and error correction without synchronizing all readouts into a unified large neural network. Although it brings this advantage, it introduces significant challenges due to inherent quantum system characteristics. As highlighted in previous work by Benjamin et al. [3], the crosstalk in frequency-multiplexed readouts and correlations between qubits are a major source of error. Consequently, separating the readouts without accounting for inter-qubit influences inevitably leads to a reduction in fidelity. This trade-off is also evident in our results, where the independent readout approaches always underperform compared to the large network for the five-qubit system. Yet, this step represents a necessary direction for advancing systems toward FTQC.

On the other hand, this fidelity sacrifice underscores the complexity of mitigating crosstalk in independent readout schemes. Thus, future work will focus on integrating crosstalk information into the teacher model, thereby enabling the student model to learn and compensate for these effects. Additionally, exploring masking techniques within large networks could serve as another effective strategy to minimize these errors. By incorporating these considerations, we aim to achieve improved fidelity in individual readouts, ultimately advancing the practicality of error-corrected quantum computation.

VII. Conclusion

This paper introduces KLiNQ, a method leveraging knowledge distillation to train lightweight neural networks optimized for FPGA implementation. We propose an individual qubit-state readout discriminator capable of enabling mid-circuit measurements while maintaining high readout fidelity. Experimental results demonstrate that KLiNQ achieves an average qubit-state-discrimination accuracy of around 0.91, and sustains an accuracy above 0.9 even with a reduced 750 ns readout trace. The FPGA-based implementation features low hardware cost and an ultra-fast latency of 32 ns. Overall, our work highlights the potential of network compression techniques for efficient, hardware-based qubit-state readout.

REFERENCES

[1] Y. Zheng and K. Kanno, "Minimizing readout-induced noise for early fault-tolerant quantum computers," *Phys. Rev. Res.*, vol. 6, p. 023129, May 2024. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevResearch.6.023129

[2] T. Walter, P. Kurpiers, S. Gasparinetti, P. Magnard, A. Potočnik, Y. Salathé, M. Pechal, M. Mondal, M. Oppliger, C. Eichler, and A. Wallraff, "Rapid high-fidelity single-shot dispersive readout of superconducting qubits," *Phys. Rev. Appl.*, vol. 7, p. 054020, May 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.7.054020

[3] B. Lienhard, A. Vepsäläinen, L. C. Govia, C. R. Hoffer, J. Y. Qiu, D. Ristè, M. Ware, D. Kim, R. Winik, A. Melville, B. Niedzielski, J. Yoder, G. J. Ribeill, T. A. Ohki, H. K. Krovi, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "Deep-neural-network discrimination of multiplexed superconducting-qubit states," *Phys. Rev. Appl.*, vol. 17, p. 014024, Jan 2022. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.17.014024

[4] L. Stefanazzi, K. Treptow, N. Wilcer, C. Stoughton, C. Bradford, S. Uemura, S. Zorzetti, S. Montella, G. Cancelo, S. Sussman *et al.*, "The qick (quantum instrumentation control kit): Readout and control for qubits and detectors," *Review of Scientific Instruments*, vol. 93, no. 4, 2022.

[5] E. Magesan, J. M. Gambetta, A. D. Córcoles, and J. M. Chow, "Machine learning for discriminating quantum measurement trajectories and improving readout," *Phys. Rev. Lett.*, vol. 114, p. 200501, May 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.114.200501

[6] L. A. Martinez, Y. J. Rosen, and J. L. DuBois, "Improving qubit readout with hidden markov models," *Phys. Rev. A*, vol. 102, p. 062426, Dec 2020. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.102.062426

[7] C. A. Ryan, B. R. Johnson, J. M. Gambetta, J. M. Chow, M. P. da Silva, O. E. Dial, and T. A. Ohki, "Tomography via correlation of noisy measurement records," *Phys. Rev. A*, vol. 91, p. 022118, Feb 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.91.022118

[8] X. Guo, K. Qin, and M. Schulz, "Hisep-q: A highly scalable and efficient quantum control processor for superconducting qubits," in *2023 IEEE 41st International Conference on Computer Design (ICCD)*. Los Alamitos, CA, USA: IEEE Computer Society, nov 2023, pp. 86–93. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCD58817.2023.00023

[9] S. Maurya, C. N. Mude, W. D. Oliver, B. Lienhard, and S. Tannu, "Scaling qubit readout with hardware efficient machine learning architectures," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023. [Online]. Available: https://doi.org/10.1145/3579371.3589042

[10] P. K. Gautam, S. Kalipatnapu, U. Singhal, B. Lienhard, V. Singh, C. S. Thakur *et al.*, "Low-latency machine learning fpga accelerator for multi-qubit state discrimination," *arXiv preprint arXiv:2407.03852*, 2024.

[11] A. M. Gunyhó, S. Kundu, J. Ma, W. Liu, S. Niemelä, G. Catto, V. Vadimov, V. Vesterinen, P. Singh, Q. Chen *et al.*, "Single-shot readout of a superconducting qubit using a thermal detector," *Nature Electronics*, pp. 1–11, 2024.

[12] E. Bäumer, V. Tripathi, A. Seif, D. Lidar, and D. S. Wang, "Quantum fourier transform using dynamic circuits," *Phys. Rev. Lett.*, vol. 133, p. 150602, Oct 2024. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.133.150602

[13] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[14] J. H. Cho and B. Hariharan, "On the efficacy of knowledge distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 4794–4802.

[15] J. Gou, B. Yu, S. J. Maybank, and D. Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[16] AMD, "Zcu216 evaluation board user guide (ug1390)," https://docs.amd.com/r/en-US/ug1390-zcu216-eval-bd/Introduction, 2023.

BlasPart: A Deterministic Parallel Partitioner for Balanced Large-Scale Hypergraph Partitioning

Shengbo Tong, Chunyan Pei, and Wenjian Yu
Dept. Computer Science & Tech., BNRist, Tsinghua Univ., Beijing 100084, China

***Abstract*—Balanced hypergraph partitioning is a fundamental problem in applications like VLSI design, high-performance computing, etc. Nowadays, large-scale hypergraphs become more common due to the increasing complexity of modern systems. Thus, fast and high-quality deterministic partitioning algorithms are largely in demand. Regarding the quality of partitioning, balance is a critical concern when the number of partitions increases. In this paper, we propose BlasPart, a deterministic parallel algorithm for balanced large-scale hypergraph partitioning. BlasPart leverages a recursive multilevel bisection framework to achieve high-quality partitions while ensuring deterministic outcomes. A level-dependent balance constraint is also proposed to further improve the efficiency and effectiveness of the proposed partitioner. Extensive experiments, with comparisons to the state-of-the-art partitioners (hMETIS, BiPart, and Mt-KaHyPar-SDet), demonstrate that BlasPart achieves better balance and scalability while maintaining competitive partitioning quality and efficiency. BlasPart runs 3.33× faster than Mt-KaHyPar-SDet on average for a 4096-way partitioning task on six benchmarks.**

***Index Terms*—balanced partitioning, deterministic parallelism, large-scale hypergraph partitioning, multilevel bisection**

I. Introduction

Balanced hypergraph partitioning is a well-known NP-hard problem [1–3] and has been investigated in many research domains. The goal of balanced hypergraph partitioning is to divide the vertices of a hypergraph into blocks of approximately equal size while minimizing the number of hyperedges that span multiple blocks. Many heuristic algorithms [4–8] have been proposed and achieved relatively good performance.

In the field of Electronic Design Automation (EDA), balanced hypergraph partitioning has been widely used in various scenarios for VLSI circuit design [9, 10]. The growing demand in the digital electronics market and intense competition among enterprises drive the need for highly integrated Systems on Chip (SoC), i.e. VLSI circuits containing billions of elements. The circuit with the connections among elements can be naturally modeled as a hypergraph. Thus, the partition of circuits is equivalent to a hypergraph partitioning problem, which is considered in various stages of VLSI design, e.g. logical design, physical design, and circuit simulation, etc [11, 12]. Specifically, the emerging cloud-based EDA, parallel processing, 3D integration, and FPGA-based emulation request the support of balanced hypergraph partitioning, so as to efficiently balance workloads and minimize interconnections. Hypergraph partitioning also has numerous applications in many other areas besides EDA, including linear algebra [13], boolean satisfiability [14], as well as storage sharding [15].

As the scale of the problem grows, the need for fast, high-quality partitioning algorithms for large-scale hypergraphs becomes more urgent. Parallelism is a common approach to accelerate partitioning, and several parallel hypergraph partitioners [16–22] have been developed. However, parallelism often introduces non-determinism, which is undesirable in some scenarios like VLSI design because it may result in redoing all the following time-consuming operations. Moreover, larger hypergraphs mean more partitions as well, which makes achieving good balance more challenging.

In response to the aforementioned challenges, we propose a deterministic parallel algorithm for Balanced large-scale hypergraph Partitioning, named **BlasPart**.The main contributions and results of this work are as follows.

1) A recursive multilevel bisection framework is proposed for parallel k-way hypergraph partitioning. It combines comprehensive treatments for efficient serial and parallel bisection of hypergraphs, so as to achieve high-quality partitions while ensuring deterministic outcomes.
2) A level-dependent balance constraint for bisection tasks is proposed for the multilevel bisection framework. It enables more efficient and effective hypergraph partitioning while achieving well-balanced results or meeting the user-specified imbalance ratio for large-scale hypergraph partitioning (with a large value of k as well).
3) The proposed techniques are integrated into the partitioner named BlasPart. Extensive experiments on large hypergraphs show that BlasPart achieves better balance and scalability while maintaining competitive partitioning quality and efficiency, compared to the state-of-the-art parallel deterministic partitioner Mt-KaHyPar-SDet [22].

II. Preliminaries

A. Hypergraph Partitioning

A hypergraph is a generalization of a graph where each edge can connect more than two vertices. In a hypergraph $H = (V, E)$, V is the set of vertices and E is the set of nonempty subsets of V called hyperedges. The goal of balanced hypergraph partitioning is to divide the vertices into k blocks (partitions) such that the number of vertices in each block is approximately equal, while minimizing the number of hyperedges across at least two blocks (`cut-net`). The

This work is supported by Beijing Natural Science Foundation (No. Z230002). The executable program of BlasPart and running instructions are available at https://numbda.cs.tsinghua.edu.cn/download.html.

balance of the result of k-way hypergraph partitioning can be measured by the imbalance ratio:

$$\delta = \max_{1 \le i \le k} \frac{||V_i| - \frac{|V|}{k}|}{\frac{|V|}{k}}, \tag{1}$$

where $|V_i|$ is the number of vertices in block i, and $|V|$ is the total number of vertices. Some hypergraph partitioners, like hMETIS [4] and TritonPart [11] take the following constraint

$$(1-\epsilon)\frac{|V|}{k} \le |V_i| \le (1+\epsilon)\frac{|V|}{k},\ \forall\ 1 \le i \le k\ , \tag{2}$$

into the problem formulation. Thus, they can produce well-balanced partitioning results, guaranteeing that imbalance ratio $\delta \le \varepsilon$. Some other partitioners like PaToH [5] and KaHyPar [8] only take the upper bound in (2) as a constraint, which makes their result's imbalance ratio possibly very large.

The objective function of the hypergraph partitioning is generally called *cutsize*, which may be different from `cut-net` in some scenarios. For example, for the hypergraph partitioning aimed at the applications of MFS-based emulation or parallel computing, slightly different metrics like `connectivity` [23] and `soed` [24] are used as the cutsize for minimization. In this work, we use the traditional `cut-net` definition [25], where each cut hyperedge only contributes 1 to the cutsize. Moreover, we pursue the balanced partitioning result with a small imbalance ratio, as it suits more applications, like workload balance in large-scale parallel computing.

B. Related Work

1) Serial Hypergraph Partitioning: There are a lot of serial hypergraph partitioning algorithms mostly based on heuristic methods. One type of partitioner employs the multilevel paradigm, which consists of coarsening, initial partitioning, and refinement, and was first proposed by hMETIS [4] and followed by PaToH [5] and UMPa [6]. KaHyPar [8] further improves the partitioning results by introducing preprocessing and flow-based techniques. TritonPart [11] is a more recent open-source, constraints-driven partitioning tool for VLSI physical design. Another type of partitioner is based on spectral methods. HyperSF [26] and HyperEF [27] achieve more efficiency in the coarsening phase without losing the key structural properties of the original hypergraph. SpecPart [28] and K-SpecPart [29] can produce vertex embeddings and further reduce the cutsize of a given partitioning result, but at the cost of higher runtime and complexity.

2) Parallel Hypergraph Partitioning: As the scale of hypergraphs grows, the need for faster partitioning algorithms has led to the development of many parallel partitioners, like Zoltan [16], Parkway [17], Social Hash Partitioner [15] and Mt-KaHyPar series [19–21]. However, none of them can guarantee deterministic results, which is a critical requirement in some domains, like VLSI circuit design.

BiPart [18] is the first deterministic parallel hypergraph partitioner. It introduces parallelism in all three phases of the multilevel paradigm, and achieves good partitioning quality and runtime. However, BiPart fails to control the vertex weights in the coarsening phase and ignores the up-to-date gains in the refinement phase, which leads to rather fluctuating results against its hyperparameters.

Mt-KaHyPar-SDet [22] is the deterministic version of Mt-KaHyPar, which outperforms BiPart regarding quality, runtime, scalability, and does not lose too much performance compared to its non-deterministic versions. It uses direct k-way partitioning and has better control over the vertex weights and partition balance throughout the whole partitioning process. The main drawback of Mt-KaHyPar-SDet is its parallel refinement method based on label propagation [30] disables its opportunity to consider the lower bound of the balance constraint in (2), which is not acceptable in some scenarios.

III. The Proposed Method

In this section, we present the techniques in BlasPart. Firstly, a recursive multilevel bisection framework is proposed to perform k-way partitions with deterministic results. Within it, efficient techniques for bisection are adopted based on whether a task is executed serially or parallelly. Then, a level-dependent strategy is proposed to enforce the balance constraint to each bisection task. Finally, we analyze the scalability of BlasPart with respect to the number of partitions and threads.

A. Overall Framework

Although [31] states that recursive bisection often achieves substantially worse results than direct k-way, it becomes different under the parallel environment, especially when we are dealing with the problem of partitioning a large-scale hypergraph into a large number of balanced blocks. The reasons lie in both the initial and refinement phases.

In the initial phase, we need to find a balanced partition of the hypergraph, but underload is very likely to happen if we keep pursuing quality, and many algorithms actually still use recursive bisection here (or they do not focus on underload). Besides, during the k-way refinement, the definition of `cut-net` is unaware of the number of blocks a cut hyperedge intersects, and can not be eliminated unless all the vertices in the hyperedge are assigned to the same partition, which is usually very difficult when k is large. By contrast, recursive bisection does not have either of these problems.

Moreover, recursive bisection can be easily parallelized by first partitioning the hypergraph into two blocks, and then using different threads to recursively partition each block into two blocks until the desired number of blocks is reached. This is the main idea behind BlasPart, which is shown in Fig. 1.

To partition a hypergraph into k blocks using p threads (suppose $k > p$ and is the power of 2 for simplicity), BlasPart utilizes the divide and conquer strategy and produces a binary tree that can be separated into two main stages. The first stage includes the first $\log(p)$ layers of the tree, where each bisection task uses multiple threads. The second stage includes the rest $\log(k/p)$ layers of the tree, where each bisection task uses only one thread and is executed serially. The last layer consists of $k/2$ bisection tasks that eventually output the k partitions. This framework involves both serial and parallel bisection algorithms, which are detailed in the following part.

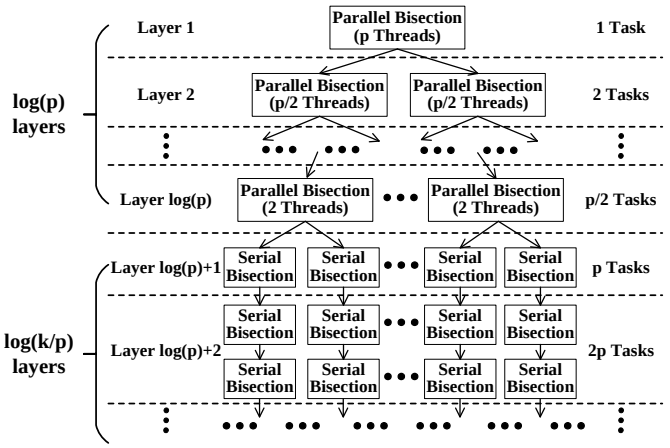


Fig. 1: The overall framework of BlasPart, for parallelly partitioning a hypergraph into k blocks using p threads.

B. Deterministic Bisection Algorithms

1) Serial: For the serial algorithm, we mostly follow the workflow of hMETIS [4]. In the coarsening phase, we use the first-choice (FC) scheme [32] by applying the heavy-edge score function to cluster highly connected vertices. In the initial partitioning phase, we use the greedy hypergraph growing (GHG) algorithm [33]. We combine the two gain functions (max-pin and max-net) with three vertex selecting strategies (sequential, global, round-robin) [8, 34] to generate multiple initial partitions, from which a few best will be eventually selected. In each level of the refinement phase, solutions with a cutsize 10% worse than the current smallest will be discarded. We use the greedy refinement [32] instead of the Fiduccia-Mattheyses (FM) algorithm [35], because during the whole recursive process, we will deal with hypergraphs of different sizes and structures, to which the speed of FM algorithm is quite sensitive. This is especially important because now we are concerned about the critical path of the whole partitioning process when a steady runtime is more important. The serial algorithm does not involve deterministic issues.

2) Parallel: For the parallel algorithm, because the serial coarsening phase is very time-consuming and lacks parallelism when the hypergraph is large, we replace it with the parallel one in [22]. This fully exploits the parallelism on the first $\log(p)$ layers of the binary tree in Fig. 1, while providing deterministic results at the same time. Because the hypergraphs in the initial partitioning phase are much smaller than the original one, we still use the serial GHG algorithm and utilize the parallelism in finding multiple diverse initial solutions, just like [22]. When collecting the results from the parallel bisection tasks, we sort them by their task ID before filtering to ensure determinism as well. Different from [22], we also apply this principle in the refinement phase. We use the same greedy refinement algorithm as the serial one, and refine different solutions in parallel. This enables the algorithm to keep more diversity to compensate for the quality loss in the coarsening phase. It also avoids the loss in the lack of up-to-date gains during the vertex moves. Determinism is also guaranteed because the order of the partitions is fixed.

C. The Balance Constraint for Bisection Tasks

When partitioning a large hypergraph, especially using the approach of recursive bisection, achieving good balance is a challenge. Some recursive bisection algorithms like hMETIS [4] and BiPart [18] only use a fixed value of imbalance factor, i.e. ϵ in (2), to enforce the balance constraint for every bisection task. In this case, in order to fulfill the overall balance constraint (2) for the k-way partitioning, the imbalance factor ϵ_i for solving the i-th layer bisection problem is set as:

$$\begin{aligned}\epsilon_i &= \min\{(1+\epsilon)^{\frac{1}{\log(k)}} - 1, 1 - (1-\epsilon)^{\frac{1}{\log(k)}}\} \\ &= (1+\epsilon)^{\frac{1}{\log(k)}} - 1, \qquad\qquad \forall\ 1 \le i \le k-1.\end{aligned} \tag{3}$$

However, this is often too strict for large k, leading to the actual imbalance ratio δ of the result being much smaller than the imbalance factor ϵ. As a result, this approach unavoidably causes a decrease in the quality of partitioning.

In this work, we propose a different strategy for setting the imbalance factor for the recursive bisection approach, which produces the partitioning result with better quality and good balance at the same time. Firstly, we see that (3) infers that at the i-th layer of bisection, the enforced constraint includes:

$$|V_j^{(i)}| \le (1+\epsilon)^{\frac{1}{\log(k)}} \cdot \frac{|V^{(i)}|}{2},\ j = 1, 2, \tag{4}$$

where $|V^{(i)}|$ denotes the number of nodes of the hypergraph for performing bisection. At the very bottom layer, i.e. layer $\log(k)$ in Fig. 1, the upper bound in (4) for a block's number of nodes would be about $(1+\epsilon)^{\frac{1}{\log(k)}} \frac{|V|}{2^{\log(k)}} = (1+\epsilon)^{\frac{1}{\log(k)}} \frac{|V|}{k}$, if the partitions are all well balanced and $|V^{(i)}| \approx \frac{|V|}{2^{(i-1)}}$. This upper bound is too small, compared to that we want to enforce, i.e. $(1+\epsilon)\frac{|V|}{k}$ in (2). Therefore, our idea is to relieve this upper bound, and make it depend on the layer of bisection. We change the upper bound of constraint to:

$$|V_j^{(i)}| \le (1+\epsilon)^{\frac{1}{\log(k)-i+1}} \cdot \frac{|V|}{2^i},\ j = 1, 2, \tag{5}$$

for bisection tasks of the i-th layer. In this way, at the bottom layer (i.e. $i = \log(k)$) the upper bound is the same as that in (2). It guarantees that the overall balance constraint on the upper bound is satisfied. At the upper layers of bisection, the constraint becomes gradually tight; at the first layer, it is the same as (4), i.e. that used in the conventional approach.

Similar analysis and manipulation are made on the lower bound of the constraint. As a result, we set the lower bound of the balance constraint for the i-th layer of bisection to:

$$|V_j^{(i)}| \ge (1-\epsilon)^{\frac{1}{\log(k)-i+1}} \cdot \frac{|V|}{2^i},\ j = 1, 2. \tag{6}$$

With the constraints (5) and (6) set for the bisection tasks, the final partitioning result will satisfy the balance constraint (2) if all the bisection problems can be solved. The exception occurs when $|V^{(i)}|$ in (4) is far from $\frac{|V|}{2^{(i-1)}}$, which may cause the absolute imbalance margins in (5) and (6) making the optimization problem infeasible. We have made extensive experiments and found that this situation happens in very low probability. If it does happen, which only includes two

types of situations: the average number of vertices $\frac{|V_1^{(i)}|+|V_2^{(i)}|}{2}$ is smaller than the lower bound, or bigger than the upper bound, we will force an even balance from this task to all its descendants to reduce the violation as much as possible. The results show that we can control the imbalance in an acceptable range. If we really need a strict balance solution, we can just set all upper and lower bounds according to (3).

We also utilize some other strategies to further optimize the balance and reduce the risk of unsolvability. One hidden strength of the recursive bisection is that after each partitioning task, the number of hyperedges decreases because the cut hyperedges in the last layer are discarded since they do not contribute to the cutsize of its descendants, according to the definition of `cut-net`. As a result, as the partitioning goes deeper, many isolated vertices emerge. They do not belong to any hyperedges and can be easily assigned to any partition without affecting the cutsize. Therefore, we ignore these vertices during the whole bisection process, and after the rest vertices are assigned, we assign these vertices to the partition with the smaller size. This is a very simple yet effective way to balance the partition size. Another strategy is to consider balance as a criterion in filtering the solutions in both the initial and refinement phases. When two solutions have the same cutsize, we break the tie by choosing the one with better balance. This criterion can also be considered during the greedy refinement phase, where we can choose the move that leads to better balance when the gain is the same, and accept moves with zero gain but can improve the balance.

D. Scalability Analysis

BlasPart aims at partitioning large hypergraphs into a large number of blocks in parallel. So, both the scalabilities in terms of the number of blocks and the number of threads are of concern. Separating hypergraphs according to the bisection results occupies very little time in the whole partitioning process, so we do not consider it in the following analysis.

When the number of blocks (k) increases, we analyze the runtime (t) breakdown by separating t according to the correlation with k:

$$t = f(k) + C, \tag{7}$$

where $f(k)$ varies along with k, and C remains constant when k changes. The reason for the poor scalability of BiPart is that its C equals 0 and the high scaling rate of $f(k)$ because it uses all threads in each bisection task.

Mt-KaHyPar-SDet improves this by using the direct k-way partitioning, which avoids the repetitive calling of the partitioning algorithm. Comparing the three phases of the multilevel paradigm, it is worth noting that when doing the k-way partitioning, the coarsening phase is obviously different from the other two, because its runtime does not depend on k. k will only influence the maximum allowed cluster size, but this will only happen at the last few levels when the coarsened hypergraph is small enough and does not consume much time. Based on the analysis above, C of Mt-KaHyPar-SDet is actually the coarsening runtime, and $f(k)$ consists of the initial partitioning and refinement runtime. The invariant coarsening runtime C and lower scaling rate of $f(k)$ are the main reasons for its better scalability than BiPart.

Compared to Mt-KaHyPar-SDet, the special point of BlasPart is that its C is actually the whole first stage (the first $\log(p)$ layers in Fig. 1), because as k increases, the only differences in the first stage are the upper and lower bounds, which will hardly affect its runtime. Considering the second stage, BlasPart has an even lower scaling rate of $f(k)$ than Mt-KaHyPar-SDet, because although the number of tasks doubles in each layer, the sizes of the hypergraphs also decrease by half respectively. Considering the hypergraphs involved in this stage are more than p times smaller than the original one, the total runtime of each layer increases very slowly, and may even result in a decrease of the total runtime on some hypergraphs when k doubles. Another incidental advantage is that other optimizations like V-cycle [4] and COCP [11] can be added in each bisection task in the last few layers to further improve the quality with little time cost, but we do not further discuss it in this paper due to space limitation.

According to Fig. 1, each bisection task in the second stage uses one thread, which involves no communication cost. Its runtime also differs little from each other, because the hypergraphs in here are small. Therefore, the scalability of BlasPart in terms of the number of threads (p) in this stage is good. However, the first stage is usually the bottleneck, especially when the number of blocks is relatively small. Because the runtime difference between tasks in the first stage is quite sensitive to the input hypergraph, which further affects the runtime critical path of the binary tree, the overall scalability of BlasPart (in terms of p) may vary on different benchmarks. Nevertheless, efficient techniques that are insensitive to the hypergraph structure have been selected (as stated in Section III-B), to avoid this problem as much as possible.

IV. Experimental Results

In this section, we evaluate the proposed BlasPart by comparing it with state-of-the-art partitioners on two sets of benchmarks. BlasPart is implemented in C++ and utilizes Intel's tbb library for parallelization. We first compare BlasPart with the newest hMETIS (v2.0 pre1[1]) which supports direct k-way partition and the complete balance constraint (2). hMETIS is a serial-computing partitioner, the SOTA of partitioning quality. Then, we compare BlasPart with the parallel partitioners Mt-KaHyPar-SDet[2] [22] and BiPart[3] [18] on larger benchmarks. All experiments are conducted on a Ubuntu 22.04 Linux server with AMD EPYC 7V12 64-Core CPU and 512GB RAM.

A. Comparison with hMETIS

The comparison with hMETIS is made on the Titan23 benchmark suite [36], which contains hypergraphs from 22 circuits covering a wide range of applications. For hMETIS, all

[1]http://glaros.dtc.umn.edu/gkhome/views/metis

[2]https://github.com/kahypar/mt-kahypar

[3]https://github.com/IntelligentSoftwareSystems/Galois/tree/master/lonestar/analytics/cpu/bipart

parameters are set to their default values. Because $nruns = 10$ in hMETIS, we also run BlasPart 10 times with different seeds as well, and calculate the best cutsize and average runtime of both partitioners. Fig. 2 shows the results for $\epsilon = 0.1$ and $k = 16$. BlasPart uses 16 threads and its cutsize and runtime are normalized by those obtained by hMETIS, respectively.

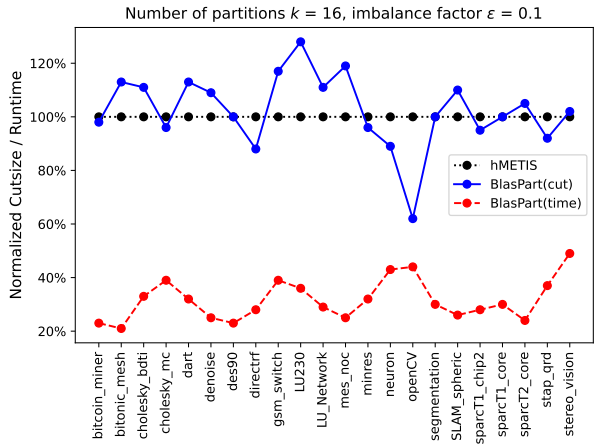


Fig. 2: Normalized cutsize and runtime of BlasPart on the Titan23 benchmark suite, with respect to those of hMETIS.

From Fig. 2, we can see that BlasPart outperforms hMETIS on cutsize on 9 of the 22 benchmarks. After statistics, we find out that the cutsize of BlasPart is just 2% larger than that of hMETIS on average, which shows that our parallel BlasPart maintains competitive partitioning quality compared to the k-way serial partitioner hMETIS. As for the runtime, although all the benchmarks are relatively small (with 90K-1M vertices), which indicates the optimization space is quite limited, BlasPart still exhibits a 3.33× speedup on average, and a maximum 4.70× speedup (on the *bitconic_mesh*). Besides, both the results of hMETIS and BlasPart fulfill the balance constraint with the measured imbalance ratio δ all within 0.1. This validates the balance strategy proposed in Section III-C.

B. Comprasion with Parallel Partitioners on Large Cases

TABLE I: Characteristics of six large benchmarks.

Benchmark	#Vertices	#Hyperedges	#Pins
WB	9,845,725	6,920,306	57,156,537
NLPK	3,542,400	3,542,400	96,845,792
SAT	13,378,010	521,147	39,203,144
TITAN	1,089,284	1,448,151	4,557,197
DAC	1,360,217	1,340,418	4,931,418
IND	10,904,226	1,327,578	21,535,851

To comprehensively evaluate the performance of BlasPart on large-scale hypergraphs, we conduct the following experiments on six large benchmarks listed in Table I. The number of pins is the sum of the cardinalities of all hyperedges. WB and NLPK are from the SuiteSparse Matrix Collection [37]. SAT is from the 2014 SAT Competition [14]. TITAN and DAC are two VLSI instances from the Titan23 benchmark suite [36] and DAC 2012 Routability-Driven Placement Contest [38], respectively. They all are the largest hypergraphs in the corresponding benchmark suite. IND is a large hypergraph we obtained from an industrial circuit design. The cases WB, NLPK, and SAT were also used in the experiments of BiPart [18] and Mt-KaHyPar-SDet [22]. In our experiment, we set $\epsilon = 0.1$ and vary k from 16 to 4096, and all the partitioners run with 16 threads and their default parameter settings.

We first compare BlasPart with Mt-KaHyPar-SDet. The imbalance ratios (δ calculated with (1)) of their results are listed in Table II, while the results of cutsize and runtime are shown in Fig. 3. From Table II we can see that Mt-KaHyPar-SDet largely violates the balance constraint in most cases, because it only considers the upper bound of the partition size. In contrast, BlasPart fulfills the balance constraint in most cases. The only exception is SAT, where the imbalance ratio of BlasPart's result is 0.179 when $k = 1024$. It still corresponds to a partitioning with a relatively good balance. The results in Table II validate the proposed balance strategy, which relaxes the balance constraint but still achieves results with good balance on large-scale hypergraph partitioning.

In Fig. 3, the normalized cutsizes obtained with BlasPart are displayed with those of Mt-KaHyPar-SDet set as the baseline. We can see that BlasPart achieves similar cutsizes to Mt-KaHyPar-SDet and overall outperforms it when k is large enough. In terms of runtime, although BlasPart generally consumes more time than Mt-KaHyPar-SDet when k takes small values like 16, 32, and 64, it gradually becomes more efficient than the latter when k increases. This is consistent with our analysis in Section III-D. The runtime of Mt-KaHyPar-SDet grows approximately exponentially with $\log(k)$ (linearly with k), while the runtime of BlasPart grows nearly linearly with $\log(k)$ (logarithmically with k). The steady runtime C of the first stage and fast speed in the second stage in BlasPart are the main reasons. It eventually reaches a maximum of 4.69× and an average of 3.33× speedup over Mt-KaHyPar-SDet on the six benchmarks when $k = 4096$. In general, for the balanced large-scale hypergraph partitioning, BlasPart achieves better balance and scaling of runtime while maintaining competitive partitioning quality compared to Mt-KaHyPar-SDet.

BlasPart is also compared with another deterministic parallel partitioner BiPart, which can support the complete balanced constraint (2). The experiments in [22] demonstrate that Mt-KaHyPar-SDet has better performance than BiPart in terms of runtime and cutsize. Our experiments confirm this. The detailed comparison results on case DAC are listed in Table III as an example. From it, we can see that BlasPart achieves significant improvements on the cutsize and runtime, compared to BiPart. As for the imbalance ratio, both partitioners satisfy the constraint with $\epsilon = 0.1$. Because BiPart employs (3) for setting the constraint for each bisection problem, it obviously degrades its partition quality due to this too-strict setting.

Lastly, an experiment is done to show the parallel scalability of BlasPart. We set $\epsilon = 0.1$ and $k = 1024$ on all benchmarks, and Fig. 4 shows the speedup of BlasPart along with the number of threads selected from $\{1, 2, 4, 8, 16, 32\}$. As we have mentioned in Section III-D, the scaling performance of BlasPart may vary on different benchmarks. BlasPart achieves the best scaling on NLPK, with the parallel speedup of 6.25× using 16 threads, but performs relatively poorly on SAT

TABLE II: Imbalance ratios of k partitions producted by Mt-KaHyPar-SDet and BlasPart on large cases (setting $\epsilon = 0.1$).

k	WB		NLPK		SAT		TITAN		DAC		IND	
	Mt-KaHyPar-SDet	BlasPart	Mt-KaHyPar-SDet	BlasPart	Mt-KaHyPar-SDet	BlasPart	Mt-KaHyPar-SDet	BlasPart	Mt-KaHyPar-SDet	BlasPart	Mt-KaHyPar-SDet	BlasPart
16	0.100	**0.092**	0.277	**0.074**	0.209	**0.091**	0.152	**0.100**	0.244	**0.099**	0.226	**0.081**
32	**0.100**	**0.100**	0.498	**0.080**	0.231	**0.093**	0.277	**0.092**	0.285	**0.097**	0.196	**0.077**
64	0.100	**0.099**	0.552	**0.094**	0.231	**0.098**	0.261	**0.095**	0.390	**0.098**	0.381	**0.087**
128	0.100	**0.098**	0.378	**0.089**	0.449	**0.100**	0.515	**0.098**	0.311	**0.100**	0.294	**0.086**
256	0.100	**0.099**	0.566	**0.097**	0.669	**0.131**	0.644	**0.100**	0.370	**0.099**	0.505	**0.098**
512	0.100	**0.099**	0.803	**0.095**	0.545	**0.100**	0.556	**0.100**	0.394	**0.099**	0.379	**0.091**
1024	**0.100**	**0.100**	0.998	**0.100**	0.449	**0.179**	0.634	**0.099**	0.365	**0.100**	0.445	**0.095**
2048	**0.100**	**0.100**	0.947	**0.100**	0.473	**0.173**	0.551	**0.098**	0.418	**0.100**	0.424	**0.098**
4096	**0.100**	**0.100**	0.763	**0.100**	0.480	**0.139**	0.568	**0.098**	0.345	**0.100**	0.421	**0.099**

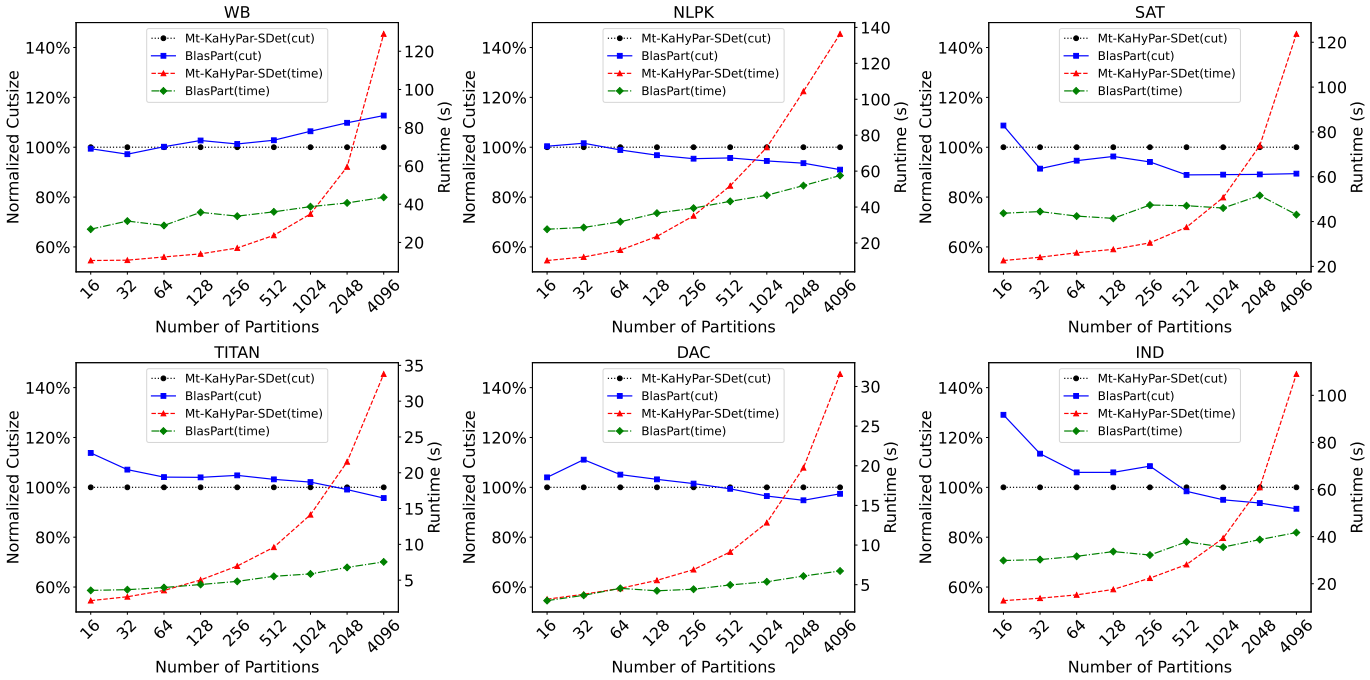


Fig. 3: Normalized cutsize and actual runtime for Mt-KaHyPar-SDet and BlasPart on the large benchmarks (16 threads).

TABLE III: Performance of BiPart and BlasPart on the DAC benchmark, setting $\epsilon = 0.1$ (16 threads).

k		16	32	64	128	256	512	1024	2048	4096
BiPart	cutsize	55049	83633	119424	142369	168388	196320	239393	298877	343686
	runtime(s)	10.17	14.29	22.22	29.82	42.68	54.72	66.42	75.74	97.70
	imbalance	0.087	0.061	0.092	0.080	0.085	0.067	0.086	0.081	0.078
BlasPart	cutsize	30219	47714	65265	83856	98104	115327	134225	154135	174936
	runtime(s)	2.99	3.66	4.54	4.23	4.43	4.97	5.37	6.09	6.73
	imbalance	0.099	0.100	0.099	0.098	0.099	0.099	0.100	0.100	0.100

because of its great disparity in the number of vertices and hyperedges. The parallel speedup of BlasPart is also limited on TITAN and DAC, because of their relatively small sizes.

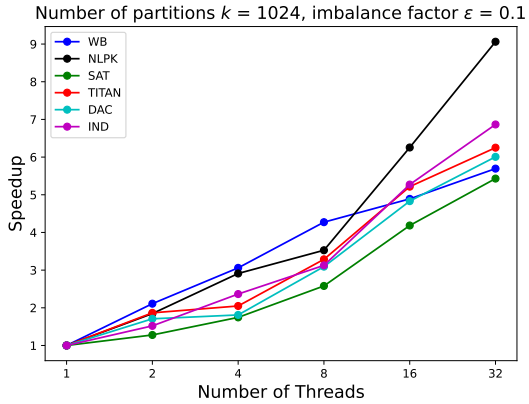


Fig. 4: Parallel speedup of BlasPart on different benchmarks.

V. Conclusion

In this paper, we propose a parallel hypergraph partitioner BlasPart, which is designed for partitioning a large-scale hypergraph into a large number of balanced blocks. BlasPart employs a two-stage recursive bisection approach, which can give partitioning results with good balance and high quality. We introduce a novel strategy for setting the balance constraint in the recursive bisection, which is more relaxed than the conventional approach and can adapt to different hypergraphs. We also utilize some other strategies to further optimize the balance and reduce the risk of unsolvability. Experimental results show that BlasPart achieves competitive partitioning quality compared to the state-of-the-art partitioners, and it is more efficient and scalable on large-scale hypergraph partitioning. In the future, we will extend BlasPart to support any number of partitions and introduce more techniques in the whole framework to further improve the partitioning quality.

REFERENCES

[1] D. S. Johnson and M. R. Garey, *Computers and intractability: A guide to the theory of NP-completeness*. WH Freeman, 1979.

[2] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures*, 2004, pp. 120–124.

[3] T. Lengauer, *Combinatorial algorithms for integrated circuit layout*. Springer Science & Business Media, 2012.

[4] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel hypergraph partitioning: Applications in VLSI domain," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 69–79, 1999.

[5] Ü. V. Çatalyürek and C. Aykanat, "PaToH (partitioning tool for hypergraphs)." 2011.

[6] U. V. Catalyurek, M. Deveci, K. Kaya, and B. Uçar, "UMPa: A multi-objective multi-level partitioner for communication minimization," *Graph partitioning and graph clustering*, vol. 588, p. 53, 2013.

[7] H. Leitgeb, "HYPE: A system of hyperintensional logic (with an application to semantic paradoxes)," *Journal of Philosophical Logic*, vol. 48, no. 2, pp. 305–405, 2019.

[8] S. Schlag, T. Heuer, L. Gottesbüren, Y. Akhremtsev, C. Schulz, and P. Sanders, "High-quality hypergraph partitioning," *ACM Journal of Experimental Algorithmics*, vol. 27, pp. 1–39, 2023.

[9] C. J. Alpert, "The ISPD98 circuit benchmark suite," in *Proceedings of the 1998 international symposium on Physical design*, 1998, pp. 80–85.

[10] A. E. Caldwell, A. B. Kahng, A. A. Kennings, and I. L. Markov, "Hypergraph partitioning for VLSI CAD: methodology for heuristic development, experimentation and reporting," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 349–354.

[11] I. Bustany, G. Gasparyan, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "An open-source constraints-driven general partitioning multi-tool for VLSI physical design," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[12] B. Li, S. Bi, H. You, Z. Qi, G. Guo, R. Sun, and Y. Zhang, "MaPart: An efficient multi-FPGA system-aware hypergraph partitioning framework," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[13] U. V. Catalyurek and C. Aykanat, "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication," *IEEE Transactions on parallel and distributed systems*, vol. 10, no. 7, pp. 673–693, 1999.

[14] A. Belov, D. Diepold, M. Heule, and M. Järvisalo, "The SAT competition 2014," in *Proc. SAT COMPETITION*, 2014, p. 45.

[15] I. Kabiljo, B. Karrer, M. Pundir, S. Pupyrev, A. Shalita, A. Presta, and Y. Akhremtsev, "Social hash partitioner: a scalable distributed hypergraph partitioner," *arXiv preprint arXiv:1707.06665*, 2017.

[16] K. D. Devine, E. G. Boman, R. T. Heaphy, R. H. Bisseling, and U. V. Catalyurek, "Parallel hypergraph partitioning for scientific computing," in *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. IEEE, 2006, pp. 10–pp.

[17] A. Trifunović and W. J. Knottenbelt, "Parallel multilevel algorithms for hypergraph partitioning," *Journal of Parallel and Distributed Computing*, vol. 68, no. 5, pp. 563–581, 2008.

[18] S. Maleki, U. Agarwal, M. Burtscher, and K. Pingali, "Bipart: a parallel and deterministic hypergraph partitioner," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021, pp. 161–174.

[19] L. Gottesbüren, T. Heuer, P. Sanders, and S. Schlag, "Scalable shared-memory hypergraph partitioning," in *23rd Workshop on Algorithm Engineering and Experiments (ALENEX 2021)*. SIAM, 2021, pp. 16–30.

[20] L. Gottesbüren, T. Heuer, P. Sanders, and S. Schlag, "Shared-memory n-level hypergraph partitioning," in *24th Workshop on Algorithm Engineering and Experiments (ALENEX 2022)*. SIAM, 01 2022.

[21] L. Gottesbüren, T. Heuer, and P. Sanders, "Parallel flow-based hypergraph partitioning," in *20th International Symposium on Experimental Algorithms (SEA 2022)*, vol. 233. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, pp. 5:1–5:21.

[22] L. Gottesbüren and M. Hamann, "Deterministic parallel hypergraph partitioning," in *European Conference on Parallel Processing*. Springer, 2022, pp. 301–316.

[23] M. Deveci, K. Kaya, B. Uçar, and Ü. V. Çatalyürek, "Hypergraph partitioning for multiple communication cost metrics: Model and methods," *Journal of Parallel and Distributed Computing*, vol. 77, pp. 69–83, 2015.

[24] A. Trifunovic and W. J. Knottenbelt, "Par*k*way 2.0: A parallel multilevel hypergraph partitioning tool," in *International Symposium on Computer and Information Sciences*. Springer, 2004, pp. 789–800.

[25] C. J. Alpert and A. B. Kahng, "Recent directions in netlist partitioning: A survey," *Integration*, vol. 19, no. 1-2, pp. 1–81, 1995.

[26] A. Aghdaei, Z. Zhao, and Z. Feng, "HyperSF: Spectral hypergraph coarsening via flow-based local clustering," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–9.

[27] A. Aghdaei and Z. Feng, "HyperEF: Spectral hypergraph coarsening by effective-resistance clustering," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[28] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "SpecPart: A supervised spectral framework for hypergraph partitioning solution improvement," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.

[29] I. Bustany, A. B. Kahng, I. Koutis, B. Pramanik, and Z. Wang, "K-SpecPart: Supervised embedding algorithms and cut overlay for improved hypergraph partitioning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

[30] H. Meyerhenke, P. Sanders, and C. Schulz, "Parallel graph partitioning for complex networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2625–2638, 2017.

[31] H. D. Simon and S.-H. Teng, "How good is recursive bisection?" *SIAM Journal on Scientific Computing*, vol. 18, no. 5, pp. 1436–1445, 1997.

[32] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th annual ACM/IEEE design automation conference*, 1999, pp. 343–348.

[33] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[34] T. Heuer, "Engineering initial partitioning algorithms for direct k-way hypergraph partitioning," Ph.D. dissertation, Karlsruher Institut für Technologie (KIT), 2015.

[35] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Papers on Twenty-five years of electronic design automation*, 1988, pp. 241–247.

[36] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz, "Titan: Enabling large and complex benchmarks in academic CAD," in *2013 23rd International Conference on Field programmable Logic and Applications*. IEEE, 2013, pp. 1–8.

[37] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, vol. 38, no. 1, pp. 1–25, 2011.

[38] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei, "The DAC 2012 routability-driven placement contest and benchmark suite," in *Proceedings of the 49th Annual Design Automation Conference*, 2012, pp. 774–782.

BoolE: Exact Symbolic Reasoning via Boolean Equality Saturation

Jiaqi Yin*, Zhan Song*, Chen Chen, Qihao Hu, Cunxi Yu
University of Maryland, College Park
College Park, US
{jyin629,cunxiyu}@umd.edu

***Abstract*—Boolean symbolic reasoning for gate-level netlists is a critical step in verification, logic and datapath synthesis, and hardware security. Specifically, reasoning datapath and adder tree in bit-blasted Boolean networks is particularly crucial for verification and synthesis, and challenging. Conventional approaches either fail to accurately (exactly) identify the function blocks of the designs in gate-level netlist with structural hashing and symbolic propagation, or their reasoning performance is highly sensitive to structure modifications caused by technology mapping or logic optimization. This paper introduces BoolE, an exact symbolic reasoning framework for Boolean netlists using equality saturation. BoolE optimizes scalability and performance by integrating domain-specific Boolean ruleset for term rewriting. We incorporate a novel extraction algorithm into BoolE to enhance its structural insight and computational efficiency, which adeptly identifies and captures multi-input, multi-output high-level structures (e.g., full adder) in the reconstructed e-graph.**

Our experiments show that BoolE surpasses state-of-the-art symbolic reasoning baselines, including the conventional functional approach (ABC) and machine learning-based method (Gamora). Specifically, we evaluated its performance on various multiplier architecture with different configurations. Our results show that BoolE identifies $3.53\times$ and $3.01\times$ more exact full adders than ABC in carry-save array and Booth-encoded multipliers, respectively. Additionally, we integrated BoolE into multiplier formal verification tasks, where it significantly accelerates the performance of traditional formal verification tools using computer algebra, demonstrated over four orders of magnitude runtime improvements.

I. Introduction

Boolean symbolic reasoning, which extracts word-level abstractions from gate-level netlists, plays a critical role in electronic design automation (EDA) such as logic synthesis, datapath optimization, and formal verification [8], [20], [21]. Additionally, the globalization of VLSI design and manufacturing processes has amplified the need for detecting malicious logic, such as hardware Trojans, to ensure hardware security [22], [26]. However, conventional approaches, such as those based on cut enumeration [3], [25], [31], structural hashing [20], [21], [42], and machine learning (ML) techniques [1], [12], [35], [37], face several critical limitations when addressing complex reasoning tasks, particularly in *bit-blasted* non-linear arithmetic netlists:

(1) *Static structural limitation* – Structural approaches, such as those based on cut enumeration and local structural hashing [3], [25], [31], often use circuit topology for shape hashing to identify structurally similar wires and form word-level abstractions. Alternatively, they rely on reference libraries to map sub-circuits by matching local truth tables [5], [42]. In these cases, the original high-level structures of a given netlist—such as carry-chains and adder trees—are often fragmented or altered during heavy logic optimization and technology mapping, rendering them unrecognizable. **(2)** *Lack of completeness and exactness*: State-of-the-art (SOTA) Boolean methods for symbolic reasoning focus on detecting Negation-Permutation-Negation (NPN) classes of functional components, which represent groups of structurally similar but not exact functional blocks. While these abstractions are useful for technology mapping [14], they fail to provide exactness guarantees, which are critical for real-world applications like formal verification. Furthermore, ML-based approaches, such as Gamora [37], HOGA [12], and DeepGate [17], [29], lack completeness and correctness guarantees due to the inherently probabilistic nature of ML models. **(3)** *Scalability challenges of functional approaches*: Functional approaches using symbolic evaluation are solver-ready (e.g., SAT-based approaches [2], [11]), but incur significant computational overhead, especially when applied to *bit-blasted* non-linear arithmetic Boolean networks [15], [20], [40].

Given the limitations of conventional approaches, we present **BoolE**, an exact symbolic reasoning framework that utilizes Boolean *equality saturation*. BoolE is designed to enhance reasoning performance for complex Boolean netlists, which are technology-mapped or logic-optimized. It takes Boolean netlists as input and infers functional blocks through Boolean-level equality saturation and term rewriting. By leveraging the e-graph saturation theory [32], BoolE offers scalable solutions to reasoning high-level blocks in heavily optimized and technology mapped arithmetic multipliers. More importantly, BoolE offers formal exactness and correctness guarantees. The key contributions of BoolE are summarized as follows:

1) **Boolean reasoning through equality saturation:** To the best of our knowledge, BoolE is the first end-to-end framework that uses equality saturation to explore a vast space of equivalent Boolean expressions. BoolE methodologies are believed to have broader impacts beyond symbolic reasoning in Boolean domain.
2) **Novel exact extraction algorithm for multi-output structures:** BoolE introduces an innovative exact extraction algorithm alongside a domain-specific Boolean ruleset, capable of handling complex multi-input, multi-output high-level structures. This ensures precise and efficient logic reconstruction, with exactness and correctness guaranteed by equality saturation theory.
3) **Comprehensive evaluation on reasoning:** Extensive experimental results demonstrate that BoolE outperforms SOTA functional, structural, and ML-based approaches by significant margins in both exact and NPN reasoning.
4) **End-to-end integration to real-world application:** BoolE has been seamlessly integrated with SOTA synthesis and verification tools, such as ABC, as well as formal verification backends like RevSCA2.0 [21]. A case study on the formal verification of arithmetic multipliers demonstrates over **four orders of magnitude** runtime acceleration on RevSCA-2.0.

II. Background

A. Boolean Networks

Boolean networks are mathematical models that represent logical relationships between binary variables, where nodes signify Boolean

*: co-first authors. J. Yin and C. Yu contributed the idea. J. Yin implemented the framework. Z. Song conducted experimental and baseline benchmarking setups. All authors contributed to the writing.

variables (0 or 1) and edges denote dependencies defined by Boolean functions. These networks are fundamental in digital circuit design for modeling combinational logic, and be used in computational biology for gene regulatory networks. An And-Inverter Graph (AIG) is a specialized Boolean network extensively used in electronic design automation (EDA). It is a directed acyclic graph (DAG) composed solely of two-input AND gates and NOT gates (inverters). AIGs offer a compact representation of Boolean functions, facilitating efficient manipulation, optimization, and verification of complex digital circuits. All combinatorial Boolean networks can be converted to AIG with De Morgan's laws.

A cut of a Boolean network is a pair (r, S), where r is a node called the root, and S is a set of nodes called the leaves, such that: (1) Every path from a primary input to r passes through at least one leaf in S. (2) For each leaf $l \in S$, there is a path from a primary input to r that goes through l but no other leaf. The size of the cut is the number of leaves $|S|$. A cut is K-feasible if its size $|S| \leq K$. This work focuses on 3-feasible cuts. The cut enumeration algorithm is a fundamental technique in Boolean network analysis, specifically used in symbolic reasoning and technology mapping. By enumerating all possible K-feasible cuts, the cut enumeration algorithm systematically explores various sub-network configurations, enabling the detection of critical Boolean functions such as FAs and multiplexers. The exhaustive exploration enables the identification of optimal sub-functions, enhances logic optimization, and supports technology mapping by providing a comprehensive understanding of the circuit functional structure. Despite the effectiveness of the cut enumeration algorithm, it struggles to accommodate structural modifications such as technology mapping and logic optimization.

NPN classification is a method used in logic synthesis and technology mapping to group Boolean functions into equivalence classes [14]. Two Boolean functions are considered NPN equivalent if one can be transformed into the other through a combination of input negations, input permutations, and output negation. Each NPN class of a representative function comprises all functions that can be derived from these transformations.

B. Boolean Word-Level Block Identification

Boolean word-level block identification in digital design aims to detect elementary functional blocks (e.g., FA) within netlists, providing critical word-level abstractions that enable logical optimization, formal verification, malicious logic detection, and other verification tasks [7], [21], [22]. Conventional approaches include structural shape hashing and functional bitslice aggregation [18], [31], [42], which analyze block-level netlists from structural and functional perspectives. Structural shape hashing assigns a unique shape to each edge in a Boolean netlist to capture its structural properties. This shape is defined as a directed graph that represents the backward-reachable logic gates from the wire. Functional bitslice aggregation, on the other hand, relies on functional matching to group equivalent nodes and edges using the cut enumeration algorithm. The cut enumeration algorithm is integrated into synthesis tools like ABC. Besides, some works also focus on recover higher-level programming abstractions in hardware description language (HDL) code [27], [30]. Recently, Graph Neural Networks (GNNs) have emerged as a promising tool for Boolean block identification [1], [12], [13], [35], [37]. Gamora [37], for instance, leverages GNNs and GPU acceleration to efficiently infer high-level functional blocks from gate-level netlists.

FA is the fundamental functional block for adder trees, consisting of sum and carry signals. which are represented by XOR and MAJ gates with identical input signals. Both the cut enumeration algorithm in ABC and the Gamora framework identify FAs based on NPN equivalence classes. In this paper, we refer to the logically equivalent FA as **exact FA**, and NPN equivalent FA as **NPN FA**.

C. Equality Saturation

Equality saturation [23], [24], [32], [43] is a rewriting optimization technique powered by an e-graph (equivalence graph) data structure, which represents an equivalence relation over expressions. An **e-graph** $\mathcal{G}$ is formally defined as a tuple $\mathcal{G} = (V, \mathcal{R}, \lambda)$, where:

- V is a finite set of vertices, referred to as **E-nodes**. Each e-node within V represents a distinct expression or sub-expression.
- $\mathcal{R} \subseteq V \times V$ defines an equivalence relation on V, partitioning it into equivalence classes known as **E-classes**. These E-classes are the equivalence classes generated by $\mathcal{R}$. Nodes within the same e-class are considered equivalent under the relation $\mathcal{R}$.
- $\lambda : V \to \Sigma \times V^k$ is a labeling function that assigns each node to an operator and an ordered list of child nodes, where Σ represents the set of operators with k operands and V^k denotes an ordered tuple of k child nodes from V.

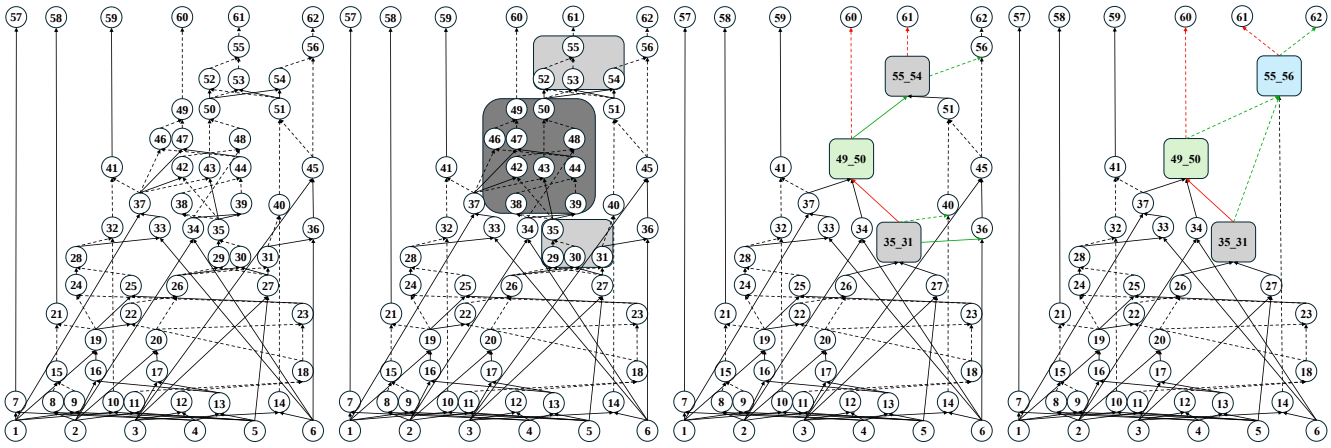
E-graphs form the foundation of **equality saturation** optimization technique, which applies rewrite rules iteratively until no new equivalences can be introduced. The process involves: (1) constructing an e-graph $\mathcal{G}_0$ from the initial expression s_0, (2) applying rewrite rules (l_i, r_i) to incorporate equivalences $[l_i]_\mathcal{R} \sim [r_i]_\mathcal{R}$, and (3) repeating until convergence, which indicates that no further changes can be made. E-graphs compactly store an exponential number of expressions in linear space by sharing sub-expressions, preserving all expressions and avoiding the phase-ordering problem [16]. Equality saturation has been applied in numerous areas within hardware design automation [4], [6], [9], [33], [34], [39]. For our implementation, we utilize the **egg** tool [36], an advanced e-graph framework, as the backend for BoolE e-graphs.

III. Motivating Example

In this section, we present a motivating example in Figure 1 to illustrate challenges in FA block identification. Figure 1(a) shows the AIG of a 3-bit carry-save-array (CSA) multiplier after ASAP 7nm technology mapping [38], with solid and dashed lines representing output signals and their negations, respectively. Prior to technology mapping, the 3-bit CSA multiplier netlist contains 3 FAs. However, after technology mapping, ABC identifies only one NPN FA block through cut enumerations, as shown in Figure 1(b). In this figure, the output of XOR and MAJ gates are highlighted in red and green, respectively. Figure 1(c) illustrates the adder tree of the 3-bit post-mapping CSA multiplier, where block `49_50` is identified as an NPN FA (marked in green), and blocks `55_54` and `35_31` are HAs (marked in gray).

While NPN classification effectively recognizes structurally similar components, it does not guarantee logical equivalence to true FAs. For tasks such as formal verification, which require precise logical equivalence to ensure the correctness of a design, identifying NPN-equivalent FAs is insufficient. Formal verification demands that the functional behavior of the circuit matches the specification exactly, without deviations introduced in NPN transformations. For example, RevSCA-2.0 [21] utilizes block identification of exact HAs and FAs to eliminate vanishing monomials by detecting and locally removing sources of redundant terms ahead of backward rewriting. However, this process requires exact logical equivalence to functional blocks.

Conventional Boolean reasoning methods, like cut enumeration, rely on static netlist structures to identify functional blocks such as FAs. However, processes like technology mapping and logic optimization often modify or eliminate these structures, rendering them



(a) AIG of 3-bit CSA multiplier w 7nm ASAP mapping.

(b) FA/HA block in the AIG.

(c) NPN FA block 49_50 and HA blocks {35_31,55_54} by cut enumeration.

(d) BoolE identifies an additional exact FA block 55_56 is extracted.

Fig. 1: Motivations for BoolE. The component blocks of exact FA, NPN FA, and HA are shown in blue, green, and grey blocks. The output signals of corresponding XOR and MAJ are marked as green and red correspondingly. (a) AIG of a 3-bit CSA multiplier generated by ABC and mapped using 7nm ASAP Technology Mapping. (b) FA and HA blocks in AIG detected by ABC. (c) Adder tree extracted from AIG. There is only one NPN FA. (d) The output netlist extracted after BoolE rewriting. An additional exact FA is reconstructed after BoolE rewriting.

unrecognizable to traditional tools. For the example in Figure 1, only one FA can be identified with cut enumeration. In contrast, equality saturation rewrites netlists into functionally equivalent forms, enabling exhaustive exploration of alternative structures. This flexibility opens opportunities for reconstructing functional blocks, thereby improving the robustness and effectiveness of symbolic reasoning tasks.

Finally, we present the netlist extracted from BoolE in Figure 1(d), where BoolE has reconstructed the netlist structure for FA identification. Apart from the NPN FA `49_50`, BoolE successfully identifies an additional exact FA, `55_56` (marked in blue), after term rewriting. For a large-scale 128-bit technology-mapped CSA multiplier, BoolE reconstructs 14,713 out of 16,128 NPN FAs within the theoretical upper bound, which is 3,771 more than what ABC can identify. Furthermore, 3,418 of the reconstructed FAs are exact FAs, representing a $4.42\times$ performance improvement compared to ABC.

IV. Approach

The overview of BoolE is illustrated in Figure 2. The framework accepts gate-level netlists as input, where each node represents a standard Boolean algebra operation (e.g., `AND`, `XOR`, `NOT`). Initially, BoolE parses the input AIG file, extracts necessary information, and converts the netlist into an e-graph. Subsequently, the BoolE rewriting engine applies rewriting rules to expand the e-graph, thereby identifying `MAJ` and `XOR` components. For e-graph extraction, we developed a DAG-based cost extraction algorithm to ensure the multi-input multi-output structure (i.e., FA) is correctly captured. Finally, BoolE converts the extracted e-graph into AIG format. And the output AIG can also be integrated into other tools (e.g., PolyCleaner and RevSCA [19], [20]) for further applications such as functional verification.

A. BoolE Methodology - Symbolic Reasoning via Equality Saturation

To identify FAs and reconstruct adder tree structures, BoolE employs equality saturation to process the input netlists. Here, we introduce the core components of the framework: e-graph construction and rewriting ruleset.

1) E-Graph Construction: BoolE gathers functional information from input netlists to accurately represent the AIG structure and construct the e-graph based on this information. The algorithm for e-graph construction is detailed in Algorithm 1. Firstly, BoolE collects data on the primary inputs and primary outputs of the netlists. Furthermore, BoolE establishes the dependency relationships for each Boolean operation, defining the input and output edges for the nodes. Subsequently, BoolE incrementally inserts operation nodes into the e-graph following the topological order of the graph nodes. Specifically, operations are inserted into the e-graph from the leaf nodes to the root nodes to ensure that child nodes are processed beforehand. This topological insertion order is critical for accurately maintaining the dependencies between operations. Each insertion generates an identifier representing the e-nodes and connects these e-nodes with their respective input parameters. The e-graph construction is shown in part ① in Figure 2.

Algorithm 1 E-graph Construction

Input: Vertex list V from parsing netlist
Output: : e-graph G_e
1: Initialize $vmap$ as empty HashMap
2: **for** each node v in TopoSort(V) **do** ▷ From leaf to root
3: $in_id \leftarrow []$
4: **for** each input i for node v **do**
5: in_id.pushback($vmap[i]$) ▷ All children are inserted
6: **end for**
7: $id \leftarrow G_e$.insert(v, in_id) ▷ Insert v to e-graph
8: $vmap[v] = id$
9: **end for**

2) E-Graph Rewriting: The rulesets are composed of two parts: (1) basic Boolean rules (R_1), such as the commutative law, associative law, and De Morgan's Laws, which aim to expand the e-graph by saturating it with more functionally equivalent expression trees; (2) rules for `XOR` and `MAJ` identification (R_2), which directly identify `XOR` and `MAJ` operations within the e-graph. A subset of these rewriting rules is presented in Table I. In total, we collected 68 rules in R_1 and 119 rewriting rules for $R2$. To construct R_2, we utilized 8- to 128-bit Booth and CSA multipliers as templates and employed the synthesis tool ABC to detect the adder trees within the netlists and extract the structural patterns of `MAJ` and `XOR` operations. We constructed the corresponding e-graph rewriting rules R_2 and eliminated duplicate rules from the rule set.

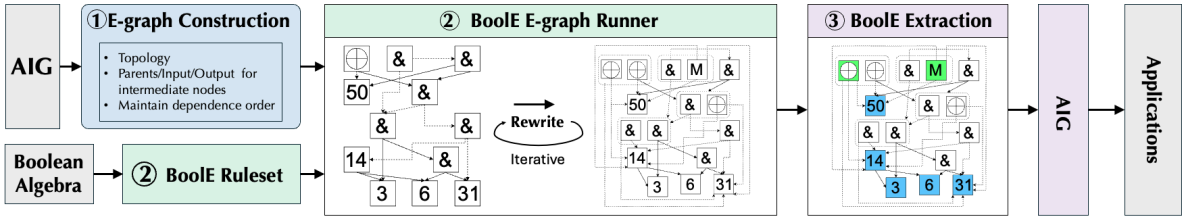


Fig. 2: The overview of BoolE, an exact symbolic reasoning framework for Boolean netlists utilizing equality saturation. It enhances performance by employing a novel extraction algorithm specifically designed to identify and capture multi-input, multi-output exact FA structure. We incorporate part of the e-graph for motivating example shown in Figure 1 to illustrate how the additional extra FA is extracted from e-graph, where the extracted e-nodes are specially highlighted. The XOR and MAJ gates forming an exact FA are highlighted in green.

The ruleset construction and e-graph rewriting correspond to part ② in Figure 2. Here we introduce 3 optimization tricks we employ in e-graph rewriting:

1) To enhance scalability while keeping performance, BoolE offers a lightweight version of rewriting rules. This version is carefully designed through empirical, manual pruning to ensure effectiveness for large-scale benchmarks, striking a balance between reducing complexity and maintaining reasoning performance.
2) Additionally, the two parts of the rewriting rules are applied incrementally in two phases. Firstly, we first apply R_1 to the initial e-graph with 10 iterations of rewriting. Subsequently, based on the R_1 rewrited e-graph, we apply R_2 with 3 iterations. This incremental saturation approach allows fine adjustment of the iteration limits for R_1 and R_2 separately.
3) After e-graph saturation, BoolE deletes redundant e-nodes that do not contribute to performance improvements. This reduces unnecessary memory usage and runtime. For instance, based on the commutative property, expressions such as $\texttt{XOR}(a,b,c)$, $\texttt{XOR}(b,a,c)$ belong to the same e-class. BoolE retains only one unique expression within each e-class, deleting the other equivalent e-nodes. A similar approach is applied to $\texttt{MAJ}(a,b,c)$ and $\text{FA}(a,b,c)$ expressions.

TABLE I: Example BoolE rewriting rules. The complete rewriting library consists of 68 Boolean rewriting rules and 39/90 additional rules specifically designed for the identification of MAJ/XOR operations.

	Pattern	Transformation
Basic rules	$a \,\&\, b$	$b \,\&\, a$
	a	$(\,a\,')\,'$
	$(a \,\&\, b)'$	$a' \,||\, b'$
	$(a \,||\, b)'$	$a' \,\&\, b'$
	$(a \,\&\, b) \,\&\, c$	$a \,\&\, (b \,\&\, c)$
MAJ rules	$(a \,\&\, b) \,||\, (a \,\&\, c) \,||\, (b \,\&\, c)$	$\texttt{MAJ}(a,b,c)$
	$(a' \,\&\, (b \,\&\, c)')' \,\&\, (b' \,\&\, c')'$	$\texttt{MAJ}(a,b,c)$
XOR rules	$(a\&b'\&c') \,||(a'\&b\&c') ||(a'\&b'\&c) ||(a\&b\&c)$	$\texttt{XOR}(a,b,c)$
	$(((a\|\|(b\&c)) \,||(b||c)') \,\&\, (a'|| \,((b\&c)'\&(b||c))))'$	$\texttt{XOR}(a,b,c)$

B. BoolE E-Graph Exact Extraction

In this section, we introduce our e-graph extraction for multi-input multi-output FA structure, which corresponds to part ③ in Figure 2.

Cost Function: Our cost function aims to maximize the number of exact FAs in the extracted expression within the e-graph G. Specifically, for an expression tree t_e composed of e-nodes e, the cost function $C(t_e)$ is defined as: $C(t_e) = \sum_{e \in t_e} -1_{\{\text{e is exact FA}\}}$. E-graph extraction seeks the optimal expression tree t^* with the lowest total cost from $\mathcal{T}$, the set of all possible expression trees in G:

$$t^* = \arg\min_{t_e \in \mathcal{T}} C(t_e)$$

Each expression tree t is constructed by selecting one node from each e-class, ensuring conformity to the e-graph structure.

Multi-output FA structure: Standard e-graph structures typically employ prefix notation expressions, assuming that all operators have single outputs. Consequently, the e-graph framework inherently lacks support for multi-output operations, only focusing on single-output functionalities. However, FA is a multi-input, multi-output structure, consisting of three inputs and two outputs representing the carry and sum, respectively.

To accommodate multi-output operations, equality saturation introduces new e-nodes into the e-graph. BoolE iterates through all e-nodes, identifying XOR and MAJ nodes with the exact same inputs. When such nodes are found, BoolE inserts an FA node that shares the same inputs and generates a tuple of outputs for carry and sum. Subsequently, BoolE employs the FST and SND pseudo-operations to project the individual carry and sum signals from the output tuple. Finally, BoolE unifies the e-classes of the FST and SND operations with the corresponding XOR and MAJ nodes, ensuring their equivalence within the e-graph framework. The structure of the FA is depicted in Figure 3. It is crucial to note that extracting FA structures is treated as an atomic operation. Thus, FST, SND, and FA must be selected collectively or not at all. This atomic extraction ensures the integrity of the FA structure is maintained during BoolE extraction.

DAG based extraction algorithm: BoolE utilizes DAG cost extraction, which counts each shared sub-expression only once for greater accuracy, to prevent double-counting of FA numbers. The extraction process is detailed in Algorithm 2. The algorithm first initializes a queue with all leaf nodes and an empty cost map for each e-class. It then iteratively processes nodes by dequeuing them and checking if all child nodes have computed costs. If so, it calculates the new cost for the node. If the new cost is lower than the previous cost, the cost map is updated, and parent nodes are enqueued. Nodes labeled FST/SND, and FA are handled specially in a post-processing step to ensure the integrity of FA structures. Each e-class maintains a cost map with the current best cost, selected e-node, and chosen child e-classes. By iteratively updating costs, the algorithm efficiently constructs the optimal expression tree t^*. This hash map, mapping child e-class IDs to their corresponding costs, introduces memory usage issues as the input e-graph grows. To save memory, BoolE uses flexible data types for the keys and values in the cost hash map. For small e-graphs (fewer than 65,536 e-classes), BoolE uses unsigned 16-bit integers for hash map keys, conserving memory. For larger e-graphs, it adjusts to unsigned 32-bit integers. This adaptability based on e-graph size ensures efficient memory utilization.

V. EXPERIMENTAL RESULTS

The experiments were conducted on a system with an Intel(R) Xeon(R) Gold 6418H CPU, featuring 48 physical cores and 2 threads per core, and 1 TByte of RAM. We use the *egg* framework [36]

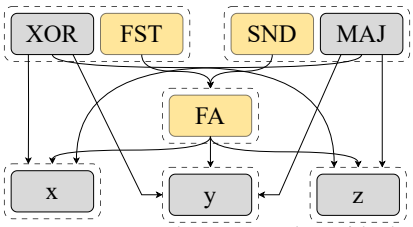


Fig. 3: FA Structure: XOR and MAJ e-nodes with the exactly same inputs will be paired. Each paired XOR/MAJ operation will insert a FA nodes and utilize *FST/SND* to project out the carry/sum bits. *FST/SND* will unified to the e-classes of XOR/MAJ

Algorithm 2 E-Graph Extraction

Input: e-graph G_e **Output:** Optimal t^* w.r.t. $Cost(G_e)$

```
function CalculateCostSet(G_e, node, Costs)
    results ← {key:Costs[key].results} for key in node.children()
    total ← Sum(result.values())
    return {results,total,node}
end function
function Extract(G_e, root)
    Queue ← LeafNode(G_e)                    ▷ Queue of e-nodes to be processed
    t* = ∅                                   ▷ Expression tree to be constructed
    Costset = ∅                              ▷ Initial empty map of e-class to cost
    while Queue is not empty do
        node = Queue.pop()
        if all c in node.children() are in Costset then
            prev_cost = Costset[class_id].total
            new_cost = CalculateCostSet(G_e, node, Costset)
            if new_cost.total < prev_cost then
                Queue.insert(node.parents())          ▷ Cost map update
                Costset[node]= new_cost
                if node is FST/SND then
                    PostProcessing()                  ▷ FA structure integrity check
                end if
            end if
        end if
    end while
end function
```

as the back-end to develop **BoolE** for e-graph construction and term rewriting. The experimental benchmarks include unsigned CSA multipliers and signed Booth multipliers, collected from the state-of-the-art (SOTA) baseline works ABC [42] and Gamora [37]. Specifically, the AIG-based approach (`&atree`) in ABC leverages structural hashing and functional pattern matching that reconstructs the adder tree structure via the cut enumeration algorithm [42]. Gamora is a SOTA graph learning-based Boolean reasoning tool [37] that targets the same objective, while the training dataset is collected with the AIG-based approach [42].

A. Performance of Symbolic Reasoning

To demonstrate the performance of BoolE, we prepared three research questions(RQs) in this section:

RQ1: How effective of BoolE without structure rewriting? – We collected the number of FAs identified by ABC and BoolE across various bitwidths for pre-mapping netlists. All adder tree structures are present in the netlists before technology mapping or logic optimization. For instance, in an n-bit CSA multiplier, the theoretical upper bound of FA number is $(n-1)^2-1$, which can be determined through exhaustive cut enumeration.

ABC can identify all adder trees in pre-mapping netlists successfully. Figure 4 presents the results alongside the theoretical upper bound for the number of pre-mapping FAs. In this figure, the upper bound curve data also represents the number of pre-mapping FAs identified by ABC and BoolE. **And BoolE also achieves the optimal solution for both CSA and Booth benchmarks across all bitwidths for pre-mapping benchmarks.** Since all NPN FAs can be exhaustively identified in pre-mapping netlists without rewriting, the reasoning performance is entirely dominated by ruleset R_2 described in Section IV-A2, which is specialized for XOR and MAJ identification. The result underscores the superior performance of ruleset R_2, which dominates the reasoning capability to discover all existing NPN FAs in pre-mapping netlists.

RQ2: How effective is BoolE under heavily logic optimization and technology mapping? – Reasoning high-level components under heavily logic optimization and technology mapping settings is known to be significant more challenging [37] [42] [12] [18], which is more critical in real-world datapath synthesis [10], [41] and formal verification scenarios [21], [40], [42]. Therefore, our objective is to demonstrate the robustness of BoolE under technology mapping and logic optimization, which reflects the rewriting and extraction capability of BoolE.

To evaluate the performance of BoolE on technology-mapped netlists, we quantified the number of FAs discovered by BoolE across technology-mapped CSA and Booth multipliers of various bitwidths. We utilized the synthesis tool ABC to perform technology mapping with the 7nm ASAP library, which encompasses a diverse set of 161 standard-cell gates. The test results are presented in Figure 4, with the CSA and Booth multiplier results shown in the left and right subfigures, respectively. The x-axis represents the bitwidth of the benchmark multipliers, while the y-axis indicates the number of FAs discovered using different reasoning tools: BoolE, ABC, and Gamora.

The results demonstrate that BoolE successfully reconstructs the majority of FA structures, and provides superior exact symbolic reasoning performance for multipliers compared to ABC and Gamora. Later in Section V-B, we also demonstrate its robustness against formal verification tool RevSCA-2.0, which integrates the same reasoning function. **Specifically, BoolE achieves an average of** 93.48% **and** 84.81% **of the theoretical upper bound for NPN FAs in technology-mapped CSA and Booth multipliers, respectively.** In comparison, ABC can only identify 68.07% and 69.28% of the maximum NPN FAs for CSA and Booth multipliers on average, respectively. Furthermore, Gamora reasoning performance drops to 63.81% and 66.68% for CSA and Booth multipliers, respectively. BoolE constantly outperforms ABC and Gamora for post-mapping netlists. Additionally, BoolE focuses on reconstructing as many exact FAs as possible. **The number of exact FAs after BoolE rewriting increased by** $3.53\times$ **and** $3.01\times$ **compared with ABC for CSA and Booth multipliers, respectively.**

RQ3: Is BoolE scalable? – To demonstrate the scalability of BoolE, we present the end-to-end runtime across all tested benchmarks. Specifically, we recorded the runtime and number of nodes in the netlist of each benchmark, including post-mapping CSA and Booth multipliers. The results are illustrated in Figure 5.

Although functional rewriting for Boolean networks involves higher time complexity compared to simple structural matching in cut enumeration, BoolE achieves efficient rewriting runtimes. It completes all benchmarks for post-mapping Booth and CSA multipliers from 4-bit to 128-bit within 50 minutes. For the largest case, 128-bit post-mapping CSA multipliers containing 191,863 AIG nodes, the rewriting process takes 50 minutes. The runtime for BoolE rewriting is insignificant compared to the order-of-magnitude improvements it brings to real-world applications. In tasks such as formal verification, the reasoning performance directly impacts the overall complexity and runtime. BoolE's superior reasoning performance enables orders of magnitude speedup in the formal verification of larger, optimized multipliers. For example, BoolE takes only 11 seconds to reason the netlist for a 24-bit multiplier, while the verification runtime is drastically reduced from 32,402.74 seconds to just 0.07 seconds. The runtime spent on the

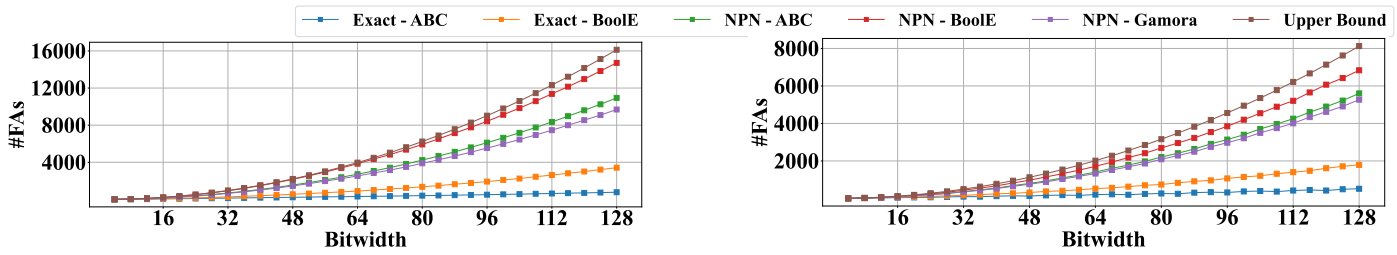


Fig. 4: Performance comparison among BoolE, ABC, and Gamora for CSA (left) and Booth (right) multipliers after ASAP 7nm technology mapping. BoolE consistently outperforms ABC and Gamora, identifying 3.53× and 3.01× exact FAs than ABC in CSA and Booth multipliers. And BoolE achieves an average of 93.48% and 84.81% of the theoretical upper bound in number of NPN FAs, respectively.

rewriting process is well compensated by the substantial reductions achieved in end-to-end verification time. The runtime and efficiency for formal verification tasks will be elaborated in Section V-B.

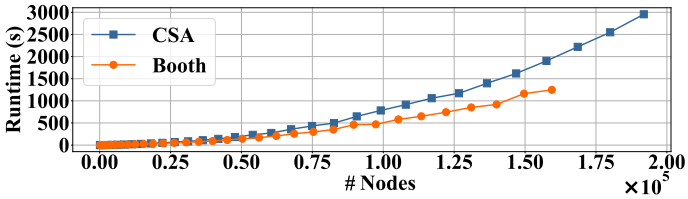


Fig. 5: BoolE runtime of post-mapping CSA and Booth multiplier w.r.t. the size of the input netlist (AIG node number).

B. Integrated Application to Formal Verification

RQ4: Can BoolE practically enhance real-world applications orthogonally (formal verification)? Boolean symbolic reasoning is a critical topic in various domains, such as functional verification and datapath optimization. We take verification of multipliers as a case study to illustrate how BoolE can enhance real-world applications. Boolean symbolic reasoning, as an orthogonal technique, can be integrated into many formal verification tools for reverse engineering [20], [21], [28]. Here we utilize RevSCA-2.0 [21] as our back-end tool. The complexity of symbolic computer algebra-based multiplier verification is determined by the maximum polynomial size (# monomials) [40], and its performance is measured by verification runtime. To mitigate polynomial explosion, RevSCA-2.0 uses cut enumeration to detect functional blocks, including exact FAs. It then eliminates vanishing monomials within these blocks before backward rewriting. Consequently, reasoning performance directly impacts the detection of functional blocks, the elimination of vanishing monomials, and the overall success of verification.

ABC offers a logic circuit optimization feature that simplifies Boolean expressions and reduces gate counts. This optimization functionality is integrated into the *dch* interface. We evaluated the total runtime of verifying a CSA multiplier benchmark, optimized using *dch* bit-optimization, under two configurations: with and without BoolE optimization. Note that RevSCA-2.0 is equipped with a functional reasoning engine with cut enumeration. The test results are presented in Table II. This table provides the verification runtime across various bitwidths. Additionally, we report the maximum polynomial size (Max Poly Size) observed during the backward iteration process, which reflects the complexity of the verification tasks. It is important to note that we only report the number of exact FAs because NPN FAs cannot be detected as functional blocks, and only exact FAs are beneficial for multiplier verification simplification. Verification runtime exceeding 72 hours is marked as time out (TO).

We can see a significant improvement in runtime brought by BoolE rewriting. The formal verification of CSA multipliers after logic optimization gets timed-out without BoolE rewriting when the bitwidth is larger than 24 bits. After BoolE rewriting, the multipliers of up to 128 bit can be formally verified. And there is a significant speedup in verifying multipliers with higher bitwidths. For example, BoolE provides 2825× runtime improvement for 24-bit *dch*-optimized multiplier via exact symbolic reasoning. Furthermore, BoolE enables the verification of *dch*-optimized multipliers with bitwidths exceeding 24 bits. However, without BoolE, RevSCA-2.0 cannot formally verify these larger multipliers within a **72-hour timeout**.

TABLE II: Verification results of `dch`-optimized CSA multipliers using RevSCA-2.0 under two configurations: with BoolE optimization (BoolE) and without optimization (Baseline).

Bitwidth	Number of Exact FAs			Max Poly Size		End-to End Runtime (s)	
	Upper Bound	BoolE	Baseline	BoolE	Baseline	BoolE	Baseline
8	48	41	0	85	373	0.445	0.035
12	120	109	0	173	2173	0.955	0.297
16	224	209	0	293	34345	2.474	17.332
20	360	341	0	445	548917	5.165	1471.574
24	528	505	0	629	8781889	11.470	32402.739
28	728	701	0	845	-	22.951	TO
32	960	929	0	1093	-	43.910	TO
64	3968	3905	0	4229	-	1232.219	TO
96	9024	8929	0	9413	-	8748.443	TO
128	16128	16001	0	16645	-	33255.292	TO

As illustrated in the fourth column of Table II, the number of exact FAs detected by ABC in CSA multipliers drops to zero due to logic optimization. This disappearance of exact FAs significantly increases the complexity of the verification process during backward rewriting, especially as the bitwidth scales. On the other hand, BoolE can reconstruct most disappeared functional blocks. In the third column of Table II, we present the number of exact FAs reconstructed by BoolE, which is compared with the upper bound in the second column which represents the maximum number of FAs in CSA multipliers. BoolE successfully reconstructs up to 99.2% of the exact FAs within the theoretical upper bound.

VI. Conclusion

In conclusion, this paper introduces BoolE, an exact Boolean symbolic reasoning framework that leverages equality saturation techniques in e-graphs to overcome the limitations of conventional methods. By assembling a comprehensive ruleset and implementing careful optimizations, BoolE achieves enhanced performance in Boolean symbolic reasoning, enabling accurate Boolean function detection. BoolE effectively reconstructs multi-output structures, such as FAs, within technology-mapped and optimized netlists. Our evaluations demonstrate BoolE's significant improvements, highlighting its potential to advance digital circuit synthesis and verification.

Acknowledgment – This work is supported by National Science Foundation (NSF) under CCF2350186, CCF2403134, CCF2349670, and CCF2349461 awards.

REFERENCES

[1] L. Alrahis, A. Sengupta, J. Knechtel, S. Patnaik, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2435–2448, 2021.

[2] A. Biere, "Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013," *SAT Competition*, 2013.

[3] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*. Springer, 2010, pp. 24–40.

[4] Y. Cai, K. Yang, C. Deng, C. Yu, and Z. Zhang, "Smoothe: Differentiable e-graph extraction," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025, pp. 1020–1034.

[5] B. Cakir and S. Malik, "Reverse engineering digital ics through geometric embedding of circuit graphs," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 4, pp. 1–19, 2018.

[6] C. Chen, G. Hu, D. Zuo, C. Yu, Y. Ma, and H. Zhang, "E-syn: E-graph rewriting with technology-aware cost functions for logic synthesis," in *ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[7] M. Ciesielski, T. Su, A. Yasin, and C. Yu, "Understanding algebraic rewriting for arithmetic circuit verification: a bit-flow model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 6, pp. 1346–1357, 2019.

[8] M. Ciesielski, C. Yu, W. Brown, D. Liu, and A. Rossi, "Verification of gate-level arithmetic circuits by function extraction," in *Proceedings of the 52nd Annual Design Automation Conference*, 2015, pp. 1–6.

[9] S. Coward, G. A. Constantinides, and T. Drane, "Automatic datapath optimization using e-graphs," in *2022 IEEE ARITH*. IEEE, 2022, pp. 43–50.

[10] S. Coward, T. Drane, E. Morini, and G. A. Constantinides, "Combining power and arithmetic optimization via datapath rewriting," in *2024 IEEE 31st Symposium on Computer Arithmetic (ARITH)*. IEEE, 2024, pp. 24–31.

[11] L. De Moura and N. Bjørner, "Satisfiability Modulo Theories: Introduction and Applications," *Communications of the ACM*, 2011.

[12] C. Deng, Z. Yue, C. Yu, G. Sarar, R. Carey, R. Jain, and Z. Zhang, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[13] Z. He, Z. Wang, C. Bai, H. Yang, and B. Yu, "Graph learning-based arithmetic block identification," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[14] Z. Huang, L. Wang, Y. Nasikovskiy, and A. Mishchenko, "Fast boolean matching based on npn classification," in *2013 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2013, pp. 310–313.

[15] D. Kaufmann and A. Biere, "Amulet 2.0 for verifying multiplier circuits," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2021, pp. 357–364.

[16] B. W. Leverett, R. G. G. Cattell, S. O. Hobbs, J. M. Newcomer, and et. al, "An overview of the production quality compiler-compiler project," *Computer*, vol. 13, no. 8, pp. 38–49, 1980.

[17] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 667–672.

[18] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia, "Wordrev: Finding word-level structures in a sea of bit-level gates," in *HOST*. IEEE, 2013, pp. 67–74.

[19] A. Mahzoon, D. Große, and R. Drechsler, "Polycleaner: clean your polynomials before backward rewriting to verify million-gate multipliers," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[20] ——, "Revsca: Using reverse engineering to bring light into backward rewriting for big and dirty multipliers," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[21] ——, "Revsca-2.0: Sca-based formal verification of nontrivial multipliers using reverse engineering and local vanishing removal," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 5, pp. 1573–1586, 2021.

[22] T. Meade, S. Zhang, Y. Jin, Z. Zhao, and D. Pan, "Gate-level netlist reverse engineering tool set for functionality recovery and malicious logic detection," in *International Symposium for Testing and Failure Analysis*, vol. 81368. ASM International, 2016, pp. 342–346.

[23] G. Nelson and D. C. Oppen, "Fast decision procedures based on congruence closure," *Journal of the ACM (JACM)*, vol. 27, no. 2, pp. 356–364, 1980.

[24] R. Nieuwenhuis and A. Oliveras, "Proof-producing congruence closure," in *International Conference on Rewriting Techniques and Applications*. Springer, 2005, pp. 453–468.

[25] P. Pan and C.-C. Lin, "A new retiming-based technology mapping algorithm for lut-based fpgas," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, 1998, pp. 35–42.

[26] S. Rajendran and M. L. Regeena, "A novel algorithm for hardware trojan detection through reverse engineering," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 1154–1166, 2021.

[27] V. Rao and Z. D. Sisco, "Register aggregation for hardware decompilation," *arXiv preprint arXiv:2409.03119*, 2024.

[28] D. Ritirc, A. Biere, and M. Kauers, "Improving and extending the algebraic approach for verifying gate-level multipliers," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2018, pp. 1556–1561.

[29] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: towards scalable circuit representation learning," *arXiv preprint arXiv:2407.11095*, 2024.

[30] Z. D. Sisco, J. Balkind, T. Sherwood, and B. Hardekopf, "Loop rerolling for hardware decompilation," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 420–442, 2023.

[31] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, "Reverse engineering digital circuits using functional analysis," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2013, pp. 1277–1280.

[32] R. Tate, M. Stepp, Z. Tatlock, and S. Lerner, "Equality saturation: a new approach to optimization," in *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, 2009, pp. 264–276.

[33] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, "Impress: Large integer multiplication expression rewriting for fpga hls," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2022, pp. 1–10.

[34] E. Ustun, C. Yu, and Z. Zhang, "Equality saturation for datapath synthesis: A pathway to pareto optimality," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–2.

[35] Z. Wang, Z. He, C. Bai, H. Yang, and B. Yu, "Efficient arithmetic block identification with graph learning and network-flow," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 8, pp. 2591–2603, 2022.

[36] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, and P. Panchekha, "Egg: Fast and extensible equality saturation," *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

[37] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie, "Gamora: Graph learning based symbolic reasoning for large-scale boolean networks," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[38] X. Xu, N. Shah, A. Evans, S. Sinha, B. Cline, and G. Yeric, "Standard cell library design and optimization methodology for asap7 pdk," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 999–1004.

[39] Y. Yang, P. Phothilimthana, Y. Wang, M. Willsey, S. Roy, and J. Pienaar, "Equality saturation for tensor graph superoptimization," *Proceedings of Machine Learning and Systems*, vol. 3, pp. 255–268, 2021.

[40] C. Yu, W. Brown, D. Liu, A. Rossi, and M. Ciesielski, "Formal verification of arithmetic circuits by function extraction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 12, pp. 2131–2142, 2016.

[41] C. Yu and M. Ciesielski, "Automatic word-level abstraction of datapath," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2016, pp. 1718–1721.

[42] C. Yu, M. Ciesielski, and A. Mishchenko, "Fast algebraic rewriting based on and-inverter graphs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1907–1911, 2017.

[43] Y. Zhang, Y. R. Wang, O. Flatt, D. Cao, P. Zucker, E. Rosenthal, Z. Tatlock, and M. Willsey, "Better together: Unifying datalog and equality saturation," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 468–492, 2023.

BLOOM: Bit-Slice Framework for DNN Acceleration with Mixed-Precision

Fangxin Liu[1, 2, †, *], Ning Yang[1, 2, †], Zongwu Wang [1, 2, *] Xuanpeng Zhu[3], Haidong Yao[3], Xiankui Xiong[3], Li Jiang[1,2], and Haibing Guan[1]
[1]Shanghai Jiao Tong University [2]Shanghai Qi Zhi Institute
[3]State Key Laboratory of Mobile Network and Mobile Multimedia Technology, ZTE Corporation, Shenzhen, China
*Corresponding Author {liufangxin, yn937391832, wangzongwu}@sjtu.edu.cn

***Abstract*—Deep neural networks (DNNs) have revolutionized numerous AI applications, but their vast model sizes and limited hardware resources present significant deployment challenges. Model quantization offers a promising solution to bridge the gap between DNN size and hardware capacity. While INT8 quantization has been widely used, recent research has pushed for even lower precision, such as INT4. However, the presence of outliers—values with unusually large magnitudes—limits the effectiveness of current quantization techniques. Previous compression-based acceleration methods that incorporate outlier-aware encoding introduce complex logic. A critical issue we have identified is that serialization and deserialization dominate the encoding/decoding time in these compression workflows, leading to substantial performance penalties during workflow execution. To address this challenge, we introduce a novel computing approach and a compatible architecture design named "BLOOM". BLOOM leverages the strengths of the "bit-slicing" method, effectively combining structured mixed-precision and bit-level sparsity with adaptive dataflow techniques. The key insight of BLOOM is that outliers require higher precision, while normal values can be processed at lower precision. By interleaving 4-bit values, we efficiently exploit the inherent sparsity in the high-precision components. As a result, the BLOOM-based accelerator outperforms the existing outlier-aware accelerators by an average $1.2 \sim 4.0\times$ speedup and $24.6\% \sim 71.3\%$ energy reduction, respectively, without model accuracy loss.**

I. Introduction

Deep Neural Networks (DNNs) have made remarkable strides in recent years, delivering exceptional performance across various applications [1]–[5]. However, the increasing complexity and size of DNN models have introduced significant challenges in model inference efficiency. The size of these models has been expanding exponentially, with a staggering 15-fold increase every two years [6]–[8], far outpacing hardware advancements and resulting in escalating computational costs for inference tasks. This widening gap between model size and hardware capabilities necessitates the development of efficient model compression techniques.

Model compression has emerged as a promising method to alleviate the computational and storage burden of DNNs [9]. One of the most popular approaches is quantization, which reduces the data precision to compress models and accelerate computations. While quantization has shown impressive results in Convolutional Neural Networks (CNNs), its effectiveness in attention-based Transformer models has been less pronounced. Recent studies [10], [11] suggest that Transformer models exhibit greater variation in parameter distributions than CNNs and are more sensitive to a small fraction of extreme values. As a result, quantizing these models often requires higher bit widths, such as 8 or 16 bits, compared to the typically lower precision demands of CNNs [12]–[15].

In recent years, various compression techniques have been explored in the design of DNN accelerators. For example, as illustrated in Figure 1(a), GOBO [10] and OLAccel [16] use coordinate lists to track the location of significant values in matrices, while BiScaled-DNNs [17] leverage block sparse index formats to store these indices. OliVe [11] manually constructs significant-sparse value pairs by tightly binding patterns to the parameters, and SPARK [18] embeds indicator bits into parameters to reduce storage while maintaining alignment. However, as shown in Figure 1(b), these methods often introduce complex decoder/encoder designs and significant deserialization/serialization overhead to manage the diverse parameter distributions.

While compression-based encoding techniques effectively separate normal from significant values, they miss opportunities for further energy reduction and accuracy improvement by ignoring sparsity in the higher-order bits of quantized values. For example, an INT8 value can be split into two 4-bit parts, where the higher-order part exhibits greater sparsity than the original value. This observation motivates us to propose a software and architecture solution that efficiently handles the sparsity within the higher-order bits of quantized values.

This paper introduces **BLOOM**, a precision-aware bit-slicing scheme that splits neural network data into specific bit paradigms, achieving mixed-precision representation through an interleaved computing architecture. By exploiting the inherent sparsity within quantized parameters, BLOOM significantly reduces storage and computation demands while maintaining model accuracy, without incurring any serialization or deserialization costs. As illustrated in Figure 1(c), unlike previous compression-based methods, BLOOM does

† These authors contributed equally. This work was supported by the National Key Research and Development Program of China (2024YFE0204300), National Natural Science Foundation of China (Grant No.62402311), Natural Science Foundation of Shanghai (Grant No.24ZR1433700), and ZTE Industry-University-Institute Cooperation Funds (Grant No. IA20240409010).

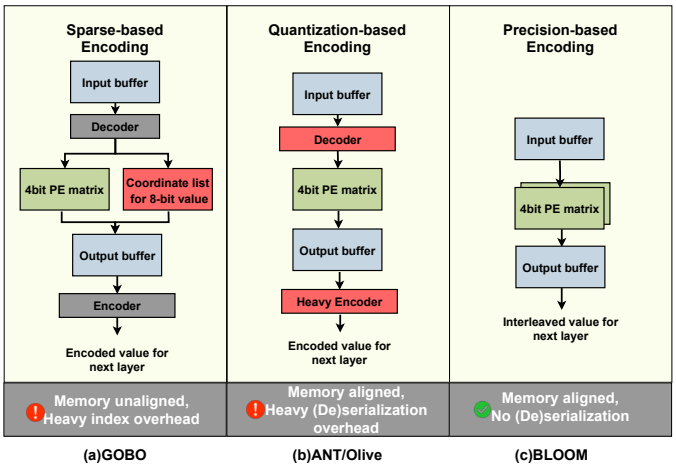


Fig. 1: Compression-based encoding comparison.

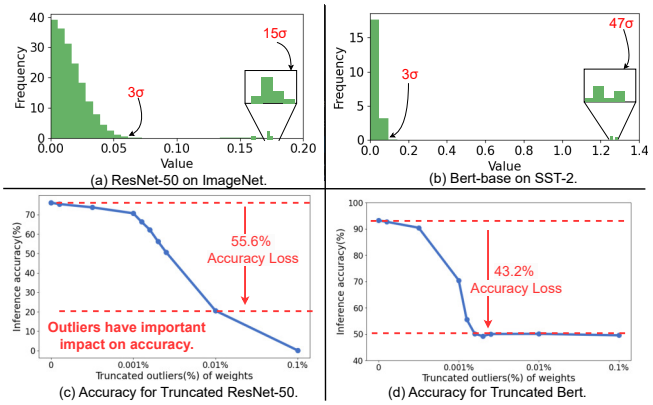


Fig. 2: (a) and (b): Outlier Comparison of CNN model and Transformer model. (c) and (d): Model Accuracy of Truncated ResNet-50 on ImageNet and Bert on SST-2.

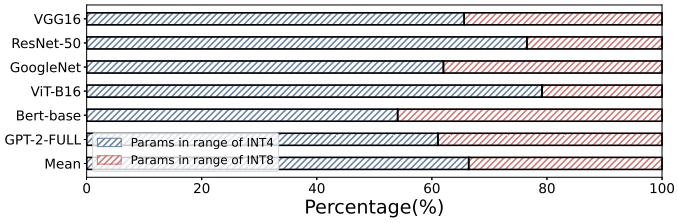


Fig. 3: In INT8 format quantized networks, the percentage of INT4 and INT8 of activation.

not require additional lists or complex encoder/decoder setups. Instead, it ensures efficient storage alignment and computation through interleaved data management.

In summary, we make the following contributions:

- We analyze the current progress in DNN compression research and identify two key challenges. To address these, we propose BLOOM, a precision-aware bit-slicing framework based on the data distribution and bit-level sparsity. BLOOM exploits the inherent high-bit sparsity in models to reduce storage requirements and enhance computational efficiency, all while preserving accuracy.
- We design a hardware-friendly, memory-interleaved computing architecture that organizes PE arrays through precision-aware processing. This ensures both computational accuracy and efficiency, without introducing any serialization or deserialization overhead.
- We implement and evaluate BLOOM, demonstrating through comprehensive experiments that it surpasses existing quantization-based encoding schemes and hardware accelerators in terms of efficiency and overall benefits. Our results highlight BLOOM's effectiveness and scalability in real-world applications.

II. Background and Challenge

A. Outlier of DNN

For DNNs, parameter distributions generally follow a Gaussian pattern, which makes them suitable for quantization [11], [19], [20]. However, a small percentage of outliers—although rare—pose significant challenges for effective quantization. As shown in Figure 2(a), for one layer in ResNet-50, over 99% of parameters are concentrated within three standard deviations (3σ), while outliers extend beyond 15 standard deviations ($> 15\sigma$). This issue is even more pronounced in transformer models like BERT, as illustrated in Figure 2(b).

If these outliers are properly managed, the vast majority of data can benefit from higher computational accuracy or reduced computational overhead. However, outliers often have a disproportionate impact on the overall accuracy of the neural network. Simply truncating or removing them can lead to irrecoverable accuracy loss. Figure 2(c) and (d) demonstrate that even directly removing as few as 0.001% of outliers can significantly degrade inference accuracy.

To preserve accuracy, outliers typically require higher-bit quantization, such as 8 or 16 bits [21]. However, this introduces bit-level redundancy for most parameters, which could be represented with lower bitwidths. In Figure 3, we show that after INT8 quantization, most parameters in both CNN-based and attention-based models can be efficiently represented using only 4 bits. Maintaining 8 bits or more for all parameters leads to unnecessary overhead.

> **Challenge 1.** DNNs exhibit inherent bit-level sparsity, but outliers make it difficult to fully exploit this sparsity.

B. Compression-based Encoding

As discussed in the previous section, effectively handling outliers is crucial for the success of quantization [10], [11], [16], [18], [21], [22]. Several compression-based encoding techniques leveraging mixed-precision quantization have been proposed: 1)Isolating outliers based on sparsity encoding: For instance, GOBO [10] uses coordinate lists to track different data representations and applies high-precision quantization to larger values. However, this approach incurs additional overhead for maintaining these lists and managing unaligned memory due to outlier removal. 2)Parameter modifications based on outlier-aware quantization: OliVe [11], for example,

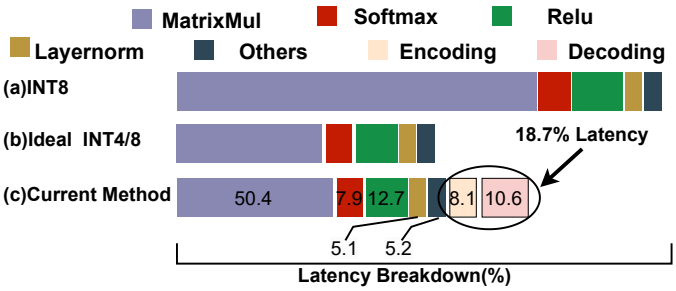


Fig. 4: Latency breakdown for different quantization methods.

applies strict constraints on parameter distribution by pruning adjacent values and storing larger ones, while SPARK [18] embeds indicator bits into parameters and uses a precision compensation mechanism to ensure mixed precision and memory alignment.

Despite reducing computational overhead, these compression techniques increase hardware complexity. The complex encoding and decoding processes introduce additional computational costs, which diminish the potential benefits of hybrid quantization. Figure 4 illustrates the performance penalties across different accelerators, showing that none achieve optimal computational efficiency.

Challenge 2. Encoding and decoding mechanisms incur serialization and deserialization overheads, which offset the performance gains from mixed precision acceleration.

Thus, due to overheads in storage, computation, or serialization/deserialization, a significant gap remains between the theoretical acceleration potential of neural network compression and the actual performance of current accelerator designs. This gap highlights a promising area for future research in DNN compression.

III. Analysis and Motivation

Quantization is an essential step for deploying neural network models, but current compression-based acceleration faces a significant challenge: **Compression Trap**. Neural networks, especially Transformers, contain non-negligible outliers that require high bitwidths to preserve accuracy, leading to substantial computational and storage redundancy. Existing compression-based methods try to reduce this redundancy by isolating or handling outliers. However, these approaches introduce serialization and deserialization overheads, negating the benefits of compression and further increasing the computational or storage burden.

To overcome this, we aim to design an acceleration scheme that uses mixed precision without requiring coding or decoding. As shown in Figure 3, a large proportion of DNN parameters can be accurately represented with 4 bits or fewer. By dividing quantized values into 4-bit segments, we can split them into low-precision and high-precision parts. Even in activation values that are less favorable for quantization, we

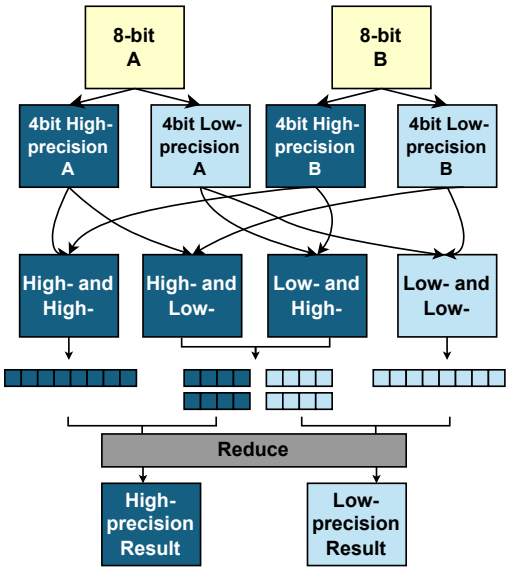


Fig. 5: An example of BLOOM calculation.

observe sparsity of 60% to 80% in the high-precision parts. This opens up new opportunities to explore more efficient quantization and compression strategies. Additionally, this 4-bit segmentation naturally aligns memory and hardware, eliminating the need for complex alignment schemes.

Building on this insight, we propose treating quantized values as independent high- and low-precision parts processed in parallel through bit-slicing, eliminating serialization or deserialization overhead. We introduce BLOOM, a precision-aware bit-slicing framework that splits 8-bit quantized parameters into two 4-bit segments for parallel computation. BLOOM then efficiently merges the results by assigning the appropriate bits to the final output, as illustrated in Figure 5. This interleaved computation leverages the sparsity of high-precision parts for energy savings and computational acceleration, without introducing additional storage or computational overhead.

IV. BLOOM Architecture Design

To fully leverage BLOOM's potential, the accelerator architecture must be customized for the framework without relying on encoders or decoders. BLOOM partitions model parameters to exploit inherent sparsity, maximizing acceleration and energy efficiency. Its precision-aware interleaved workflow optimizes computation without additional hardware and seamlessly integrates with mainstream acceleration platforms like systolic arrays. This section details the architecture.

A. Architecture Overview

This section presents how BLOOM integrates in weight-stationary systolic array architecture with minimal modifications. On such an architecture, our proposed BLOOM architecture can directly support the mixed precision, which is efficient for quantizing DNN tensors that have different importance and distribution. The architecture overview is shown in Figure 6, which includes 4-bit PEs in weight-stationary systolic array

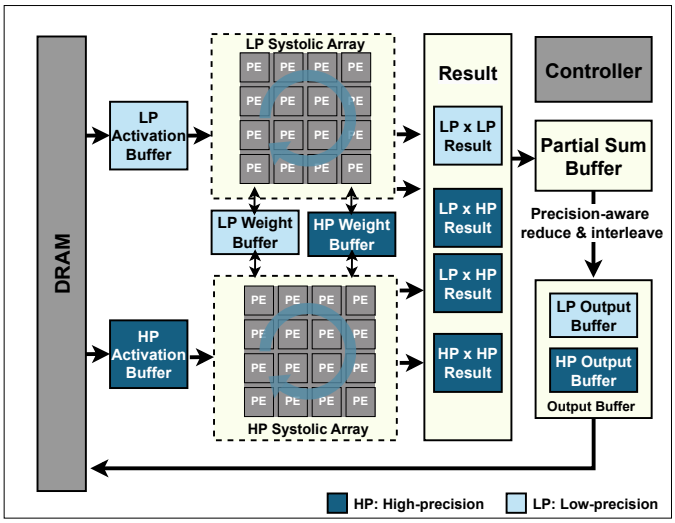


Fig. 6: Overview of BLOOM architecture.

pages, along with input/weight/output/partial sum buffers, an accumulation unit, a controller, and peripheral circuits.

A key feature of BLOOM is its precision-aware bit-slicing of model parameters into two PE groups, which are responsible for computing the high-precision (HP) and low-precision (LP) parts, respectively. These parts are interleaved during computation and stored in their respective buffers but remain separate until they converge to finalize the computation. This design allows BLOOM to fully leverage the inherent sparsity within the high-precision parts, thereby maximizing performance.

B. Joint support for the mixed-precision computation

BLOOM uses the bit-slice mechanism to separate data into a sparse high-precision part and a dense low-precision part. These parts are processed in parallel on distinct PE arrays, where the corresponding weights are also preloaded. This setup allows all computations to be performed in 4-bit precision, minimizing both computation and storage overhead. Based on the nature of the data, there are three types of computation: high-precision with high-precision, low-precision with low-precision, and high-precision with low-precision.

- **High-precision with high-precision** computations: BLOOM leverages the sparsity in the high-precision part to remove unnecessary block-wise calculations.
- **Low-precision with low-precision** computations: BLOOM performs dense 4-bit matrix multiplications, which significantly reduces computational and memory costs compared to 8-bit operations that incur additional overhead due to memory alignment requirements.
- **Low-precision with high-precision** computations: BLOOM can still benefit from the sparsity in the high-precision data. However, to maintain a balance between the two types of operations, some computations may be delegated to the high-precision array. This is because the larger sparsity in high-precision data can result in lower workloads. Such a flexible scheme ensures efficient workload balancing while still utilizing sparse acceleration where applicable. By default, BLOOM assigns high-precision computations to the HP array and low-precision computations to the LP array.

Through such an adaptive and precision-aware approach, BLOOM maximizes sparsity benefits while maintaining flexibility and efficiency across varying data precision requirements.

C. Execution flow of BLOOM

Figure 6 also illustrates the execution flow of BLOOM, which involves the following five steps:

- Data Slicing: The inputs are divided into 4-bit high-precision and 4-bit low-precision parts. Such a partitioning is performed in storage using the corresponding buffers, without any encoding processes.
- Weight Setup: Weights are pre-split into high-precision and low-precision components and stored in the respective buffers. These weights are fixed in the systolic array prior to computation.
- Parallel Computation: The partitioned inputs and weights are processed in a 4-bit PE array. The high-precision data, due to its inherent sparsity, benefits from acceleration mechanisms, while the low-precision data is handled in a dense format. As detailed in Section IV-B, computations are categorized into three types, with the majority benefiting from the sparsity in high-precision data.
- Partial Sum Accumulation: Instead of directly aggregating results from each case, BLOOM accumulates partial sums in a buffer. This approach ensures intermediate results are managed efficiently.
- Precision-Aware Reduction: BLOOM performs a reduction operation on the accumulated partial sums to produce 8-bit output results. These results are then interleaved, with the high-precision and low-precision parts separated and sent to the output buffer.

Overall, BLOOM efficiently handles models with outliers without wasting number representations. By leveraging the inherent sparsity of the high-precision parts and using interleaving techniques in computation and storage, BLOOM avoids additional costs. This design allows a memory-aligned architecture to manage outlier values with minimal hardware overhead while achieving significant performance gains. BLOOM can be seamlessly integrated into existing hardware accelerators, such as systolic arrays.

D. Workload-aware PE Configuration

Due to the varying workloads between high-precision and low-precision computations, the completion times of the two types of PE arrays in BLOOM differ. This makes proportional workload allocation critical. Although both types use 4-bit PEs with the same computational speed, the sparsity in the high-precision data results in a lower workload. Therefore, more PE arrays must be allocated to low-precision computations, or additional tasks should be assigned to high-precision PE arrays to prevent bottlenecks.

However, dynamically adjusting the workload during runtime may introduce additional overhead, potentially offsetting

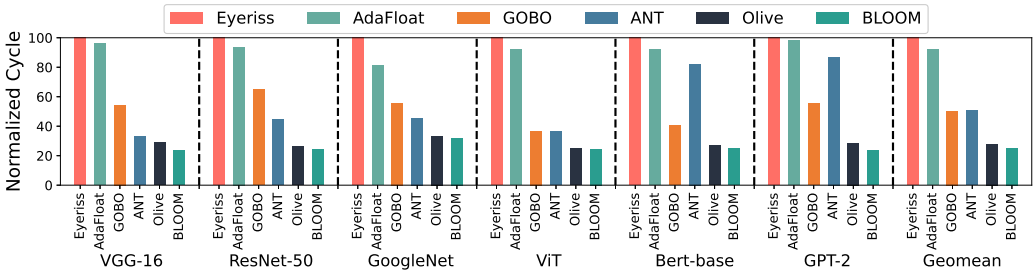


Fig. 7: Comparison of the normalized latency in different designs.

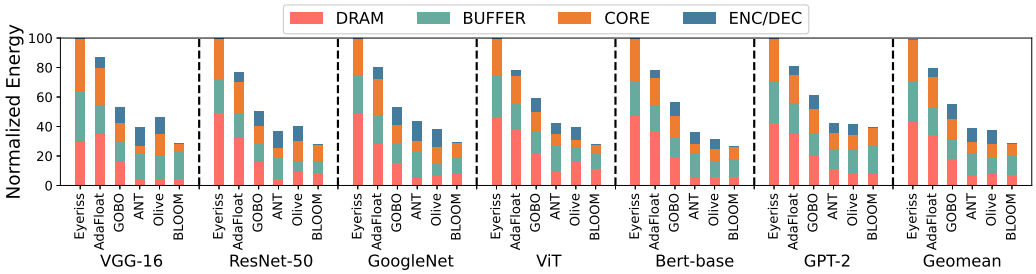


Fig. 8: Comparison of the normalized energy in different designs.

the performance gains. To end this, we analyze the workloads of high-precision and low-precision parts within weights offline. By examining the workload distribution, we can allocate the appropriate number of PEs for high- and low-precision tasks for different models, ensuring that the processing times for both PE arrays are as balanced as possible.

For mixed-precision computations, this offline analysis allows for flexible task deployment, which helps optimize overall efficiency and maintain a balance between different types of computations. This approach ensures that performance is maximized across various workloads. Further details on this balance will be explored in Section V-C.

V. Evaluation

A. Methodology

Benchmark. To evaluate the performance of the BLOOM framework, we implement it using the PyTorch [23] framework. We select three representative networks: VGG-16, ResNet-50, and GoogleNet, running them on the ImageNet dataset [24]. To demonstrate BLOOM's scalability, we also evaluate mainstream attention-based models: ViT [25] with ImageNet, Bert-base with the GLUE dataset, and GPT2-medium with the WikiText-103 dataset [26]. Additionally, we implement the proposed BLOOM architecture in Verilog and synthesize it using Synopsys Design Compiler [27] to estimate chip area and total power consumption under 45nm technology at 500MHz [28]. We also use CACTI [29] to model the energy and area of each on-chip buffer.

Baseline. We compare BLOOM with the following five baselines: 1) Eyeriss [30], [31], which is a spatial energy-efficient dataflow architecture. It uses coarse-grained quantization of INT16 throughout the network, and it is a mainstream NN accelerator architecture. In the evaluation, we will use it as the baseline of the standard DNN accuracy. 2) AdaptivFloat [32], or AdaFloat, which dynamically clips parameter range at a layer granularity to create faithful encodings, and requires an 8-bit float to maintain the original model accuracy; 3) GOBO [10], which employs a list of coordinates to indicate the locations of different data representations, using high-precision quantization for larger data representations. 4) ANT [33], which is a hardware-friendly quantization accelerator that combines the power-of-two data types and INT type for low bit-width; 5) Olive [11], which is outlier-aware quantization accelerator. We take reported results in their paper for comparison, and for models that are not available in their paper, we reproduce their experiments and report the results.

TABLE I: Detailed result of evaluated model and dataset.

Type	CNN-based			Attetion-based		
Model	VGG-16	ResNet-50	GoogleNet	ViT	BERT	GPT2-XL
Dataset	ImageNet				SST-2	Wikitext
FP-32 Acc.(%)	71.59	76.15	62.46	84.19	93.23	16.94 (PPL)
BLOOM Acc.(%)	71.59	76.05	62.35	84.03	92.78	17.15 (PPL)

B. Experimental Results

Accuracy. For quantized DNN models, the BLOOM framework leverages intrinsic sparsity with minimal accuracy loss. As shown in Table I, BLOOM achieves nearly original accuracy for commonly used vision and language models. On the ImageNet dataset, BLOOM exhibits an average accuracy loss less than 0.1% against the original FP32 model. And for LLM like GPT2, deploying the BLOOM framework also has nearly no effect on the perplexity. This minimal impact on accuracy is due to BLOOM's use of an 8-bit format for computations, which does not alter the computation process or results. The experimental results demonstrate that BLOOM effectively maintains model accuracy while delivering performance acceleration, even for LLM.

Performance. Figure 7 illustrates the normalized total execution cycles across different accelerators for six networks. BLOOM shows the most significant performance improvement. Compared to other accelerators, BLOOM achieves

TABLE II: Configurations of different accelerators under 45nm process.

	Eyeriss [30]	GOBO [10]	ANT [33]	Olive [11]	BLOOM
#PEs	224	4096	4096	4096	4096
Bitwidth	INT16	INT4	INT4	INT4	INT4
Decoder	-	256	128	192	-
Area(mm2)	0.32	0.417	0.327	0.338	0.32

substantial speedups, with average improvements of $3.96\times$, $3.64\times$, $1.99\times$, $2.01\times$, and $1.20\times$ over Eyeriss, AdaFloat, GOBO, ANT, and Olive, respectively. Notably, BLOOM achieves a $3.69\times$ speedup over ANT on GPT2, highlighting its efficiency. GOBO and ANT perform less effectively on most models, with ANT especially struggling with attention-based models due to their complex coding and decoding mechanisms. OliVe, despite its optimization for decoding, suffers from a complex encoding process and a limited percentage of outlier pairs, resulting in less impressive speedups compared to BLOOM. This is because BLOOM eliminates the serialization and deserialization overhead typically associated with encoding and decoding, which leads to a significant acceleration of calculations.

Energy. Figure 8 shows the normalized energy consumption across six networks, broken down into DRAM, global buffer (labeled as BUFFER), processing core (labeled as CORE), and encoder/decoder (labeled as ENC/DEC). BLOOM exhibits the lowest energy consumption for both CNN-based and attention-based models. On average, BLOOM consumes 71.3%, 64.4%, 48.4%, 26.8%, and 24.6% less energy compared to Eyeriss, AdaFloat, GOBO, ANT, and Olive, respectively. Since other accelerators require complex encoder/decoder mechanisms that add significant energy overhead, BLOOM's absence of these components provides a clear advantage. Even with attention-based models, which have a larger number of parameters, BLOOM maintains substantial energy efficiency due to its simple and efficient interleaved computation flow.

Area. Table II provides a comparison of the area and number of PEs for BLOOM and other accelerators, all scaled to a 45-nm process. The table also considers the area required for decoders, where applicable. BLOOM maintains a consistent total PE area with SA while minimizing area overhead. Other accelerators, such as GOBO, incur significant area overhead due to the inclusion of decoders. In contrast, BLOOM's use of compact 4-bit PEs and its lack of a decoder result in the lowest area overhead. This design choice allows BLOOM to fit more PEs within the same area budget, leading to enhanced computational performance.

C. Design Space Exploration

This section discusses workload balancing between high- and low-precision PE arrays, as introduced in Section IV-B and Section IV-D. We then analyze BLOOM's scalability by evaluating its performance across various model sizes.

Workload-aware PEs configuration. To evaluate the performance gains of BLOOM under different scenarios and determine the optimal ratio of high- to low-precision PE arrays, we conducted experiments with three models. We compared

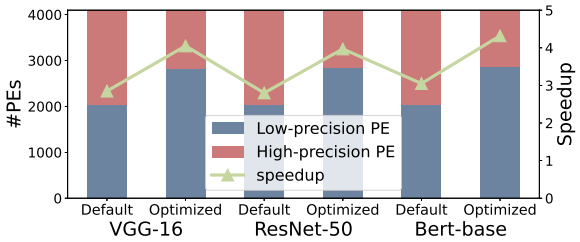


Fig. 9: Speedup of different models under different settings.

two cases: 1) assigning an equal number of PE arrays for high- and low-precision computations, and 2) adjusting the number of PE arrays for each precision type to ensure that the high-precision and low-precision PE arrays can complete their computations nearly simultaneously.

Figure 9 illustrates the results. Allocating an equal number of high- and low-precision PE arrays yields a significant speedup of approximately $3\times$, primarily due to BLOOM's use of 4-bit PEs. However, due to the inherent computational speed differences between the two types of arrays, low-precision computations can become a bottleneck. By adjusting the PE array allocation, we enhance the computational parallelism of the low-precision part, achieving a more pronounced speedup of over $4\times$. The effective allocation of PE arrays leverages BLOOM's advantages, resulting in significant improvements in computational efficiency.

Scalability of BLOOM. We analyze how increasing model sizes affect the computation speedup achieved by BLOOM. As shown in Figure 10, as model sizes increase, the computation speedup with BLOOM framework also rises. Such a trend is attributed to BLOOM's architecture design, where higher sparsity can get a greater pronounced speedup effect. Notably, as the model size increases, the sparsity naturally embedded in the parameters increases, which further emphasizes the advantages of the BLOOM architecture. These findings demonstrate BLOOM's scalability and consistent performance in enhancing energy efficiency, especially for transformer-based models with large parameter sizes.

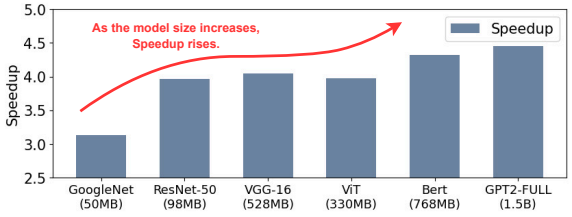


Fig. 10: Speedup with different model size.

VI. Conclusion

We introduce BLOOM, a precision-aware bit-slicing framework for efficient DNN acceleration with minimal hardware overhead. BLOOM leverages the inherent sparsity in high-precision quantized values caused by outliers, avoiding serialization and deserialization costs. It seamlessly integrates with existing platforms through interleaved computational streams, enabling mixed-precision encoding without information loss. Evaluations show that BLOOM outperforms similar quantization schemes in performance, energy efficiency, and accuracy.

REFERENCES

[1] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digital signal processing*, vol. 73, pp. 1–15, 2018.

[2] E. Ozen and A. Orailoglu, "Evolving complementary sparsity patterns for hardware-friendly inference of sparse dnns," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.

[3] F. Liu, W. Zhao, Y. Chen, Z. Wang, Z. He, R. Yang, Q. Tang, T. Yang, C. Zhuo, and L. Jiang, "Pim-dh: Reram-based processing-in-memory architecture for deep hashing acceleration," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1087–1092.

[4] L. YUAN, C. HUI, Y. WU, R. LIAO, F. JIANG, and Y. GAO, "Video enhancement network based on cnn and transformer," *ZTE Communications*, vol. 22, no. 4, p. 78, 2024.

[5] L. Deng and Y. Zhao, "Deep learning-based semantic feature extraction: A literature review and future directions," *ZTE communications*, vol. 21, no. 2, p. 11, 2023.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[8] OpenAI, "Gpt-4 technical report," 2023.

[9] F. Liu *et al.*, "Improving neural network efficiency via post-training quantization with adaptive floating-point," in *ICCV*, 2021.

[10] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[11] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[12] Z. Song, B. Fu, F. Wu, Z. Jiang, L. Jiang, N. Jing, and X. Liang, "Drq: dynamic region-based quantization for deep neural network acceleration," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 1010–1021.

[13] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[14] S.-E. Chang, Y. Li, M. Sun, R. Shi, H. K.-H. So, X. Qian, Y. Wang, and X. Lin, "Mix and match: A novel fpga-centric deep neural network quantization framework," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 208–220.

[15] F. Liu, W. Zhao, Z. Wang, Y. Chen, Z. He, N. Jing, X. Liang, and L. Jiang, "Ebsp: evolving bit sparsity patterns for hardware-friendly inference of quantized deep neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 259–264.

[16] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 688–698.

[17] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, K. Gopalakrishnan, and L. Chang, "Biscaled-dnn: Quantizing long-tailed datastructures with two scale factors for deep neural networks," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[18] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 1029–1042.

[19] J. Shen, K. Jiang, and X. Tan, "Boundary data augmentation for offline reinforcement learning," *ZTE Communications*, vol. 21, no. 3, p. 29, 2023.

[20] N. Yang, F. Liu, Z. Wang, J. Zhao, and L. Jiang, "Searchq: Search-based fine-grained quantization for data-free model compression," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.

[21] X. Wei, Y. Zhang, X. Zhang, R. Gong, S. Zhang, Q. Zhang, F. Yu, and X. Liu, "Outlier suppression: Pushing the limit of low-bit transformer language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 402–17 414, 2022.

[22] F. Liu, N. Yang, Z. Song, Z. Wang, H. Li, S. Huang, Z. Song, S. Pei, and L. Jiang, "Inspire: Accelerating deep neural networks via hardware-friendly index-pair encoding," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[23] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[24] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[25] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[26] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[27] S. D. Compiler, [Online]. Available: https://www.synopsys.com/support/training/rtlsynthesis/design-compiler-rtl-synthesis.html, 2019.

[28] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[29] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[30] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[31] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, 2019.

[32] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei, "Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[33] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

PISA: Efficient Precision-Slice Framework for LLMs with Adaptive Numerical Type

Ning Yang[1,2], Zongwu Wang[1,2], Qingxiao Sun[3], Liqiang Lu[4] and Fangxin Liu[1,2,*]
[1]Shanghai Jiao Tong University, [2]Shanghai Qi Zhi Institute, [3]China University of Petroleum-Beijing, [4]Zhejiang University
*Corresponding Author {yn937391832, liufangxin}@sjtu.edu.cn

***Abstract*—Large language models (LLMs) have transformed numerous AI applications, with on-device deployment becoming increasingly important for reducing cloud computing costs and protecting user privacy. However, the astronomical model size and limited hardware resources pose significant deployment challenges. Model quantization is a promising approach to mitigate this gap, but the presence of outliers in LLMs reduces its effectiveness. Previous efforts addressed this issue by employing compression-based encoding for mixed-precision quantization. These approaches struggle to balance model accuracy with hardware efficiency due to their value-wise outlier granularity and complex encoding/decoding hardware logic. To address this, we propose PISA (Precision-Slice Framework), an acceleration framework that exploits massive sparsity in the higher-order part of LLMs by splitting 16-bit values into a 4-bit/12-bit format. Crucially, PISA introduces an early bird mechanism that leverages the high-order 4-bit computation to predict the importance of the full calculation result. This mechanism enables efficient computational skips by continuing execution only for important computations and using preset values for less significant ones. This scheme can be efficiently integrated with existing hardware accelerators like systolic arrays without complex encoding/decoding. As a result, PISA outperforms state-of-the-art precision-aware accelerators, achieving a $1.3-4.3\times$ performance boost and $14.3-66.7\%$ greater energy efficiency, with minimal model accuracy loss. This approach enables more efficient on-device LLM deployment, effectively balancing computational efficiency and model accuracy.**

I. Introduction

Transformer-based large language models (LLMs) have achieved remarkable success [1], but their large model sizes and high computational demands pose significant challenges. For instance, GPT-175B requires at least 350GB of storage for FP16 inference, equivalent to five 80GB A100 GPUs [2].

Quantization [3]–[7] is one of the most effective ways to reduce LLM inference costs. By quantizing weights and activations to lower bit-widths, both memory and bandwidth demands can be greatly reduced, while accelerating computations. Currently, LLM weights can be quantized to 4-bit with minimal accuracy impact [4], [6], [8]. However, quantizing activations remains challenging due to outlier issues. A small fraction of activation values with extremely large magnitudes significantly impacts model accuracy, making it difficult to apply low bit-width (4-bit) activation quantization without sacrificing model accuracy.

Previous works, such as OLAccel [9], GOBO [10], and OliVe [11], have proposed precision-aware quantization with dedicated hardware support. OLAccel and GOBO use mixed-precision quantization (4-bit for normal values, 16-bit for outliers) to achieve near 4-bit average precision with negligible accuracy loss, but they treat outliers as sparse matrices, requiring additional codebooks and complex control logic. OliVe introduces outlier-victim pair encoding with localized outlier storage to ensure aligned memory access, but it limits outlier bit-width extension and compromises hardware efficiency by adding encoders and decoders. While these methods effectively separate normal values from outliers, they miss opportunities for further energy reduction and accuracy improvement by ignoring the inherent sparsity in the higher-order bits of quantized values [12]–[14]. For example, an INT8 value can be split into two 4-bit parts, where the high-order part shows greater sparsity than the original value.

Motivated by this observation, we propose PISA (Precision-Slice Framework with Adaptive Numerical Type), which splits model data into 4-bit high-order and 12-bit low-order parts for mixed-precision representation using an interleaved computing architecture. PISA focuses on high-order 4-bit computation, exploiting the inherent sparsity in quantization parameters through an early bird mechanism. This mechanism predicts the importance of full calculations using partial sums from high-order computation: important results are completed, while less significant ones are skipped and replaced with a preset value. By eliminating unnecessary calculations, PISA significantly reduces computational energy and time. Rather than relying on complex encoder/decoder configurations, PISA ensures efficient storage alignment through interleaved data management and maintains model accuracy with optimized thresholds and presets.

We make the following contributions in this paper:

- We propose PISA, an precision-slice framework based on data distribution and bit-level sparsity. Using a 4bit-12bit unbalanced splitting, PISA exploits inherent high-bit sparsity to reduce storage requirements and enhance computational efficiency while preserving accuracy.
- We introduce an early bird mechanism that leverages the partial sum from the high-order 4-bit computation

This work was supported by the National Key Research and Development Program of China (2024YFE0204300), National Natural Science Foundation of China (Grant No.62402311), Natural Science Foundation of Shanghai (Grant No.24ZR1433700).

to predict the importance of the full calculation. This mechanism enables efficient computational skips for less significant values, further reducing computational costs without compromising model accuracy.

- We design a hardware-friendly, memory-interleaved computing architecture that organizes PE arrays through precision-aware processing. This architecture efficiently handles the 4bit-12bit split representation and implements the early bird mechanism using a simple comparator, significantly accelerating computation.
- We implement and evaluate PISA, demonstrating its superiority over existing quantization-based encoding schemes and hardware accelerators. Evaluations show significant improvements in both efficiency and overall performance, highlighting PISA's effectiveness and scalability in real-world LLM applications.

.

II. Background

A. Outliers within Transformer Models

Transformer model parameters typically follow a Gaussian-like distribution, suitable for uniform quantization [11], [13]. However, a small fraction of parameters, termed outliers, exhibit extremely large magnitudes. Figure 1(a) shows the distribution of activation and weight parameters for a single GPT2-XL layer. While 99.67% of parameters fall within (3σ) of a fitted Gaussian distribution, outliers extend to tens or even hundreds of standard deviations.

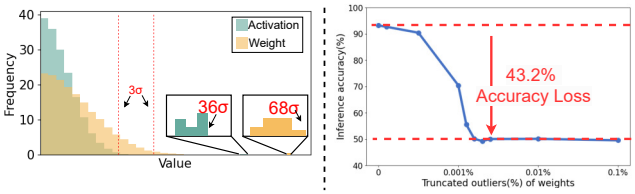
These outliers, despite their rarity, significantly impact overall model accuracy. We experimentally remove the largest part of the parameter values, as shown in Figure 1(b). The accuracy result demonstrates that removing as few as 0.001% of outliers can substantially degrade inference accuracy.

Key Insight 1: Outliers are crucial for preserving model accuracy, making their proper management and accurate representation essential for effective model compression.

To maintain accuracy, outliers often require higher-bit representation (8 or 16 bits) [15]. However, this approach introduces bit-level redundancy in data representation. While most data falls within a range that could be represented with lower bitwidths, the presence of outliers forces the use of a uniform higher precision for all parameters, resulting in significant bit redundancy. Figure 2 illustrates that activations and weights stored in INT16 format, most parameters exhibit massive bit redundancy in their high-order part, particularly in the top 4 bits. This uniform high precision for all parameters leads to unnecessary storage and computational overhead.

Key Insight 2: Redundancy from outliers can be eliminated or mitigated by separately storing and processing the high-order and low-order parts of data representation.

The challenge lies in developing a method to effectively identify and handle outliers without compromising model accuracy. Traditional approaches struggle to balance preserving critical outlier information with efficient compression. In summary, outliers in transformer models present both challenges and opportunities for compression. This calls for the efficient design that adaptively handle parameters based on their importance to overall model performance.



(a) GPT2-XL on Wikitext2. (b) Accuracy for Truncated Bert.

Fig. 1: (a): Outlier Comparison GPT2-XL on Wikitext-103. (b): Model Accuracy of Truncated Bert on SST-2.

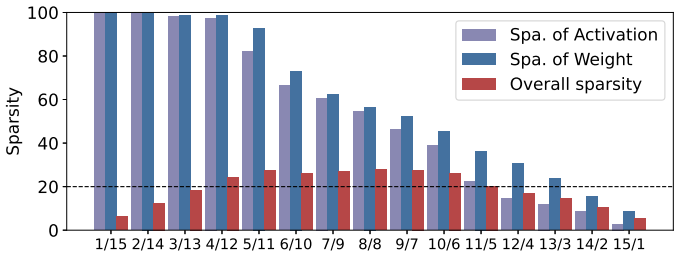


Fig. 2: Activation, weight, and overall sparsity across precision-slice methods. 'x/y' denotes slicing a 16-bit parameter into x-bit and y-bit parts and computing their sparsity.

B. Transformer Accelerators

Previous works [4]–[6], [16]–[19] have proposed various quantization/architecture co-design methods to address the activation outlier challenge in LLMs. Several quantization algorithms smooth out the activation distribution by scaling down outlier magnitudes, enabling uniform quantization. However, encoding outliers with the same low bit-width as normal values fails to provide sufficient accuracy.

To maintain model accuracy, several outlier-aware architectures have been proposed. For instance, OLAccel [9] and GOBO [10] store high-precision outliers separately from normal values and employ dedicated processing units for outliers. However, these approaches face challenges due to variable-length outlier storage resulting from random occurrence, leading to unaligned memory access. Additionally, they require complex control logic to index outliers and orchestrate computation between normal and outlier values. OliVe [11] attempts to address these issues by locally storing outlier values and allocating co-located bit-width for indexing. While this approach achieves aligned memory access and reduces control overhead, it can only extend limited bit-width to represent outlier values, potentially leading to insufficient precision and accuracy loss. Moreover, to accommodate outlier values with local storage and random occurrence, OliVe extends all computing units to enable outlier processing, which adversely impacts its hardware efficiency.

Despite reducing computational overhead, these compression techniques increase hardware complexity. Complex encoding and decoding processes introduce additional computational costs, diminishing the potential benefits of mixed-precision quantization. Figure 3 shows that encoding/decoding

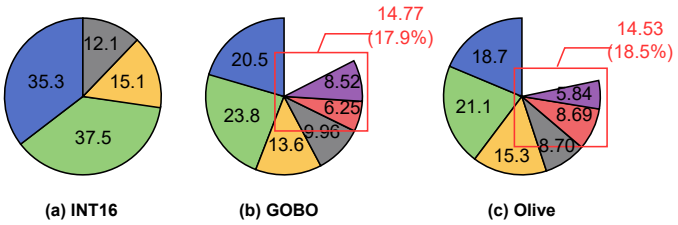


Fig. 3: Latency breakdown for different quantization methods.

consumes nearly 20% of latency in quantization-based accelerators, significantly offsetting acceleration gains.

Key Insight 3: Current mixed precision acceleration methods, despite their potential, often compromise overall performance due to the trade-off between precision and hardware complexity. This highlights the need for more efficient approaches that better balance accuracy and acceleration.

III. PISA ALGORITHM

A. Main Idea and Design Overview

Transformers contain non-ignorable outliers that require high bit-widths to maintain accuracy, leading to significant computational and storage redundancy. Existing compression-based approaches attempt to reduce this redundancy by isolating or handling outliers. However, these methods introduce serialization and deserialization overheads that offset compression benefits and increase computational or storage burdens.

To this end, we propose an outlier-aware acceleration scheme that avoids complex overhead. Our approach quantizes weights and activations into 16-bit integers, ensuring comprehensive support for a wide range of model sizes and architectures. As illustrated in Figure 2, slicing the quantized values reveals considerable overall sparsity, even for unfavorably quantized activation value weights. After thorough analysis, we opted for a 4-12 bit partitioning paradigm: 4 bits for the higher-order part and 12 bits for the lower-order part.

Such a partitioning offers several advantages. The 4-bit higher-order part exhibits over 95% sparsity, even for activations, opening new avenues for efficient quantization and compression. It also helps identify important outliers, enabling selective computation skipping for faster processing. The 12-bit lower-order part can be divided into three 4-bit segments, naturally aligning with memory and hardware, eliminating the need for complex alignment schemes [20].

Based on these insights, we introduce PISA (Outlier-aware Precision-Slice), a framework that treats quantized values as two independent parts—high-precision and low-precision—processed in parallel through precision-slicing. PISA splits 16-bit quantized model parameters into 4-bit and 12-bit segments, both participating in parallel computation in 4-bit format. Through interleaved computation and simple comparators, PISA leverages the sparsity of higher-order parts to achieve energy savings and performance acceleration.

B. Precision-Slice Computation of PISA

Based on the $4/12$ bit division, we propose a precision-slicing computation flow with an **Early Bird** mechanism. This approach leverages the properties of outliers and the sparsity of the higher-order part to accelerate LLM computation, as illustrated in Figure 4.

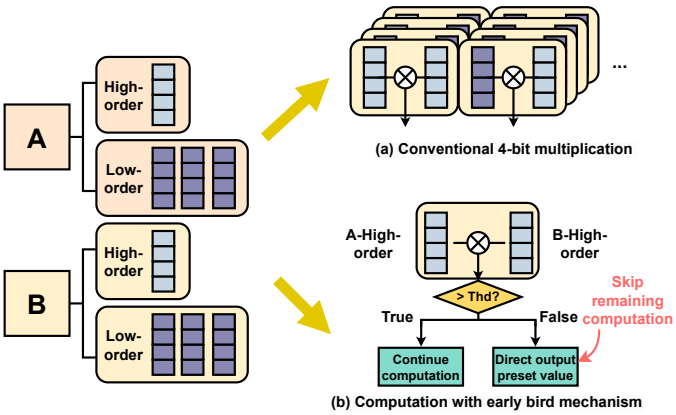


Fig. 4: PISA Computation Mechanism: (a) Conventional 4-bit multiplication, (b) PISA's early bird mechanism.

For 16-bit numbers A and B, we split and execute multiplication in 4-bit format (Figure4(a)). The early bird mechanism (Figure4(b)) leverages the partial sum from the high-order 4-bit computation to predict the importance of the full calculation. When computing the product of A and B, we first calculate the product of their higher-order parts and perform a threshold detection. If this partial result exceeds a predetermined threshold, we consider the calculation "significant" to the neural network and proceed to complete the full computation. Conversely, if the result is below or equal to the threshold, we deem it less important, mask the result to a preset value, and skip the remaining calculations.

This approach enables efficient computational skips for less significant values, substantially reducing overall computational costs without compromising model accuracy. To optimize the trade-off between efficiency and accuracy, we determine the optimal threshold and preset values through offline layer-wise validation. This process involves running the validation set and fine-tuning these parameters to minimize any potential accuracy loss while maximizing computational savings.

By combining the early bird mechanism with the intrinsic sparsity of higher-order bits, PISA accelerates LLM computation, improving the performance of both attention and linear modules in the Transformer.

IV. PISA ARCHITECTURE

This section describes integrating PISA into a systolic array architecture with fixed weights. With minimal architectural modifications, PISA delivers substantial acceleration benefits. Notably, it requires no additional encoders/decoders or extra storage overhead, offering a distinct advantage over other accelerators.

A. Overview

As illustrated in Figure 5(a), PISA architecture consists of multiple PE pages, each containing multiple PE arrays capable of performing 16-bit computations in a 4-bit format. Each

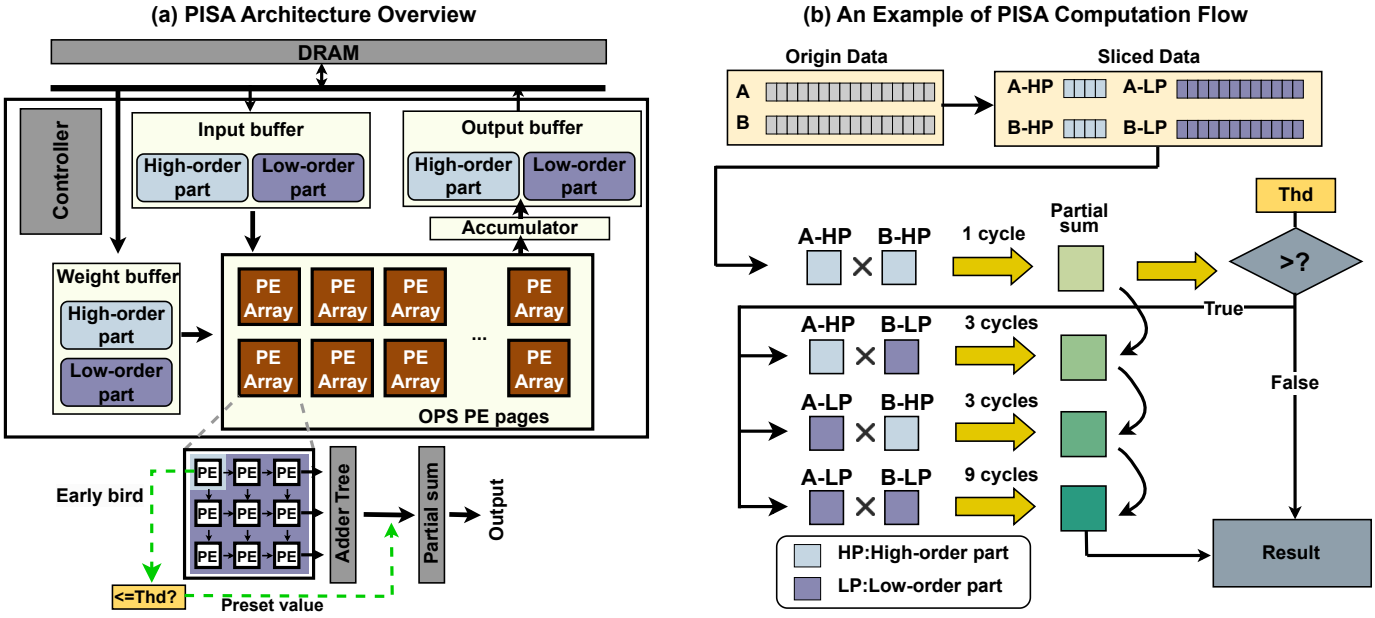


Fig. 5: PISA (a) Architecture Overview and (b) An Example of Computation Flow.

PE page is equipped with a controller, input buffer, weight buffer, output buffer, partial sum buffer, an accumulator, and associated peripheral circuits with control logic.

A key feature of the PISA architecture is its precise categorization of model parameters according to precision-slicing. Parameters are divided into higher-order and lower-order parts based on the slicing strategy and stored in their respective buffers. These parts remain separate and are only intertwined during computation. After computation, they are again separated before output, and not merged until just before the final output is performed.

This design leverages the sparsity and importance of the high-order part for computational acceleration. By keeping high and low-order parts separate, PISA efficiently applies its early bird mechanism, skipping unnecessary computations. This approach leads to significant performance gains without compromising accuracy.

B. PE Design

The PISA computation process shares similarities with conventional systolic arrays but introduces two key differences: 4-bit precision computation and an early bird acceleration mechanism. Figure 5(a) illustrates this design.

PE arrays efficiently process inputs and weights independently, feeding results into an adder tree and summarizing them in a partial sum buffer for final output. We implement a simple logical modification to prioritize high-order part computations, allowing straightforward output from a single PE for comparison with a fixed threshold Thd. When computation results fall below or equal to this threshold, the early bird mechanism activates, bypassing remaining high-order with low-order and low-order with low-order operations. Instead, it inputs a preset value to the partial sum buffer.

To demonstrate the early bird mechanism's efficacy, Figure 5(b) depicts the PISA architecture's computational flow. Consider a scenario with a single 4-bit PE for sliced data: traditional computation requires 16 cycles to process two numbers (A and B) and sum them. In contrast, PISA computes higher-order parts of A and B in the first cycle, comparing results against the threshold to determine whether to continue. If interrupted, the subsequent 15 computation cycles are skipped. For systolic arrays requiring seven cycles, PISA can potentially save six cycles.

This PE design exploits both the sparsity and the significance of higher-order bits in 16-bit computations. It often requires only a single 4-bit computation to obtain results in most cases, leveraging the fact that higher-order bits contribute more significantly to the final outcome. As a result, this approach substantially reduces computational overhead and energy consumption, thereby enhancing overall efficiency.

C. PISA Execution Flow

Figure 5 illustrates the execution flow of PISA computation, which comprises five steps:

- **Data Slicing:** Input data is partitioned into 4-bit high-order and 12-bit low-order components. This division occurs directly in storage using dedicated buffers, eliminating the need for additional encoding processes.
- **Weight Preparation:** Weights are pre-partitioned into high-order and low-order components and stored in their respective buffers. These pre-processed weights are then fixed in the systolic array prior to computation.
- **Early Bird Computation:** The partitioned inputs and weights enter the PE array in 4-bit format. For each computation set, PISA initially processes the high-order parts and determines whether to activate the early termination mechanism. If activated, the remaining computations in the set are bypassed, and predetermined values are used to populate the partial sum buffers. This approach leverages the sparsity of higher-order data to significantly accelerate computation.

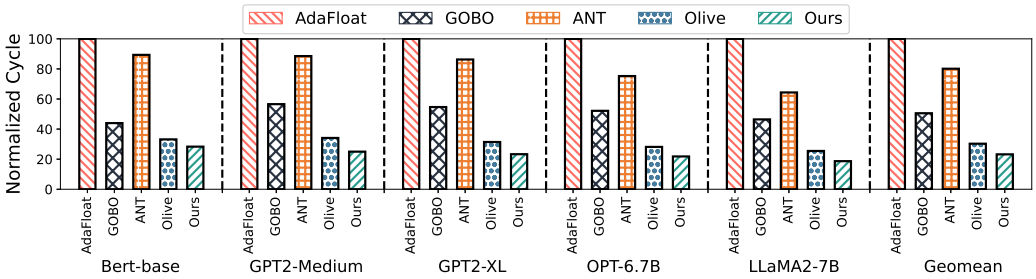


Fig. 6: Comparison of the normalized latency in different designs.

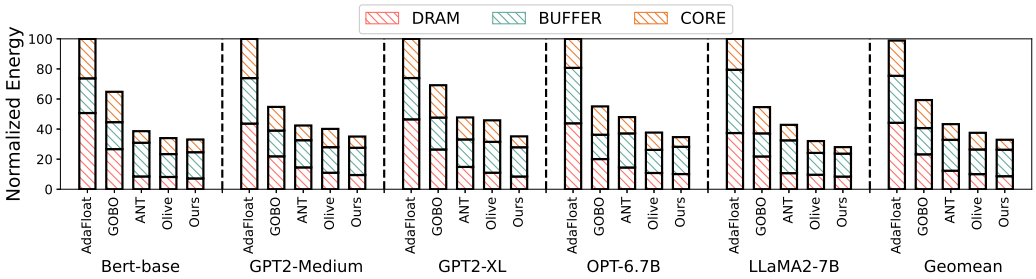


Fig. 7: Comparison of the normalized energy in different designs.

TABLE I: Accuracy and Perplexity of LLMs with PTQ. Lower perplexity (PPL) indicates better performance.

Model	BERT	GPT2-medium	GPT2-XL	OPT-6.7B	LLaMA-7B
Dataset	SST-2 (Acc.) ↑	Wikitext-103 (PPL)↓			
FP32	93.23	22.76	16.94	10.86	5.68
Olive 8/16-bit [11]	92.88	25.42	19.49	18.34	36.47
PISA 8/16-bit	92.77	23.55	18.15	13.11	8.98

- **Partial Sum Accumulation:** Rather than directly aggregating results from each case, PISA accumulates partial sums in a dedicated buffer. This strategy ensures efficient management of intermediate results.
- **Precision-Aware Reduction:** PISA performs a reduction operation on the accumulated partial sums to generate 16-bit output results. These results are then stored in an interleaved format, with the 4-bit high-precision and 12-bit low-precision parts separated before being transferred to the output buffer.

PISA efficiently handles LLMs with outliers by exploiting the inherent sparsity in high-order bits introduced by these outliers. This approach, combined with selective computation skipping, achieves significant performance improvement without incurring additional costs. Moreover, due to the absence of coding and decoding mechanisms, PISA can be seamlessly integrated into existing accelerators, such as systolic arrays.

V. Evaluation

A. Methodology

Benchmark. To evaluate the performance of the PISA framework, we implement it using the PyTorch [21] framework. We select several widely-accepted transformers: Bert, GPT2-medium, GPT2-XL, OPT-6.5B [22], and LLaMA-7B. Bert is tested on the GLUE dataset suite [23], and the remaining models are validated using the Wikitext-103 dataset [24]. Additionally, we implement the proposed PISA architecture in Verilog and synthesize it using SynPISAys Design Compiler [25] to estimate chip area and total power consumption under 45nm technology at 500MHz [26]. We also use CACTI [27] to model the energy and area of each on-chip buffer.

Baseline. We compare PISA with the following four baselines: 1) AdaptivFloat [28], or AdaFloat as short, which dynamically clips parameter range at a layer granularity to create faithful encodings, and requires high-precision float to maintain the original model accuracy; 2) GOBO [10], which employs a list of coordinates to indicate the locations of different data representations, using high-precision quantization for larger data representations. 3) ANT [29], which is a hardware-friendly quantization accelerator that combines the power-of-two and INT data type for low bit-width; 4) Olive [11], which is outlier-aware quantization accelerator. We reproduce the method based on the methods provided in these papers and expand the data format from 8 bits to 16 bits to make fair comparisons.

B. Experimental Results

Accuracy/Perplexity. Table I shows the evaluation results of PISA on the SST-2 and Wikitext-103 datasets. The model under the PISA framework maintains very low accuracy loss and excellent perplexity. These results demonstrate that even with most computations skipped through the early bird mechanism, PISA can maintain model accuracy while providing acceleration for LLMs. To summarize, our PISA quantization framework pushes the limit of quantization to a new state-of-the-art, as it achieves nearly original accuracy for commonly used LLMs while significantly reducing computational costs.

Performance. Figure 6 illustrates the normalized total execution cycles across different accelerators for five networks. PISA demonstrates superior performance improvement, achieving substantial speedups with average improvements of $4.31\times$, $2.19\times$, $3.45\times$, and $1.30\times$ over AdaFloat, GOBO, ANT, and Olive, respectively. Notably, PISA achieves a $3.70\times$ speedup over ANT on GPT2-XL. GOBO and ANT perform less effectively on most models, especially struggling with

TABLE II: The core's configuration and area breakdown of PISA and other work under 45nm process.

Architecture	Core		
	Component	Number	Area(mm^2)
PISA	Comparator (3.5μm^2)	64	0.387
	8-bit PE(377.5 μm^2)	1024	
Olive [11]	8-bit Decoder(245.5μm^2)	64	0.412
	16-bit Decoder(320.4μm^2)	32	
	8-bit PE(377.5 μm^2)	1024	
ANT [29]	Decoder(24.2μm^2)	64	0.388
	8-bit PE(377.5 μm^2)	1024	
GOBO [10]	8-bit PE(377.5 μm^2)	1024	0.450
	16-bit SPU (996.0 μm^2)	64	
AdaFloat [28]	8-bit PE(377.5 μm^2)	1024	0.387

attention-based models due to their complex encoding and decoding mechanisms. PISA stands out by fully exploiting inherent sparsity through precision-slicing and achieving significant computational speedup via the early-bird mechanism, without additional encoding and decoding overhead.

Energy. Figure 7 shows the normalized energy consumption across five networks, broken down into DRAM, global buffer (BUFFER) and processing core (CORE). PISA exhibits the lowest energy consumption for all transformers, consuming on average 66.7%, 44.5%, 24.0%, and 14.3% less energy compared to AdaFloat, GOBO, ANT, and Olive, respectively. This energy advantage stems from PISA avoiding complex encoder/decoder mechanisms and leveraging the higher-order sparsity and importance in LLMs to skip computations.

Area. Table II compares the area and number of PEs for PISA and other accelerators, all scaled to a 45-nm process, including decoder areas where applicable. PISA achieves the lowest area overhead by using compact 4-bit PEs without decoding mechanisms and implementing the early bird mechanism with a single comparator. In contrast, other accelerators like GOBO incur substantial area overhead due to the inclusion of decoders. This design allows PISA to accommodate more PEs within the same area budget, improving throughput.

C. Sensitivity Analysis

This section examines the impact of the early bird mechanism on PISA, focusing on how different thresholds and preset values affect performance and accuracy.

TABLE III: Perplexity results with Different Threshold for GPT2-XL. Preset values are layer-wise optimized.

Threshold	Origin	0	3	7	15	31	Layer-wise Optimization
Percentage of early birds(%)	-	87.6	87.9	88.6	90.4	92.8	91.3
PPL	16.94	18.04	20.12	25.66	89.27	766.82	18.15

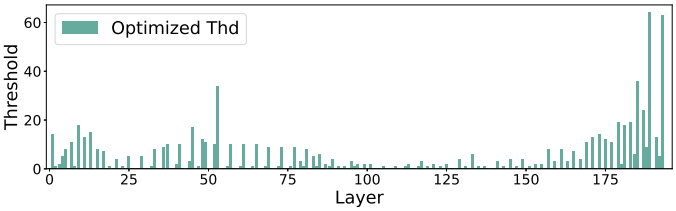


Fig. 8: Optimized layer-wise Threshold of GPT2-XL.

TABLE IV: Perplexity results with Different Strategies of Preset Value for GPT2-XL. Threshold are layer-wise optimized.

Strategy of preset value	Origin	All set to 0	Mean Value of each layer	Optimization after search
PPL	16.94	135.15	42.49	18.15

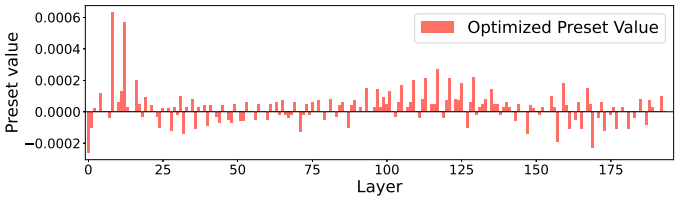


Fig. 9: Optimized layer-wise Preset Values of GPT2-XL.

Impact of Threshold. The early bird mechanism in PISA skips remaining computations if the product of two 4-bit higher-order parts is below a threshold, outputting a preset value instead. Choosing an appropriate threshold is crucial to balance accuracy and performance. TableIII shows the computation saving and perplexity results for fixed thresholds (0 and $2^n - 1$) on the GPT2-XL model. To optimize, we performed a layer-wise binary search for thresholds that minimize MSE on the validation set. Figure8 displays the optimal thresholds for each layer. This approach allows PISA to exploit computation redundancy in insensitive layers, achieving higher performance than fixed thresholds with negligible accuracy loss.

Impact of preset value. The early bird mechanism uses a preset value to preserve information when skipping computations. Choosing a suitable preset value is crucial for maintaining model accuracy. Table IV shows the evaluation of several strategies. Setting all preset values to 0 significantly degrades inference performance, similar to using a fixed threshold. We optimized preset values layer by layer, starting from the mean value of each layer's parameters and searching a small area around it. This approach successfully maintained low perplexity, as shown in Table IV. Figure9 shows the distribution of optimized preset values.

Our analysis reveals that layers with both threshold and preset values set to 0 are predominantly linear projections within the attention block. Consequently, for a simplified configuration, we can uniformly set the thresholds and preset values of all linear projection layers in the attention block to 0 while still maintaining strong performance.

VI. Conclusion

This paper introduces PISA, an precision-slice framework designed to enhance the efficiency of LLM inference. The key insight of PISA is to exploit the inherent sparsity and contribution within the computation of higher-order parts through the bit-slicing scheme. By integrating an early-bird mechanism, PISA selectively bypasses unnecessary computations for better performance. Evaluations show that PISA delivers significant performance gains with only a negligible accuracy loss.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[3] F. Liu, W. Zhao, Z. Wang, Y. Chen, Z. He, N. Jing, X. Liang, and L. Jiang, "Ebsp: evolving bit sparsity patterns for hardware-friendly inference of quantized deep neural networks," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 259–264.

[4] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh, "Gptq: Accurate post-training quantization for generative pre-trained transformers," *arXiv preprint arXiv:2210.17323*, 2022.

[5] S. Sharify, Z. Xu, X. Wang *et al.*, "Combining multiple post-training techniques to achieve most efficient quantized llms," *arXiv preprint arXiv:2405.07135*, 2024.

[6] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 38 087–38 099.

[7] F. Liu *et al.*, "Improving neural network efficiency via post-training quantization with adaptive floating-point," in *ICCV*, 2021.

[8] M. van Baalen, A. Kuzmin, M. Nagel, P. Couperus, C. Bastoul, E. Mahurin, T. Blankevoort, and P. Whatmough, "Gptvq: The blessing of dimensionality for llm quantization," *arXiv preprint arXiv:2402.15319*, 2024.

[9] E. Park, D. Kim, and S. Yoo, "Energy-efficient neural network accelerator based on outlier-aware low-precision computation," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018, pp. 688–698.

[10] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 811–824.

[11] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–15.

[12] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. H. Tan, Z. Li, D. Fan, X. Qian *et al.*, "Non-structured dnn weight pruning—is it beneficial in any platform?" *IEEE transactions on neural networks and learning systems*, vol. 33, no. 9, pp. 4930–4944, 2021.

[13] F. Liu, N. Yang, H. Li, Z. Wang, Z. Song, S. Pei, and L. Jiang, "Spark: Scalable and precision-aware acceleration of neural networks via efficient encoding," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 1029–1042.

[14] F. Liu, N. Yang, Z. Song, Z. Wang, H. Li, S. Huang, Z. Song, S. Pei, and L. Jiang, "Inspire: Accelerating deep neural networks via hardware-friendly index-pair encoding," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[15] X. Wei, Y. Zhang, X. Zhang, R. Gong, S. Zhang, Q. Zhang, F. Yu, and X. Liu, "Outlier suppression: Pushing the limit of low-bit transformer language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 17 402–17 414, 2022.

[16] Z. Yao, R. Yazdani Aminabadi, M. Zhang, X. Wu, C. Li, and Y. He, "Zeroquant: Efficient and affordable post-training quantization for large-scale transformers," *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 168–27 183, 2022.

[17] Y. Lin, H. Tang, S. Yang, Z. Zhang, G. Xiao, C. Gan, and S. Han, "Qserve: W4a8kv4 quantization and system co-design for efficient llm serving," *arXiv preprint arXiv:2405.04532*, 2024.

[18] H. Xia, Z. Zheng, X. Wu, S. Chen, Z. Yao, S. Youn, A. Bakhtiari, M. Wyatt, D. Zhuang, Z. Zhou *et al.*, "Fp6-llm: Efficiently serving large language models through fp6-centric algorithm-system co-design," *arXiv preprint arXiv:2401.14112*, 2024.

[19] N. Yang, F. Liu, Z. Wang, J. Zhao, and L. Jiang, "Searchq: Search-based fine-grained quantization for data-free model compression," *IEEE Transactions on Circuits and Systems for Artificial Intelligence*, 2024.

[20] G.-M. Tang, K. Takata, M. Tanaka, A. Fujimaki, K. Takagi, and N. Takagi, "4-bit bit-slice arithmetic logic unit for 32-bit rsfq microprocessors," *IEEE Transactions on Applied Superconductivity*, vol. 26, no. 1, pp. 1–6, 2015.

[21] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[22] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin *et al.*, "Opt: Open pre-trained transformer language models," *arXiv preprint arXiv:2205.01068*, 2022.

[23] A. Wang, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[24] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," *arXiv preprint arXiv:1609.07843*, 2016.

[25] S. D. Compiler, [Online]. Available: https://www.synopsys.com/support/training/rtlsynthesis/design-compiler-rtl-synthesis.html, 2019.

[26] D. Fujiki, S. Mahlke, and R. Das, "In-memory data parallel processor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 1–14, 2018.

[27] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "Cacti 7: New tools for interconnect exploration in innovative off-chip memories," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 14, no. 2, pp. 1–25, 2017.

[28] T. Tambe, E.-Y. Yang, Z. Wan, Y. Deng, V. J. Reddi, A. Rush, D. Brooks, and G.-Y. Wei, "Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020, pp. 1–6.

[29] C. Guo, C. Zhang, J. Leng, Z. Liu, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "Ant: Exploiting adaptive numerical data type for low-bit deep neural network quantization," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 1414–1433.

DeepPUFSCA: Deep learning for Physical Unclonable Function attack based on Side Channel Analysis support

Ngoc Phu Doan*, Tuan Dung Pham*, Zichi Zhang*, Viet Hung Tran*, Jack Miskelly,
Hans Vandierendonck, Anh Tuan Hoang, Máire O'Neill, Thai Son Mai
Queen's University Belfast, UK
Email: {ndoan01, h.vandierendonck, thaison.mai}@qub.ac.uk; m.oneill@ecit.qub.ac.uk

***Abstract*—Physical Unclonable Function (PUF) poses a vulnerability that it could be imitated by machine learning attacks and side channel attacks, which break its physical uniqueness and unpredictable characteristic. Hence, many works are concerned with enhancing PUF design by introducing more nonlinear modules inside to differentiate approximating PUF behavior from the attacker side. However, the safety of these PUFs are still an open area and need to be verified. In this paper, we propose DeepPUFSCA, which is a deep learning-based model that uniquely combines both challenge and side-channel information features during training to attack PUF. To gather the data, we conduct a design of an arbiter PUF on FPGA and measure its power consumption. Our intensive experiments on this dataset demonstrate that DeepPUFSCA outperforms other machine learning-based methods in terms of attacking accuracy, even the novel ensemble algorithms. Moreover, we also show that combined side channel information boosts the model performance compared to attacking with challenge-response only.**

***Index Terms*—AI and Machine Learning, FPGA Systems, Security & Privacy**

I. Introduction

In recent decades, Physical Unclonable Function (PUF) has been introduced as a hardware component that works as a lightweight and secured fingerprint. This module could be easily integrated into complex networks such as IoT and satellites for identification and authentication [1]. Depending on the number of possible challenge-response pairs (CRPs), there are two types of PUF: strong PUF and weak PUF [2]. In fact, strong PUF has a numerous number of CRPs; hence, it contains the physical uncloning characteristic. Many variations of strong PUFs have been studied and the most popular PUF is Arbiter PUF (APUF) and its variant: XOR PUF [3], feed-forward PUF (FF PUF) [4], controlled PUF [5], multiplexer-based PUF (MPUF) [6], and interpose PUF (iPUF) [7].

Although Arbiter-based PUF has many advantages in security, its vulnerability has been exploited by Machine Learning (ML) attacks and side channel analysis (SCA) with many reported successful attacks. Machine Learning-based attacks such as Evolutional Search (ES) [8], Logistic Regression (LR) [9] and Support Vector Machine (SVM) [10] are conducted by collecting a subset of CRPs from the PUF and exploits them to train the ML model. Meanwhile, a side channel attack is employed to extract leakage information of PUFs through power trace or time delay [11]–[13].

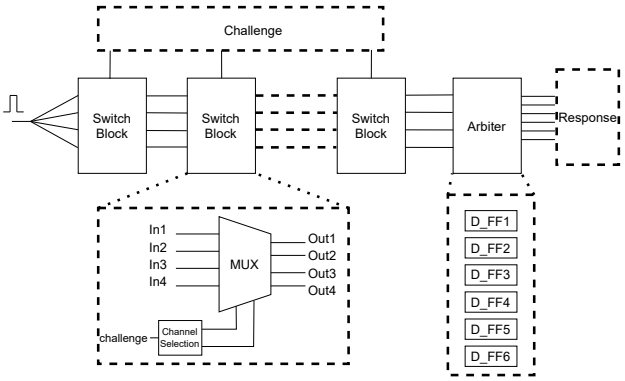


Fig. 1: 4x4 Arbiter PUF design

To enhance the safety of PUFs, there are recent works aiming to design ML-resistant architecture. They follow diverse approaches, e.g., exploiting encryption of the data traversing inside PUF before to hide information access from attackers [14] or introducing more randomness and non-linearity sources into a PUF design [3], [6], [7].

In response to techniques that defend PUFs against ML, the ML-based attack model is evolving to exploit PUFs. Since Classical ML attacks and SCA could not model strong PUFs with a very complicated structure [15], Deep learning (DL) methods have recently emerged as a promising alternative solution to attack PUFs [16], [17]. E.g., Yu et al. [17] employs CNN to attack an APUF. Similarly, a deep feedforward neural network [16] has been successfully used to attack both 64-bit and 128-bit APUF and some of its variants like XOR APUF, LSPUF, MPUF or IPUF.

Recently, Durova et al. [18] introduced a novel 4x4 APUF, a more extensive design of a vanilla APUF as shown in Figure 1. This design has more complexity compared to conventional APUF due to a double of pulse traversing lines. By exploiting six APUFs at the end, this design works as

* Equal contribution

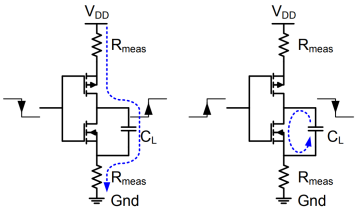


Fig. 2: CMOS gate during charge and discharge process [20]

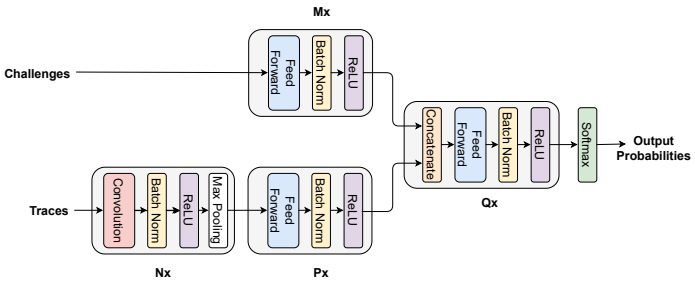


Fig. 3: DeepPUFSCA architecture

a parallel PUF, leading to a complicated relation between challenge and response. Therefore, 4x4 APUF is claimed to be more resistant to modeling attacks compared to other PUF [18], [19]. Hence, in our study, we adopt this 4x4 APUF design [18] as our main study subject.

Contributions. Our contributions are summarized as follows:

- Though the 4x4 APUF design is claimed to be more resistant against attack [18], [19], we propose a hybrid DL architecture to show that it is still vulnerable to DL attacks. To the best of our knowledge, we are the first to attack the 4x4 APUF.
- We present two theoretical mathematical models to formulate the relationship between challenge-response pairs and the power side channel in Section II. This motivates us to use side channel as an additional feature of our hybrid model (described below in Section III) thus enhancing attack accuracy (as shown in Section IV-A).
- We propose DeepPUFSCA, a DL-based model that, instead of using only challenge as an input, makes use of an additional power side channel to enhance the prediction results. Our approach is backed up by theoretical analysis on the correlation between side channel and PUFs output as explained above. Extensive experiments in Section IV-A show that DeepPUFSCA acquires much better performance than many common ML approaches in PUFs like Logistic Regression, SVM or recent advantage methods in ML community like CatBoost or LGBM.

II. Preliminaries

A. Physical Unclonable Function

At first, we present several mathematical definitions about our investigated physical unclonable function design.

Definition 1 (Physical Unclonable Functions): A PUF is defined as a mapping $\mathcal{M} : C \underset{\mathcal{N}}{\longrightarrow} R$ where $C = \{0,1\}^{d_c}$ is a challenge set, $R \subseteq \{0,1\}^{d_r}$ is a response set and $\mathcal{N}$ is randomness from uncontrollable manufacturing variation. We often refer to d_c, d_r as the security parameters of the PUF. The tuple (c_i, r_i) where $c_i \in C, r_i \in R$, are called challenge response pairs (CRPs).

Definition 2 (Strong PUFs): Strong PUFs are PUFs that have a large number of CRPs (c_i, r_i), $i = 1, 2, \ldots, N$.

One of the powerful characteristics of strong PUFs is that there is a tiny chance for an attacker to conduct an exhaustive attack due to the huge search space and the limitation of computational resources. In fact, the probability of success is approximating $1/N = 2^{-\lambda} \approx 0$ when $N \to +\infty$.

Definition 3 (Power consumption): During the activation process, PUF changes the status of logic components inside from active logic level to low logic level and vice versa, which consumes some electronic energy. We denote the power consumption as $t \in T$ where $T \subset \mathbb{R}^{d_t}$.

B. 4x4 Arbiter PUF

4x4 Arbiter PUF is a more extensive design of a vanilla arbiter PUF. Instead of exploiting 2x2 MUX as parallel paths for pulse traversal, it takes advantage of 4x4 MUX, which has double in number of input and output pins, to enhance design complexity, which could benefit attacking resistances. As shown in Figure 1, the 4x4 arbiter PUF consists of a sequence of switch blocks ending with an arbiter block. By triggering a pulse at the starting point, the pulse will propagate through four parallel lines. These lines are changed in order depending on the status of switch blocks, which is controlled by challenge. At the end of the design, we will obtain response bits from the arbiter block by comparing all couple combinations to examine which line is faster. This comparison strategy leads to achieving $C_4^2 = 6$ output bits. Besides, there are $c = 128$ bits in challenge, and it is divided into 24 separated blocks to control the switches.

III. Methodology

A. Problem formulation

Let $\mathcal{M} : C \times T \to R$ is an instance of 4x4 Arbiter PUF design where $C \subseteq \{0,1\}^{d_c}$, $T \subseteq \mathbb{R}^{d_t}$ and $R \subseteq \{0,1\}^{d_r}$ with the dimensions d_c, d_t, d_r. The training data, S including CRPs and traces in which $S = \{(c_1, t_1, r_1), \ldots, (c_n, t_n, r_n)\}$ where (c_i, r_i) are i.i.d. and drawn from a space $(C \times T)$ while the labels r_i satisfy $r_i = \mathcal{M}(c_i, t_i)$. We denote a hypothesis class $\mathcal{H}_\Theta = \{h_\theta, \theta \in \Theta\}$, and our target is to find the closest hypothesis to the ground truth $\mathcal{M}$ based on the algorithm $\mathcal{A}(S)$. To bound the solution space of class $\mathcal{H}_\Theta$, we introduce some key characteristics from our target design.

Proposition 1: The 4x4 APUF mathematical model $\mathcal{M}$ has a strong correlation in probability with challenges C.

Proof: Let $\mathbf{r} = (m_i)_{6\times 1}$ where $m_i \in \{0,1\}$ is a response of the 4x4 APUF. We denote $\odot$ is a cross-dimensional pairwise

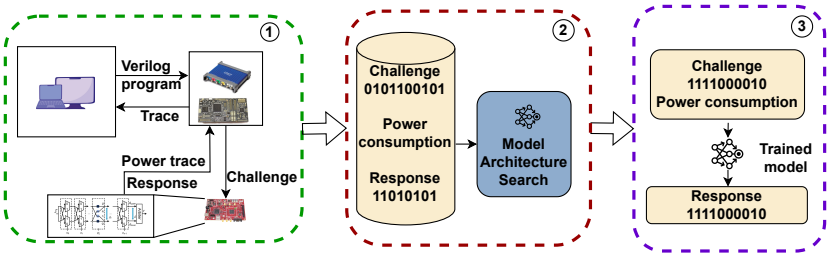


Fig. 4: Modeling scheme for 4x4 APUF

comparison such that $\odot\mathbf{a} = [(a_i <= a_j)_{i<j}]_{\frac{d_a(d_a-1)}{2}\times 1}$. We form a challenge-response related formula for the PUF [18]:

$$\mathbf{r} = \odot F(\prod_{n=1}^{N} S_n)\delta \tag{1}$$

In which, δ is a unit vector in $\mathbb{R}^5$, S_n is a probabilistic matrix denoting the uncertainty behavior of a switch block, F is a matrix presenting an error introduced by an arbiter flip-flop. Let's take an expectation on both sides. We have:

$$\mathbb{E}\mathbf{r} = \odot F(\prod_{n=1}^{N} \mathbb{E}S_n)\delta = \odot F(\prod_{n=1}^{N} W_n C_n)\delta \tag{2}$$

where C_n is a partition of challenge C, W_n is a weight matrix presenting the correlation between S_n and C_n in the design. ■

Proposition 2: In CMOS-based PUF, the model $\mathcal{M}$ has a strong relation with the energy consumption T. We assume that the model $\mathcal{M}$ is approximated in probability by deterministic $\mathcal{M}^{'}(C)$ following Proposition 1.

Proof: First, we determine a power model for 4x4 APUF by cumulating energy consumption over all elements inside the circuit. Let's take a look at Figure 2, The power consumption for the ith CMOS gate is written as follows:

$$P_i = C_L V_{DD}^2 Pr^{0\to 1} f \tag{3}$$

where f is the working frequency of the device, V_{DD} is the voltage of the power supply, and $Pr^{0\to 1}$ is the probability of changing the state of the component from 0 to 1. The power consumption of this device is written as:

$$T = \sum_{i=1}^{N} C_{Li} V_{DDi}^2 Pr_i^{0\to 1} f_i = WPr = W(Pr_{\text{det}} + Pr_{\text{noise}}) \tag{4}$$

where $Pr = \{Pr_1^{0\to 1}, Pr_1^{1\to 0}, \ldots, Pr_N^{0\to 1}, Pr_N^{1\to 0}\}$ is a state vector of elements in the design. Due to the introduction of noise during implementation, the behavior of components changes the state with the probability of Pr_{noise} compared to an ideal design with deterministic probability Pr_{det}. The deterministic state P_{det} is decided by challenge C, Pr_{noise} is decided in the behavior of a switch block. Let $Pr_{\text{det}} = \phi(C)$

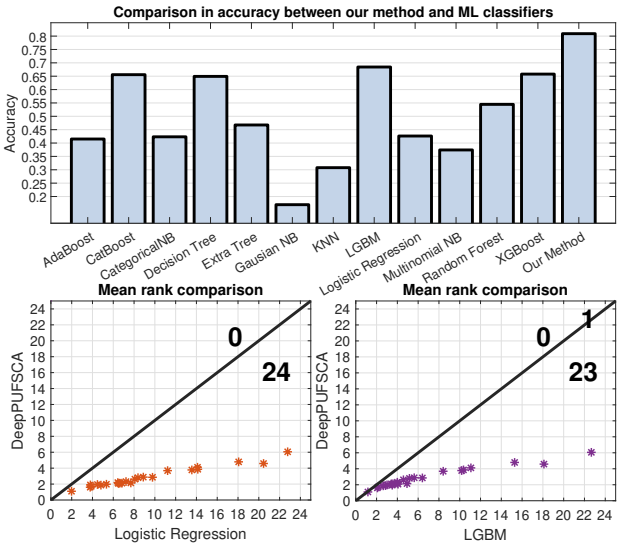


Fig. 5: Comparison of our model with other ML approaches in accuracy

from Equation 4, we have $Pr_{\text{noise}} = W^{-1}T - \phi(C)$. From Equation 1, we transform as follows:

$$\begin{aligned}\mathbf{r} &= \mathbb{E}\mathbf{r} + \mathcal{N}\mathbf{r} \\ &= \odot F(\prod_{n=1}^{N} W_n C_n)\delta + \odot F(\prod_{n=1}^{N} \mathcal{N}S_n)\delta \\ &= \odot F(\prod_{n=1}^{N} W_n C_n)\delta + \omega(P_{\text{noise}}) \\ &= \odot F(\prod_{n=1}^{N} W_n C_n)\delta + \omega(W^{-1}T - \phi(C))\end{aligned} \tag{5}$$

where, ω and ϕ are functions. $\mathcal{N}$ denotes a noise extractor. ■

B. Deep learning model for 4x4 Arbiter PUF

We introduce a hypothesis class $\mathcal{H}_\Theta$ to approximate Function 2. The architecture is presented in Figure 3 where it takes challenge C and trace T as inputs and outputs the probability value for each possible class of response. In this case, we take advantage of the MLP layer to extract features for challenge and the CNN layer to extract features for trace. Then, two

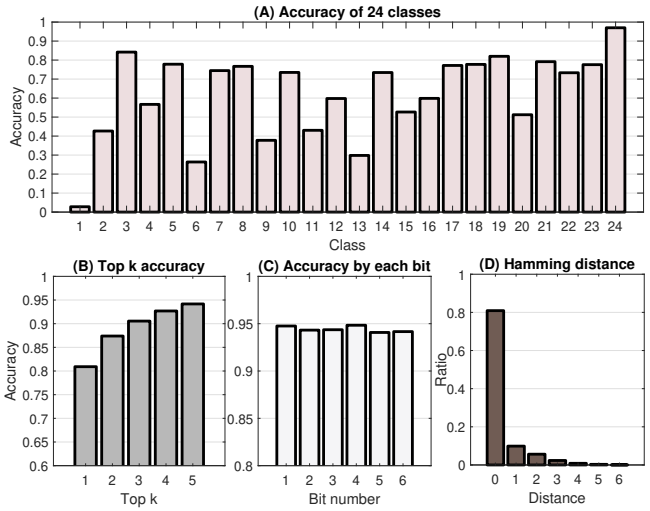


Fig. 6: Performance of our model in different views

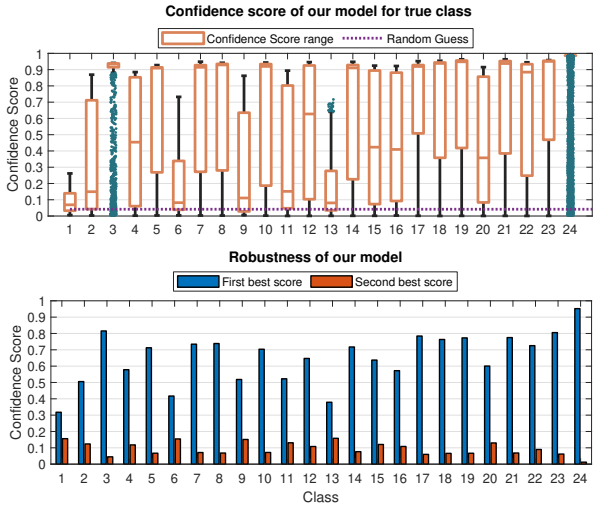


Fig. 7: Robustness of our method

features are combined before pushing to an MLP classifier model. Formally, let $f_{\theta_1}, g_{\theta_2}$ and f_{θ_3} are MLP extractor, CNN extractor, and MLP classifiers, respectively. Our target is finding a parameter $\Theta = (\theta_1, \theta_2, \theta_3)$ that minimises an error through the population $\mathcal{D}$:

$$\Theta = \arg\min_{\Theta} \int_{(c,r,t)\sim\mathcal{D}} \mathcal{E}(f_{\theta_3}(f_{\theta_1}(c), g_{\theta_2}(t)), r) \quad (6)$$

where $\mathcal{E}$ is an error function that denotes how far from the model prediction value to the ground truth value r.

C. Modeling scheme for 4x4 Arbiter PUF

The overall process of PUF attack by side-channel information analysis with three steps is shown in Figure 4. At first, a 4x4 Arbiter PUF is programmed by VHDL, and then it is embedded into the FPGA target board. To collect power consumptions of this board, the synchronous system among a computer, PICO scope, and the board is conducted, which can be able to run automatically. Then, the data collection, $\{(c_i, r_i, t_i)_{i=1,\ldots,n}\}$ including challenges, responses, and traces, is exploited for constructing an ML prediction model. After that, we make use of model architecture search to optimise our model parameters to minimise the objective function 6 and evaluate this model on the test dataset.

IV. Experiment

A. Experimental setup

Data collection. We setup a system of ChipWhisperer Lite with CW305 Artix FPGA Target, PICO scope A3405, and PC as in Figure 8. We implement 4x4 APUF on FPGA. Our scope captures the power consumption of the FPGA when the circuit triggers to an active mode. This PICO scope can measure a signal with a sampling rate of 1 GS/s and ADC specs of 8–12 bits resolution. With these configurations, we obtain a set of power traces with a length of 522 values.

Training and testing dataset. We collect 500000 samples with challenges, responses, and power consumption signals. Then, we split this data into two sets as a training dataset and a testing dataset with a ratio of 4:1. Due to noisy variance in power traces, we apply *Minmax Scaler* to normalise these power consumptions to a range of 0-1 to enhance our data consistency.

Model architecture and hyper-parameters. We conduct a model architecture as shown in Figure 3. In the challenge branch, we implement a MLP extractor with $M = 15$ layers. In the trace branch, we conduct a CNN model with $N = 5$ layers and a kernel size of 11, followed by a series of MLP layers with $P = 15$ layers. Finally, both features from the two above branches are concatenated to a vector and forwarded to a MLP classifier with $Q = 15$ layers. On top of that, we exploit cross-entropy loss, SGD optimiser with a learning rate, $lr = 2e^{-5}$ and an early stopping strategy to optimise our model. All MLP extractors are using 64 neurons for hidden layers.

Evaluation metrics. To evaluate the model's performance, we use many metrics, such as the following:

- Accuracy: Accuracy is determined by the ratio of corrected predictions to the total number of predictions.
- Top-k accuracy: Top k accuracy is calculated by the ratio of times where the label is among the top k labels predicted to the total number of predictions.
- Mean rank: We define the rank of a sample as the ranking of its label in confidence scores predicted by a model. The mean rank of a class is calculated by averaging the rank of all samples inside this class.

B. Side channel hybrid attack performance

Performance comparisons to advantage ML methods. To evaluate the performance of DeepPUFSCA, we compare our model with many Machine Learning algorithms, including classical ML models (Categorical Naive Bayes, Decision Tree, Gaussian Naive Bayes, KNN, Logistic regression) and many

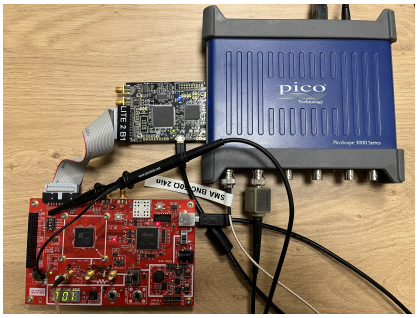


Fig. 8: Power measurement system setup

recent advanced ML algorithms (AdaBoost [21], CatBoost [22], Extra Tree [23], LGBM [24], Random forest [25] and XGBoost [26]). Figure 5 shows that our method achieves 81% accuracy, which significantly outperforms all others. Compared to Logistic Regression which is a popular approach to model arbiter PUF, our approach significantly increases 35%. Even with the advanced ML algorithms, our model is still better with 11% higher than LGBM which is at 70% in terms of accuracy.

We also employ a mean rank comparison of our model with LR and LGBM by scatter chart on Figure 5. The further point below the diagonal line means that our model is much better than LR and LGBM. Specifically, our method dominates marginally all 24 classes compared to other methods.

Performance for response classes and bits. In Figure 6, we examine the classification performance of DeepPUFSCA from different perspectives. The result of each class shows that most classes achieve a higher margin accuracy than the random guess with 4% predictive accuracy. In terms of performance on each bit (6 bit responses), DeepPUFSCA achieves nearly perfect accuracy at 95% for all six bits. Figure 6 also presents top-k accuracy for our model prediction over 24 classes. In fact, the accuracy increases dramatically when k is increasing with 87% when $k = 2$ and 90% when $k = 3$. This result can be used in an exhausted attack scheme where we query an opinion of our models multiple times for one challenge until the corrected response is predicted. This strategy will significantly reduce the search space for challenge response pairs during brute force attacks from $O(2^{24n})$ to $O(2^{kn})$ where n is the length of challenge in binary presentation. Besides, we also analyse the hamming distance between DeepPUFSCA prediction and a true label. The distance 0 refers to the fact that the ratio of prediction samples is correct. The distance statistic confirms that if our model infers wrongly, the number of incorrected bits in the predicted response is one or two bits with 10.58% and 7.55% over total 19% mistaken cases.

How does the side channel help to improve the performance? We exploit the best configuration architecture of DeepPUFSCA and remove the trace branch to see the contribution of power traces during training and testing. The experimental result demonstrates that our method enhances 2.5% if we utilise side channels for the learning algorithm.

TABLE I: Performance training with trace vs w/o trace

Accuracy (%)	
w/o trace	w trace
78.67	**81.11**

Therefore, this could imply that the side channel signal contains information leakage of PUF during its activation.

Robustness of our model. We analyse the robustness of our model based on two aspects of the model's confidence score in Figure 7. The first aspect is the distribution of confidence scores for each class. As we observe, the confidence scores are better than Random Guessing for almost all classes, and most of them vary from 0.1 to 0.9. Since investigating only the confidence score distribution of the correct class might not show a complete picture of robustness, we also investigate the relationship of the correct class and other classes. To show this relationship, we plot the first-best confidence score and second-best confidence score for each class prediction at the bottom of Figure 7. If the gap between the first-best score and the second-best score is large, it means that our model is less likely to select the wrong class and thus more robust. Overall, a large gap between the best score and the second-best score is present in all classes. The gap for all classes is larger than 0.2 except for classes 1, 6, and 13. Of all three classes that have their confidence gap lower than 0.2, Class 1 has the lowest gap at 0.1. The above results showed that our model is highly robust despite some variance in the confidence score.

C. Ablation study

Number of training samples. DeepPUFSCA achieves the attacking performance of 40% accuracy at 100K samples; the model is at peak 81% accuracy with 500K samples. Hence, we choose 500K data to optimise our model parameters.

Model's depth. We train DeepPUFSCA with different numbers of layers (M, N, P, Q). To reduce the search space, we assume that $M = P = Q, N = 5$ and the number of layers is $M+P+Q$. Figure 9 justifies that the choice $M+P+Q = 45$, which is corresponding to $M = P = Q = 15$, is an optimal hyperparameter for our model's depth.

CNN's kernel size. We train our model with different sizes of CNN kernels. The kernel 11 returns the highest accuracy, so we exploit this configuration for DeepPUFSCA. Regarding the choice of CNN's kernel size, the accuracy of our method varies in a small range from 0.775 to 0.81, which showed the stability of our approach.

Number of nodes in hidden layers. Depending on the number of nodes in MLP hidden layers, we optimise our model in four cases with corresponding nodes as Tiny (32), Small (64), Medium (256), and Large (512). The Small architecture is the best compared to other configurations.

Model generalisation. Model generalisation [27] is defined as the ability to predict well on data that has not been seen during training. To evaluate generalization, we can use the gap between training loss and validation loss. Therefore,

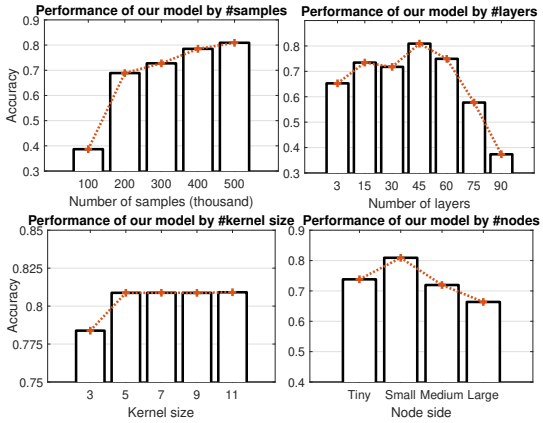


Fig. 9: Effect of different hyper-parameters to our model

we present a loss trending during training DeepPUFSCA on Figure 10. From epoch 0 to 150, the gap between training loss and validation loss is closed to 0. After the epoch 200, there is a small gap between 2 losses. We observe that at epoch 350, the validation loss is convergent around 0.65 while the training loss is still decreasing. Hence, we stop the training procedure at the epoch 350.

V. Related Works

Side channel attack (SCA) for PUF. Side channel attack is a type of non-invasive attack based on the leakage of information due to imperfect implementation. Based on this harmful, many papers [11]–[13] evaluate weakness points of PUF through side channel such as power side channel, timing, response repeatability, etc. Delvaux et al. [11], [12] exploits the response stability to learn APUF mathematical model by assuming that the response acts as normal distribution. Impressively, [13] can accurately predict the number of APUF returning to one before XOR together based on frequent patterns of power and timing delay side channel. Although SCA could reveal some useful information, it only has been used for APUF, XOR PUF. This approach might be ineffective to attack a complicated PUF since the information in side channel is more difficult to detect.

ML attacking PUF. ML has been demonstrated to pose a huge threat for PUF due to high-accuracy attacks. The vulnerability of these PUFs comes from the nature of PUF design by constructing the combined circuit of linear blocks and nonlinear blocks cascadingly [28]. Generally, two successful ML attacks have been achieved so far: non-deterministic ML and deterministic ML.

In 2010, Rührmair [8] conducted Evolutionary Search to attack FFPUF with nearly perfect accuracy. Inspiring by biology evolution, the ES algorithm is a non-deterministic ML, which iteratively generates new model parameters until the model is closed to the target model. Later, the authors [29], [30] utilise CMA-ES, first cloning the real-world XOR Arbiter PUF-based RFID tags. It is a disadvantage that these algorithms may be out of track to the target during updating and lead to a bad solution due to randomness in weight generation procedure.

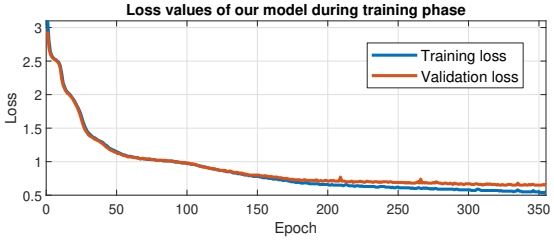


Fig. 10: Loss function over epochs

Many deterministic ML algorithms are proposed such as Logistic Regression [7]–[9], [31], SVM [10], [28], Gradient Boosting [28], etc. perform well on iPUF which is a protected design of PUF. Recently, ANN is introduced as a powerful method that can successfully attack on many PUFs as APUF, XOR PUF and iPUF [16], [32]–[35] with high accuracy.

Hybrid ML-SCA for PUF. Another approach to modeling PUF is hybrid ML-SCA attack. In this method, side channel information is analysed to reveal relevant features related to response, and this data will be fed into ML algorithms to improve attacking accuracy. [13] leverages power side channel to extract the cumulative number of APUFs that respond to one before doing XOR, and then integrate it to the ML objective function. This method achieves a very high accuracy when attacking 16-XOR PUF. Recently, [15] proposed a method that combines CRPs with power side channels to make challenge-synthesis-feature pairs. In this paper, side channel power will be encoded to a binary form and then combined with response to create a multi-class labels. They achieve higher accuracy than attacking with a side-channel model or mathematical model on popular PUFs. In contrast to this approach, we extract features of challenge and trace first, and combine them together to enrich information to predict response.

In our study, we exploit the 4x4 APUF as a victim for the first time. This design is demonstrated as a complicated circuit with higher responses bit than many popular PUFs. To attack this design, we propose a unique way to hybrid modeling 4x4 APUF by exploiting DL with two feature extractors: a challenge extractor and a trace extractor, that shows our attacking improvement.

VI. Conclusion

In this paper, we introduce DeepPUFSCA, a new ML-based model to attack PUF using both CRPs and power side channels. Our method addresses key challenges: (i) We theoretically analyse the relationship between CRPs and power traces; (ii) Our DeepPUFSCA successfully attacks 4x4 APUF with high accuracy at 81%; (iii) We propose an approach to incorporate the power trace into a novel model to enhance attacking performance by 2.5%. In future works, we target investigating other types of side channel trace, such as electromagnetic. Moreover, since the attacking performance on 4x4 APUF is lower than other common 2x2 APUFs as presented in Section IV-A, we aims to explore the relationship between PUF attack resistance and $n \times n$ APUF with a bigger n.

VII. Acknowledgement

We wish to thank the anonymous reviewers for their valuable comments that helped to improve our work. Ngoc Phu Doan and Tuan Dung Pham are supported by the Cyber AI Hub Doctoral Training Programme at CSIT, Queen's University Belfast, funded by the UK Government through the New Deal for Northern Ireland. The funding is delivered on behalf of the Northern Ireland Office and the Department for Science, Innovation, and Technology by Innovate UK. This research is also part-funded by the European Union (Horizon Europe 2021-2027 Framework Programme Grant Agreement number 10107245), Engineering and Physical Sciences Research Council under grant number EP/X029174/1, and Department of Economy.

References

[1] H. Ning, F. Farha, A. Ullah, and L. Mao, "Physical unclonable function: architectures, applications and challenges for dependable security," *IET Circuits Devices Syst.*, vol. 14, no. 4, pp. 407–424, 2020.

[2] T. McGrath, I. E. Bagci, Z. M. Wang, U. Roedig, and R. J. Young, "A PUF taxonomy," *Applied physics reviews*, vol. 6, no. 1, 2019.

[3] G. E. Suh and S. Devadas, "Physical unclonable functions for device authentication and secret key generation," in *DAC*, 2007, pp. 9–14.

[4] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *IEEE Symposium on VLSI Circuits*, 2004, pp. 176–179.

[5] Y. Gao, H. Ma, S. F. Al-Sarawi, D. Abbott, and D. C. Ranasinghe, "PUF-FSM: A controlled strong PUF," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 37, no. 5, pp. 1104–1108, 2018.

[6] D. P. Sahoo, D. Mukhopadhyay, R. S. Chakraborty, and P. H. Nguyen, "A multiplexer-based arbiter PUF composition with enhanced reliability and security," *IEEE Trans. Computers*, vol. 67, no. 3, pp. 403–417, 2018.

[7] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. van Dijk, "The interpose PUF: secure PUF design against state-of-the-art machine learning attacks," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2019, no. 4, pp. 243–290, 2019.

[8] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *CCS*, 2010, pp. 237–249.

[9] N. Wisiol, C. Mühl, N. Pirnay, P. H. Nguyen, M. Margraf, J. Seifert, M. van Dijk, and U. Rührmair, "Splitting the interpose PUF: A novel modeling attack strategy," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2020, no. 3, pp. 97–120, 2020.

[10] G. Hospodar, R. Maes, and I. Verbauwhede, "Machine learning attacks on 65nm Arbiter PUFs: Accurate modeling poses strict bounds on usability," in *WIFS*. IEEE, 2012, pp. 37–42.

[11] J. Delvaux and I. Verbauwhede, "Side channel modeling attacks on 65nm arbiter PUFs exploiting CMOS device noise," in *HOST*, 2013, pp. 137–142.

[12] ——, "Fault injection modeling attacks on 65nm arbiter and RO sum PUFs via environmental changes," *IACR Cryptol. ePrint Arch.*, p. 619, 2013.

[13] U. Rührmair, X. Xu, J. Sölter, A. Mahmoud, M. Majzoobi, F. Koushanfar, and W. P. Burleson, "Efficient power and timing side channels for physical unclonable functions," in *CHES*, vol. 8731, 2014, pp. 476–492.

[14] Q. Ma, C. Gu, N. Hanley, C. Wang, W. Liu, and M. O'Neill, "A machine learning attack resistant multi-PUF design on FPGA," in *ASP-DAC*, Y. Shin, Ed., 2018, pp. 97–104.

[15] W. Liu, R. Wang, X. Qi, L. Jiang, and J. Jing, "Multiclass classification-based side-channel hybrid attacks on strong PUFs," *IEEE Trans. Inf. Forensics Secur.*, vol. 17, pp. 924–937, 2022.

[16] P. Santikellur, A. Bhattacharyay, and R. S. Chakraborty, "Deep learning based model building attacks on arbiter PUF compositions," *IACR Cryptol. ePrint Arch.*, p. 566, 2019.

[17] Y. Yu, M. Moraitis, and E. Dubrova, "Why deep learning makes it difficult to keep secrets in FPGAs," in *DYNAMICS*, 2020, pp. 1–9.

[18] E. Dubrova, "A reconfigurable arbiter PUF with 4 x 4 switch blocks," in *ISMVL*, 2018, pp. 31–37.

[19] C. Aknesil and E. Dubrova, "An FPGA implementation of 4× 4 arbiter PUF," in *ISMVL*. IEEE, 2021, pp. 160–165.

[20] F.-X. Standaert, "Introduction to side-channel attacks," *Secure integrated circuits and systems*, pp. 27–42, 2010.

[21] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, 1997.

[22] L. O. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," in *NeurIPS*, 2018, pp. 6639–6649.

[23] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Machine learning*, vol. 63, pp. 3–42, 2006.

[24] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *NeurIPS*, 2017, pp. 3146–3154.

[25] L. Breiman, "Random forests," *Machine learning*, vol. 45, pp. 5–32, 2001.

[26] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *SIGKDD*, 2016, pp. 785–794.

[27] G. N. Karystinos and D. A. Pados, "On overfitting, generalization, and randomly expanded training sets," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1050–1057, 2000.

[28] A. Vijayakumar, V. C. Patil, C. B. Prado, and S. Kundu, "Machine learning resistant strong PUF: possible or a pipe dream?" in *HOST*, 2016, pp. 19–24.

[29] G. T. Becker, "The gap between promise and reality: On the insecurity of XOR arbiter pufs," in *CHES*, ser. Lecture Notes in Computer Science, vol. 9293, 2015, pp. 535–555.

[30] ——, "On the pitfalls of using Arbiter-PUFs as building blocks," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, vol. 34, no. 8, pp. 1295–1307, 2015.

[31] U. Rührmair, J. Sölter, F. Sehnke, X. Xu, A. Mahmoud, V. Stoyanova, G. Dror, J. Schmidhuber, W. Burleson, and S. Devadas, "Puf modeling attacks on simulated and silicon data," *IEEE Trans. Inf. Forensics Secur.*, vol. 8, no. 11, pp. 1876–1891, 2013.

[32] J.-Q. Huang, M. Zhu, B. Liu, and W. Ge, "Deep learning modeling attack analysis for multiple FPGA-based apuf protection structures," in *ICSICT*, 2018, pp. 1–3.

[33] M. Khalafalla and C. H. Gebotys, "PUFs deep attacks: Enhanced modeling attacks using deep learning techniques to break the security of double arbiter PUFs," in *DATE*, 2019, pp. 204–209.

[34] W. Ge, J. Q. Huang, B. Liu, M. Zhu, and Y. Cao, "A deep learning modeling attack method for MISR-APUF protection structures," in *APCCAS*, 2018, pp. 398–402.

[35] M. Khalafalla, M. A. Elmohr, and C. H. Gebotys, "Going deep: Using deep learning techniques with simplified mathematical models against XOR BR and TBR PUFs (attacks and countermeasures)," in *HOST*, 2020, pp. 80–90.

Constraint Graph-based PCB Legalization Considering Dense, Heterogeneous, Irregular-Shaped, and Any-oriented Components

Chiao-Yu Ou[1], Yan-Jen Chen[1], and Yao-Wen Chang[1,2]
[1]Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106319, Taiwan
[2]Department of Electrical Engineering, National Taiwan University, Taipei 106319, Taiwan
cyou@eda.ee.ntu.edu.tw; yjchen@eda.ee.ntu.edu.tw; ywchang@ntu.edu.tw

***Abstract*—In modern printed circuit board (PCB) designs, the increasing complexity poses more challenges for automatic placement. Existing PCB placement methods cannot handle complex constraints with heterogeneous, irregular-shaped, and any-oriented components for double-sided PCB designs well. This paper proposes the first constraint graph-based legalization approach for these constraints. We use a slicing technique to model a component more accurately with a set of rectangles instead of resorting to the naive bounding box approximation. Unlike the commonly used linear programming method for macro placement in integrated circuit (IC) designs, we employ a mixed integer linear programming (MILP) formulation to effectively expand the solution space for heterogeneous, irregular-shaped, and any-oriented components, particularly with high-density designs. Experimental results demonstrate the effectiveness and the robustness of our work.**

I. Introduction

In modern printed circuit board (PCB) design, miniaturization has become a significant trend due to the rising demand for portable and wearable electronic devices. As a result, creating small and compact products has become a necessity. These high-density layouts with complex design rules pose significant challenges for PCB placement. Traditionally, many companies managed PCB placement manually. However, as design complexities continue to grow, the need for an efficient electronic design automation (EDA) tool for modern PCB placement has become increasingly urgent.

Existing EDA tools developed for integrated circuit (IC) are not well-suited for PCB layouts due to significant differences in their placement requirements. These differences can be categorized into four main challenges: (1) irregular components and boards, (2) numerous and complex constraints, (3) high placement flexibility, and (4) the presence of through-hole (TH) pads on double-sided PCBs.

First, modern PCBs may involve irregular board shapes tailored for specific applications, unlike IC designs with regular components that enhance symmetry and consistency. Components such as inductors, transformers, and sensors are also irregularly shaped, making the placement process more complicated. These irregularities increase design complexity and make algorithm designs challenging. Second, numerous constraints among heterogeneous components and pads must be satisfied [1] in PCB designs due to issues such as signal and power integrity requirements and manufacturing process limitations. For instance, the complex design rules include *component-to-component* and *pad-to-pad* clearance, as shown in Figure 1(a). The clearance for P0 to P2 and P3 are different. Third, unlike IC layouts where components are restricted to rotations of 0°, 90°, 180°, or 270°, PCB components can be rotated to arbitrary angles to optimize space utilization and minimize wirelength. This flexibility requires advanced legalization tools to support placement. As depicted in Figure 1(b), the components around the central IC are rotated at various angles to minimize wirelength. Further, most PCB designs allow components to be placed on both sides of the board. Unlike surface mount device (SMD) pads, which only affect one surface, TH pads penetrate the board, creating obstacles on the opposite side. As shown in Figure 1(c), these obstacles must be considered when placing back-side components.

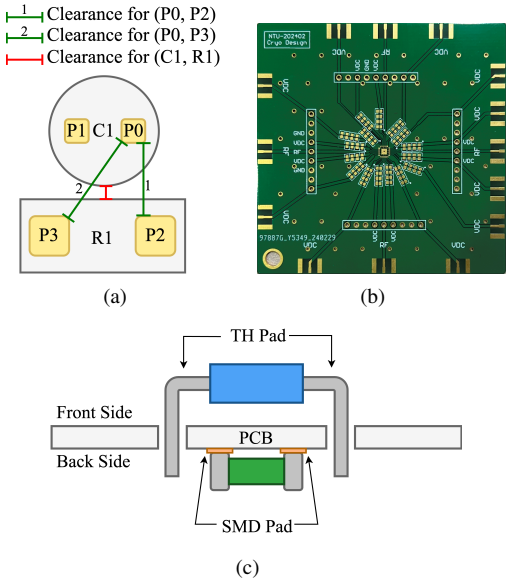


Fig. 1. Properties of PCBs. (a) An example of clearance constraints for components and pads. Clearances for (P0, P2) and (P0, P3) may be different. (b) An image of a PCB with components placed at arbitrary rotation angles. (c) Illustration of through-hole (TH) pads and surface mount device (SMD) pads on a double-sided PCB. These properties complicate the placement process.

A. Previous Work

For handling non-overlapping components, the algorithms can be classified into two types: (1) rectangular-based legalization, commonly used in mixed-size circuits for IC designs and (2) irregular shape-based methods, often applied in nesting problems for industries such as metal manufacturing, textiles, and logistics.

For rectangular-based methods, traditional macro placement and legalization methods, such as sequence-pair [2], MP-tree [3], [4], constraint-graph (CG) [5], transitive closure graph (TCG) [6], [7], and corner-stitching [8], [9], are widely adopted in modern IC placement frameworks. Similar to the traditional macro placement methods in IC designs, some PCB placement works [10], [11] applied B*-tree-based methods [12]. Wang *et al.* [10] considered the irregular boards by utilizing irregular outline division and obstacle block generation. However, these traditional macro placement-like methods often underperform in wirelength and density optimization when compared to analytical placement. In other PCB placement works [1], [13], components were also treated as rectangles during legalization. Tsou *et al.* [1] proposed a window-search-based legalization for each prioritized component after the force-directed global placement. NS-place [13] performed an mixed integer linear programming (MILP)-based legalization to minimize wirelength by formulating relative positioning constraints from global placement. In the legalization algorithms of these two works, the objective of minimizing displacement is hardly mentioned. Additionally, treating irregular components using their bounding box may lead to a significant waste of space, especially in high-density cases.

For irregular shape-based methods, the nesting problem focuses on fitting irregular or regular 2D shapes into rectangular spaces [14], [15], aiming to maximize space utilization and minimize material waste. A common data structure in these problems is no-fit polygon (NFP), which describes the permissible movement of one shape relative to another without any overlap. However, since NFP only avoids overlaps between two components, its computational cost for minimizing displacement increases exponentially as the number of components grows. Besides, unlike the nesting problem, packing techniques are not well-suited for the legalization problem to restrict feasible solutions, as packing techniques cannot ensure displacement minimization.

In conclusion, the rectangular-based legalization methods do not consider irregular properties, and the irregular shape-based methods do not fit the legalization objectives. Furthermore, clearance constraints were not considered in either of these previous works. As a result, an effective automated legalization tool is needed in modern PCB placement to alleviate the constraints imposed on global placement and maximize space utilization to meet design requirements. Without such a legalization tool,

This work was partially supported by AnaGlobe, ASUS, Delta Electronics, Google, Maxeda Technology, TSMC, NSTC of Taiwan under Grant NSTC 110-2221-E-002-177-MY3, NSTC 113-2218-E-002-041,NSTC 113-2223-E-002-007, and NSTC 113-2640-E-002-001.

PCB placement often relies on manual adjustments [10], which is time-consuming and inefficient.

B. Contributions

Our work is dedicated to meeting the extensive and complex constraint requirements of modern PCB designs. We mainly focus on the legalization stage after global placement, ensuring a legal solution that satisfies all constraints and preserves the desired solution from global placement by minimizing the total displacement. The main contributions of this paper are summarized as follows.

- To the best of our knowledge, this paper proposes the first legalization work on irregular-shaped components and boards by utilizing consecutive rectangle approximation.
- By introducing our modified transitive closure graph (mTCG), this work supports single- and double-sided PCBs while considering the impacts of TH pads and considers complex clearance constraints for the heterogeneous components in PCB designs.
- We apply a constraint graph-based legalization method by approximating each irregular component into consecutive rectangles and proposing a new constraint formulation using MILP.
- Experimental results show the effectiveness and robustness of our formulation. Compared with the rectangle-based legalization method, our proposed method significantly improves the success rate of legalization for high-density PCBs.

The remainder of this paper is organized as follows. Section II formulates the irregular-shaped components and board legalization problem. Section III presents our algorithm. Section IV shows the experimental results. Section V concludes this paper.

TABLE I
NOTATIONS USED IN THIS PAPER.

Notation	Description
C, P, F	The set of components, pads, and fixed components, respectively
c_i	The i-th component in C
$b_{i,p}$	The p-th block in c_i
$p_{i,k}$	The k-th pad in c_i
$b_{i,k,p}$	The p-th block in $p_{i,k}$
(x_i, y_i)	The center coordinates of c_i
w_i, h_i	The width and height of c_i
G_h, G_v	The horizontal and vertical graphs in mTCG
$v_{hi,p}, v_{vi,p}$	Two respective nodes of the corresponding $b_{i,p}$ on G_h and G_v
$e_{ip,jq}$	The edge from $v_{hi,p}$ to $v_{hj,q}$
$w(e_{ip,jq})$	Weight of $e_{ip,jq}$ on mTCG

II. PRELIMINARIES

For PCB placement, a design flow with global placement followed by legalization, similar to macro placement, is commonly adopted [1], [13]. Global placement focuses on key objective optimization, allowing component overlaps to explore a broader solution space. Then, legalization is used to remove component overlaps. This section first gives irregular PCB legalization model and then formulates the legalization problem.

A. Irregular PCB Legalization Model

Table I summarizes the notations used throughout this work. In our PCB legalization problem, a double-sided placement consists of a set $C = C_f \cup C_b$, representing m_f front-side components and m_b back-side components, where $C_f = \{c_i^f \mid i = 1, 2, \ldots, m_f\}$ and $C_b = \{c_i^b \mid i = 1, 2, \ldots, m_b\}$, and a set $P = P_f \cup P_b$, representing front-side pads and back-side pads, where $P_f = \bigcup_{c_i \in C_f} \bigcup_{p_{i,k} \in c_i} p_{i,k}$ and $P_b = \bigcup_{c_i \in C_b} \bigcup_{p_{i,k} \in c_i} p_{i,k}$.

Each component $c_i \in C$ is composed of a set of blocks $\{b_{i,p} \mid b_{i,p} \in c_i\}$ and a set of pads $\{p_{i,k} \mid p_{i,k} \in c_i\}$. Each pad $p_{i,k} \in P$ is further composed of a set of blocks $\{b_{i,k,p} \mid b_{i,k,p} \in p_{i,k}\}$. We refer to these blocks $b_{i,k,p}$, derived from pads, as ***pblocks***, to distinguish them from blocks originating directly from components. The center coordinates, width, and height of the bounding box for each component c_i are denoted as (x_i, y_i), w_i, and h_i, respectively. Components and their pads may have irregular shapes and arbitrary orientations. Additionally, a set of fixed components $F \subseteq C$ may be included, and a set of TH pads $P_{th} \subseteq P_f$ is allowed in double-sided placements. For single-sided PCBs, side information is omitted. The board's width and height are represented as W and H, respectively, and the board itself can be irregular.

This work considers the following constraints:

- **Board boundaries constraints**. All movable components must remain within the board boundaries.
- **Non-overlapping constraints**. Components and pads on the same side must not overlap. TH pads must be treated as occupying both sides of the board.
- ***Component-to-component*** **clearance**. A minimum spacing requirement between any two components on the same side.
- ***Pad-to-pad*** **clearance**. A minimum spacing requirement between any two pads on the same side, while TH pads are treated as occupying both sides.

The required clearances may vary depending on the combination of two components or pads.

B. Problem Formulation

Our PCB legalization problem is formally defined as follows:

- **The Irregular PCB legalization Problem**: Given an initial PCB placement with two sets, C and P, our objective is to find legalized positions for components while minimizing their displacement from the original positions and satisfying the abovementioned constraints.

By minimizing the component displacement, we can maintain the desired wirelength and routability since the previous stage, global placement, has been optimized for it.

III. PROPOSED METHODOLOGY

We propose a CG-based algorithm for legalizing irregular components, which constructs an mTCG to satisfy the given constraints and optimize component positions. The algorithm flow is illustrated in Figure 2.

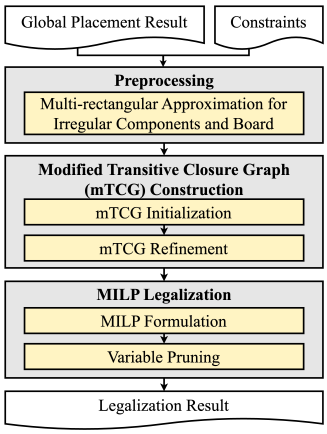


Fig. 2. Proposed irregular PCB legalization flow.

The algorithm consists of three main stages. The first *Preprocessing* stage addresses irregular components and boards, preparing them for the subsequent rectangle-based processing. The second *Modified Transitive Closure Graph (mTCG) construction* stage constructs mTCG to represent the geometric relations between components and pads. The final *MILP Legalization* stage employs the MILP formulation to legalize components.

A. Preprocessing

At this stage, we employ multi-rectangular approximation to handle irregular-shaped components and boards to enable the subsequent rectangle-based legalization. Specifically, we approximate a complex shape with multiple rectangles, providing a more accurate representation of PCB designs than the bounding box approximation. In our PCB designs, three key targets require preprocessing: irregular-shaped components and pads, highly non-convex components, and irregular-shaped boards.

First, for irregular-shaped components and pads, we start by approximating their shapes using convex hulls. If any pad extends beyond the outline of its corresponding component, we take the convex hull of both the component and the corresponding pads. An example of a rotated component is shown in the left part of Figure 3(a). Next, each convex hull is vertically sliced into multiple blocks, with a spacing defined by W_s. For each component or pad, the width of the rightmost block is adjusted to the modulus of w_i with respect to W_s. An example of this slicing process is shown in the middle part of Figure 3(a). Additionally, adjacent blocks with identical y-coordinates and height are merged into a single block to reduce the problem size. Second, for highly non-convex

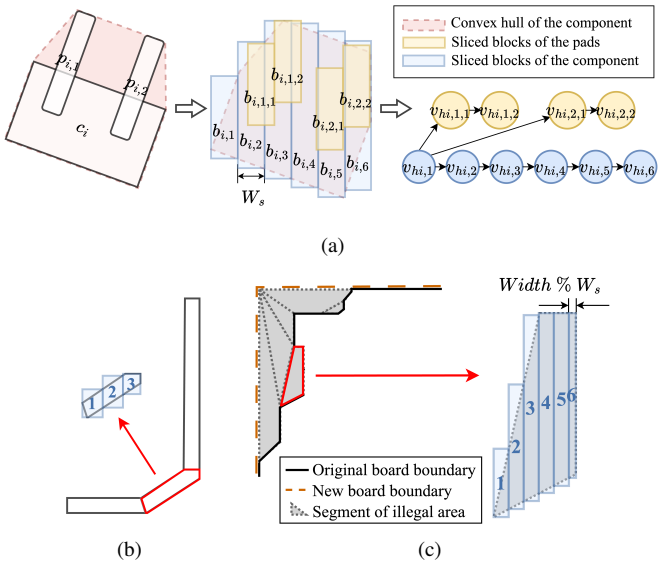


Fig. 3. Multi-rectangular approximation for irregular components and boards. (a) An example of a rotated component and the corresponding nodes on mTCG. The red dashed polygon represents the convex hull of the component. (b) An example of a highly non-convex component is split into three convex segments. (c) An example of an irregular board. The illegal area is split into convex segments. The last block width is the remainder of the polygon's width divided by W_s.

components (such as L-shaped heat sinks), we decompose them into several convex segments before applying the slicing technique, as shown in Figure 3(b). Each segment is treated as an individual component to other components, while internally, they recognize each other as parts of the same component. Lastly, irregular-shaped boards are approximated as rectangular boards containing fixed components that maintain zero clearance from other components. These fixed components are added to both C_f and C_b. Each illegal area within the rectangular board is divided into several convex polygons and sliced into blocks, as illustrated in Figure 3(c).

After processing all the components and pads, we calculate the available space in four directions—up, down, left, and right—around each pad $p_{i,k} \in P$. The calculation involves extending in each direction until encountering the nearest pad $p_{i,l}$ (where $p_{i,k}, p_{i,l} \in c_i$ and $k \neq l$) or reaching the board boundary. These recorded spaces for each pad will help the subsequent refinement stage for constructing an mTCG. Once all pad spaces are recorded, pads are grouped if the space between $p_{i,k}$ and $p_{i,l}$ is too small, creating a dead space. The grouped pads are then enclosed within a convex hull, treated as a single pad, and slicing techniques are reapplied.

B. Modified Transitive Closure Graph (mTCG) Construction

To define the relationships between components at this stage, we construct two directed graphs in an mTCG, representing the horizontal and vertical relations for each pair of blocks. An edge $v_i \rightarrow v_j$ in the horizontal (vertical) constraint graph G_h (G_v) represents that v_i should be placed on the left (bottom) side of v_j. Unlike the original TCG, which only considers horizontal or vertical relationships between macros, the mTCG defines both horizontal and vertical relationships for any two same-side components (c_i and c_j) or pads ($p_{i,k}$ and $p_{j,l}$). The relative position of c_i with respect to c_j falls into one of the four categories: top-left, bottom-left, bottom-right, and top-right. Following the same definition of feasibility as in a TCG [7], it is clear that, given the constraints imposed by the board outline and fixed components, a legal solution may not always be achievable with the initial graphs. Therefore, refinement of these graphs is necessary. The initialization and refinement processes for these graphs are detailed as follows.

1) mTCG Initialization

In the initialization step, we adopt the same method as in XDP [5] to construct edges on G_h and G_v. The edges between two nodes represent the constraints between the corresponding (p)blocks. For each edge $e_{ip,jq}$ and $e_{ikp,jlq}$ on an mTCG, we set the weights $w(e_{ip,jq})$ and $w(e_{ikp,jlq})$ to $(w_{i,k}+w_{j,q})/2+s_c$ and $(w_{i,k,p}+w_{j,l,q})/2+s_p$, respectively. The s_c and s_p denoted here represent the clearance constraints between components and pads.

In the original TCG definition [6], each pair of nodes must be connected by exactly one edge in either G_h or G_v. However, in the proposed mTCG, edges are only generated for vertices if the two (p)blocks are on the same side. To be specific, edges are generated in the following cases for single-sided PCBs.

- Between two blocks (or pblocks) from different components, where at least one component is movable.
- Between adjacent blocks (or pblocks) within the same component (or pad).
- Between block $b_{i,1}$ and pblock $b_{i,k,1}$ within the same component for every $c_i \in C$.

The example of edge construction within a single component is shown in the right part of Figure 3(a).

For the double-sided PCBs, the cases for edge generation are the same as those mentioned above, with the additional restriction that the (p)blocks must be on the same side for the first case. Furthermore, due to the influence of TH pads on the back side, an additional case is required:

- Between pblocks within TH pads and every back-side (p)blocks.

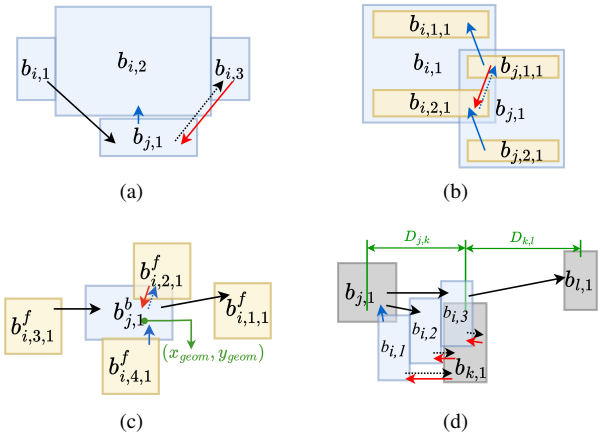


Fig. 4. mTCG refinement examples. The black arrows represent the edges in G_h, while the blue arrows represent the edges in G_v. The dashed arrows represent the initial edges, while the red solid arrows represent the refined edges. In (d), edges between ($b_{i,1}$, $b_{l,1}$) and ($b_{i,2}$, $b_{l,1}$) are omitted for clarity.

2) mTCG Refinement

To address the infeasible situations arising from the given constraints, we refine the edges in an mTCG for four specific cases: (1) between multi-block components or pads, (2) between same-side pads, (3) between TH pads and back-side components, and (4) between fixed and movable components. We traverse G_h and G_v for each case to identify specific edge combinations and apply appropriate refinement methods.

The first refinement aims to clearly define the geometric relationship between two components or pads, specifically whether $c_i(p_i)$ is positioned on the top-left, bottom-left, top-right, or bottom-right of $c_j(p_j)$. As shown in Figure 4(a), if $x_i < x_j$ and $x_{i,3} > x_{j,1}$, we reverse the edge direction from $v_{hj,1} \rightarrow v_{hi,3}$ to $v_{hi,3} \rightarrow v_{hj,1}$. Furthermore, when one of the components is fixed, the solution space may be restricted due to the insufficient space between the fixed component and the board boundaries, as illustrated in Figure 5. In this example, c_i is initially defined to be on the top-left of c_j, but the limited space on the left-side of c_j restricts the solution space, as shown in Figure 5(a). Therefore, when defining the relative positions between a movable component and a fixed component, we exclude areas where the movable component cannot be placed. In this case, c_i is instead positioned on the top-right of c_j, resulting in a more extensive solution space, as shown in Figure 5(b).

The second refinement applies to all edges between two same-side pads. The main concept is to ensure that the relationship between the pads reflects the relationship between their corresponding components to prevent legalization failures. As the example shown in Figure 4(b), assume that c_i is defined on the top-left corner of c_j, so we adjust $p_{i,2}$ to be above $p_{j,1}$. This requires reversing the edge direction from $v_{vi,2,1} \rightarrow v_{vj,1,1}$ to $v_{vj,1,1} \rightarrow v_{vi,2,1}$.

The third refinement addresses the edges between TH pads and back-side components in double-sided PCBs. The main concept is to avoid having components or pads trapped between multiple TH pads by detecting if there exists "obstacle$_1$ $\rightarrow$ target $\rightarrow$ obstacle$_2$" during traversal. Here,

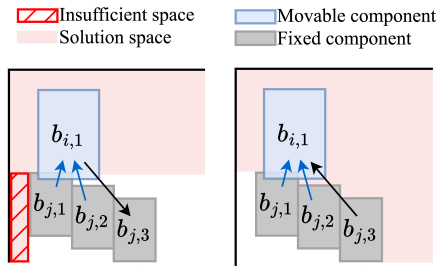


Fig. 5. The special refinement case for fixed components. The black arrows represent the edges in G_h, while the blue arrows represent the edges in G_v. The pink region represents the solution space for the movable c_i relative to the fixed c_j. (a) Before refinement: c_i is defined on the top-left to c_j. (b) After refinement: c_i is defined on top-right to c_j.

obstable$_1$ and obstable$_2$ are TH pads, while target represents the movable back-side (p)blocks, which can be consecutive blocks (or pblocks) within a single component (or pad). To refine this relationship, we calculate the geometric center of all TH pads $p_{i,k}$ for each c_i, denoted as $(x_{\text{geom}}, y_{\text{geom}})$. This geometric center is then compared with the target coordinates to determine the relationship between target and each $p^f_{i,k}$. However, if one of the following equations is satisfied, the edge directions remain unchanged.

$$\begin{cases} x_2 - x_1 \geq w(e_{1,t}) + w(e_{t,2}), & \forall e_{1,t}, e_{t,2} \in G_h \\ y_2 - y_1 \geq w(e_{1,t}) + w(e_{t,2}), & \forall e_{1,t}, e_{t,2} \in G_v \end{cases}, \tag{1}$$

where x_1 and x_2 represent the x-coordinates of obstacle$_1$ and obstacle$_2$, $e_{1,t}$ and $e_{t,2}$ represent the edges between (obstacle$_1$, target) and (target, obstacle$_2$), respectively.

As the example shown in Figure 4(c), since $x_{\text{geom}} > x^b_j$ and $y_{\text{geom}} < y^b_j$, each $p^f_{i,k}$ should be positioned in the bottom-right of c^b_j. However, assuming a clearance of zero, the horizontal distance in G_h between $b^f_{i,1,1}$ and $b^f_{i,3,1}$ is enough for $b^b_{j,1}$, so the edge $v^f_{i,3,1} \rightarrow v^b_{j,1}$ remains unchanged. In contrast, in G_v, the vertical distance between $b^f_{i,2,1}$ and $b^f_{i,4,1}$ is smaller than $h^b_{j,1}$, so the edge $b^b_{j,1} \rightarrow b^f_{i,2,1}$ must be reversed.

The final refinement addresses the infeasibility caused by fixed components. The example pseudocode for G_h is provided in Algorithm 1 to illustrate our approach. A similar algorithm is applied for G_v. First, we detect the edge combination of "obstacle$_1$ $\rightarrow$ target $\rightarrow$ obstacle$_2$", as shown in Line 3. Here, obstable$_1$ represents the source node or a fixed block, obstable$_2$ represents the sink node or a fixed block, while target represents the movable block or consecutive blocks within a single component.

For each detected set, the obstacles must satisfy the *refinement condition*:

$$overlap_height(obstacle, target) \geq r_{refine} \cdot h_t \tag{2}$$

on G_h, where h_t represents the height of the target (Line 5). The target can consist of consecutive blocks within a single component. We define the accommodating space $D_{j,k}$ between two obstacles b_j and b_k as the difference between their x- (or y-) coordinates. Suppose the *accept condition*:

$$D_{j,k} \geq r_{accept} \cdot (w(e_{1,t}) + w(e_{t,2})) \tag{3}$$

is violated, where $e_{1,t}$ and $e_{t,2}$ represent the edges between (obstacle$_1$, target) and (target, obstacle$_2$), respectively. In that case, we perform a backward breadth-first search (BFS) from obstacle$_1$ and a forward BFS from obstacle$_2$ to find fixed components that also satisfy both the two conditions by replacing the target as its entire component c_i (Lines 6–8). During the BFS step, we ultimately select the fixed component from the same level that achieves the minimum displacement when swapped with the target. Among the two candidates, the one with the smaller distance relative to the target is chosen to be the swapping object (Lines 9–10). As the example shown in Figure 4(d), obstacle$_1$ is $b_{j,1}$, the target consists of $\{b_{i,2}, b_{i,3}\}$, and obstacle$_2$ is $b_{k,1}$. Since the *refinement conditions* are satisfied, the accept check is triggered. After the BFS step, o_1 is identified as $b_{j,1}$ because there are no blocks positioned to the left of $b_{j,1}$, and o_2 is identified as $b_{k,1}$ because it satisfies both Inequalities (2) and (3). Since the distance between $b_{k,1}$ and c_i is smaller than that between $b_{j,1}$ and c_i, we intend to position c_i on the right side of $b_{k,1}$. The two parameters r_{refine} and r_{accept} are user-specified, ranging from 0 to 1.

Algorithm 1 Infeasibility Check for Fixed Components on G_h

1: **Input:** G_h, r_{refine}, r_{accept}
2: **Output:** Refined G_h
3: S = traverse(G_h);
4: **for** each s = {obstacle$_1$, target, obstacle$_2$} $\in$ S **do**
5: **if** (obstacle$_1$, target) satisfy the *refinement condition* **and** (target, obstacle$_2$) satisfy the *refinement condition* **then**
6: **if** the *accept condition* between (obstacle$_1$, obstacle$_2$) is violated **then**
7: o_1 = BFS_backward(obstacle$_1$);
8: o_2 = BFS_forward(obstacle$_2$);
9: o = smaller_displacement(o_1, o_2);
10: swap(o, target);
11: **end if**
12: **end if**
13: **end for**

During the refinement process, it is crucial to constantly monitor and eliminate any graph cycles that may arise from adjusting edge directions due to the directed and acyclic nature of an mTCG.

C. MILP Legalization

We transform the original linear programming problem used in the CG-based methods (e.g., those in previous works [5], [7]) into an MILP problem to enable optimization with arbitrary geometric relationships between irregular components. In this formulation, we introduce additional Boolean variables to specify whether the horizontal or vertical relationship applies between any two blocks, effectively expanding the solution space. This approach significantly enhances the success rate of legalization for irregular-shaped components, particularly in high-density PCB designs.

1) MILP Formulation

In order to minimize total displacement, the irregular PCB legalization will be formulated as an MILP problem and solved using an existing solver. We use $d(v_{hi,p})$ and $d(v_{vi,p})$ to represent the horizontal and vertical displacements of each block $b_{i,p}$. A similar way for each pblock $b_{i,j,p}$ is omitted for clarity. For each node $v_{hi,p}$ in G_h, we compute $L(v_{hi,p})$ and $R(v_{hi,p})$, representing the leftmost and rightmost coordinates where the block $b_{i,p}$ can be placed. The edges in the mTCG are categorized into *internal edges* and *external edges*, representing edges within one component and between two same-side components, respectively. The constraints for internal and external edges are called internal and external constraints.

The internal constraints of movable components aim to fix the relative positions of two blocks within a single component, as described below:

$$L(v_{hi,(p+1)}) = L(v_{hi,p}) + (x_{i,(p+1)} - x_{i,p}), \forall e_{ip,i(p+1)}, \in G_h, \tag{4}$$

$$R(v_{hi,p}) = R(v_{hi,(p+1)}) - (x_{i,(p+1)} - x_{i,p}), \forall e_{ip,i(p+1)} \in G_h, \tag{5}$$

where $x_{i,p}$ and $x_{i,(p+1)}$ represent the x-coordinates of $b_{i,p}$ and $b_{i,(p+1)}$. These constraints ensure that all blocks and pblocks within a single component have the same displacement, forcing them to move together to maintain the shape of the component and pads.

The external constraints of movable components aim to enforce non-overlapping and satisfy clearance constraints, as described below:

$$L(v_{hj,q}) = \max\left(a_{ip,jq}(L(v_{hi,p}) + w(e_{ip,jq}))\right) \forall e_{ip,jq} \in G_h, \tag{6}$$

$$R(v_{hi,p}) = \min\left((1 - a_{ip,jq})W + a_{ip,jq}\left(R(v_{hj,q}) - e_{ip,jq}\right)\right) \quad \forall e_{ip,jq} \in G_h, \tag{7}$$

$$\begin{cases} a_{ip,jq} \geq a_{i(p+1),jq}, & \text{if } c_i \text{ is set to the left of } c_j \\ a_{ip,jq} \leq a_{i(p+1),jq}, & \text{if } c_i \text{ is set to the right of } c_j \end{cases}, \tag{8}$$

where the Boolean variable $a_{ip,jq}$ indicates the relationship between blocks $b_{i,p}$ and $b_{j,q}$. Specifically, $a_{ip,jq}$ = 1 or 0 denotes that the two blocks are horizontally or vertically related. By introducing these Boolean variables for each pair of blocks, we can expand the solution space for all components and pads.

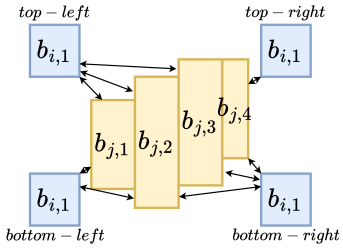


Fig. 6. The actual external constraints set between two same-side components are represented by bidirectional arrows. The corresponding sets of external constraints vary for the different relationships defined for c_i and c_j.

For every fixed blocks $b_{i,p}$ of $c_i \in F$, we set constraints as follows:

$$L(v_{hi,p}) = x_{i,p} \text{ and } R(v_{hi,p}) = x_{i,p}. \tag{9}$$

In the proposed mTCG, for any two same-side components c_i and c_j, ($c_i, c_j \in C$), there is an edge between each pair of $b_{i,p}$ and $b_{j,q}$. However, constructing constraints for each pair of blocks is redundant. To reduce the number of variables and constraints in our problem, we set the actual external constraints only between the same-side blocks of the two components. This configuration will not affect the solution space. Figure 6 illustrates an example.

The objective is to minimize the total Manhattan displacement, the sum of the total horizontal and vertical displacements, represented as $d(G_h) + d(G_v)$. The definition of $d(G_h)$ are shown in the following equations:

$$d(v_{hi,1}) = \begin{cases} L(v_{hi,1}) - x_{i,1}, & \text{if } L(v_{hi,1}) \geq x_{i,1} \\ x_{i,1} - R(v_{hi,1}), & \text{if } R(v_{hi,1})) \leq x_{i,1} \\ 0, & \text{otherwise} \end{cases}, \tag{10}$$

$$d(G_h) = \sum_{c_i \in C} d(v_{hi,1}). \tag{11}$$

2) Variable Pruning

The complexity of the MILP problem is given by the numbers of variables and constraints. Consequently, variable pruning is essential for our formulation.

Theorem 1: The MILP formulation has $O(M^2 + MN)$ variables and constraints, where M and N represent the numbers of movable and fixed blocks, respectively.

Proof 1: The mTCG contains $O(N + M)$ nodes and $O(M^2 + MN)$ edges. Each edge in the mTCG introduces a boolean variable and represents a corresponding constraint. As a result, the MILP formulation includes $O(M^2 + MN)$ variables and constraints.

The introduction of Boolean variables $a_{ip,jq}$ for external constraints in Equations 6–7 greatly expands the solution space. However, this expanded solution space often over considers potential placements, leading to unnecessary computational complexity for rectangular components.

In most global placement results, certain pairs of rectangular components exhibit strong horizontal or vertical alignment. We impose restrictions based on these alignments to adequately restrict the solution space for these components. Specifically, for the two horizontally arranged rectangles c_i and c_j, if $b_{i,1}$ and $b_{j,1}$ satisfy the condition that the overlapping length is greater than or equal to $r_{restrict} \cdot \max(h_{i,1}, h_{j,1})$, they are restricted to be horizontally related. A similar method is applied for vertically arranged rectangular components, except that they are restricted to be vertically related. Empirically, the pruning technique reduces an average 5% variables.

Additionally, restricting the direction of two rectangular components offers two key advantages. First, the solution space for irregular components remains, ensuring a high success rate in finding feasible solutions, even in high-density cases. Second, this approach preserves the relative positions established during global placement, maintaining placement consistency and retaining wirelength optimizations achieved during global placement.

IV. Experimental Results

We implemented our algorithm in C++ and performed all experiments on an AMD Ryzen Threadripper 3990X 64-Core Processor with a 2.9GHz CPU and 128GB Memory. Gurobi [16] was applied as the MILP solver, and an in-house commercial placer was used as the global placer. Besides, we used an in-house design rule checker to report the number of design rule violations (DRVs), which include non-overlapping, boundary, and clearance constraints. Two parameters r_{refine} and r_{accept} in the mTCG refinement stage for all experiments were set to 0.2 and 0.8, respectively. $r_{restrict}$ is set to 0.4 and 0.8 for single- and double-sided PCBs, respectively.

To evaluate our proposed algorithm, we conducted experiments on four industrial cases and three open-sourced benchmarks from PCBench [17]: *ice40_stm32_fsmc_test_pcb_fsmc_ice40*, *Horticulture_Light_EQ_3-Band_3-channel-led-driver-rev0.4*, and *mod12003_mod12003*, which are referred to as ice, Hor, and mod, respectively. All benchmarks are double-sided and include irregular components. Table II gives the statistics of the benchmarks. For each benchmark, the number of components (#C), number of pads (#P), and utilization (Util) for both sides are reported. We conducted experiments on both single- and double-sided cases to show the generality. For the evaluation of single-sided PCB placements, we focused on the front-side components, disregarding the influence of TH pads on the backside. Benchmark names were modified by appending the suffixes "-s" and "-d" to indicate whether they were treated as single-sided or double-sided.

We compared our work with XDP [5], a classical approach to macro legalization. Since the original XDP cannot be executed on the machine we used, we implemented their algorithms and denoted our implemented version as XDP* in the following statements. Besides, as XDP* cannot handle fixed components, we modified XDP* by incorporating additional constraints in Equation 9 to address the fixed components, ensuring fair comparisons. Moreover, in the original XDP, constraint graphs do not include edges between every pair of components to represent constraints like those used in the TCG and our mTCG. Therefore, the modifications of clearance constraints are difficult to be applied to modified XDP*. Therefore, in our first experiments, we evaluated performance without considering clearance constraints. Our approach is denoted as Ours when clearance constraints are not considered. The comparisons of XDP*, modified XDP*, and Ours are listed in Table III. For double-sided PCB experiments, the XDP-based method lacks techniques for simultaneous legalization on both sides, necessitating a sequential approach. Specifically, we first legalized the front side, treating the TH pads as obstacles, and then legalized the back side. The comparisons of our work with and without considering clearance constraints are listed in Table IV. The detailed numerical comparisons are reported in the following subsections.

Tables III and IV compare the percentage increases in HPWL (WL), displacement (Disp), and runtime (RT) for single-sided and double-sided PCBs. We use "LF" to indicate the cases where legalization failed, meaning no feasible solution was found. Since the original XDP cannot handle fixed components, cases containing fixed components are omitted from comparison, denoted as "-" in Tables III. The performance of the proposed algorithm is discussed in the following sections: Section IV-A compares our work to the rectangular-based legalization method, Section IV-B compares the effects of considering clearance constraints, and Section IV-C demonstrates the effects of pruning. See the detailed comparisons in the following subsections.

TABLE II
Benchmark Statistics.

Benchmark	Front Side		Back Side		#TH	Irregular
	#C/#P	Util(%)	#C/#P	Util(%)	Pads	Board?
PCB1	15/31	59.557	72/175	34.379	31	Yes
PCB2	20/42	51.290	69/157	19.945	42	Yes
PCB3	108/329	46.031	98/218	20.362	27	Yes
PCB4	88/257	36.936	119/331	39.877	55	No
ice	21/24	30.064	23/70	27.528	6	No
Hor	16/72	35.343	60/151	17.459	18	Yes
mod	186/488	40.424	3/7	1.291	47	No

A. Comparisons with XDP-based Methods on Single- and Double-sided PCBs

The XDP* and modified XDP* produced identical results for the cases without fixed components. Overall, our method outperformed modified XDP* in both wirelength and displacement. On average, for cases where both modified XDP* and Ours successfully find solutions, the modified XDP* resulted in a 15.84% increase in HPWL, and our method

TABLE III
COMPARISONS BETWEEN THE XDP-BASED METHODS AND OUR WORK WITHOUT CONSIDERING CLEARANCE CONSTRAINTS ON BOTH SINGLE-SIDED AND DOUBLE-SIDED PCBS.

Bench.	XDP*			modified XDP*			Ours		
	WL(%)	Disp(mm)	RT(s)	WL(%)	Disp(mm)	RT(s)	WL(%)	Disp(mm)	RT(s)
PCB1-s		-			LF		2.2	13.8	1.7
PCB2-s		-			LF		1.4	23.6	13.3
PCB3-s		-		23.0	417.8	0.6	26.3	192.2	20.4
PCB4-s		-		0.8	23.8	6.6	0.6	22.0	11.3
ice-s		-		3.17	51.6	0.4	2.9	40.0	2.1
Hor-s		-			LF		15.1	86.3	0.8
mod-s	29.4	1165.9	8.0	29.4	1165.9	8.00	10.2	263.9	257.5
PCB1-d		-			LF		10.6	129.9	96.8
PCB2-d		-			LF		16.4	152.9	202.7
PCB3-d		-			LF		38.5	290.5	40.7
PCB4-d		-			LF		6.3	210.1	139.9
ice-d		-		9.5	244.0	0.67	0.7	136.7	16.7
Hor-d		-			LF		23.6	157.4	14.4
mod-d	29.2	1175.4	8.0	29.2	1175.4	8.0	14.1	252.7	232.9

TABLE IV
COMPARISONS BETWEEN WITH AND WITHOUT CONSIDERING CLEARANCE CONSTRAINTS ON BOTH SINGLE-SIDED AND DOUBLE-SIDED PCBS.

Bench.	Ours				Ours			
	WL(%)	Disp(mm)	RT(s)	#DRV	WL(%)	Disp(mm)	RT(s)	#DRV
PCB1-s	2.2	13.8	1.7	0	3.0	23.9	1.0	0
PCB2-s	1.4	23.6	13.3	1	1.4	26.7	14.2	0
PCB3-s	26.3	192.2	20.4	166	33.6	271.1	35.2	4
PCB4-s	0.6	22.0	11.3	5	0.8	25.5	17.9	0
ice-s	2.9	40.0	2.1	0	2.9	40.0	1.9	0
Hor-s	15.1	86.3	0.8	1	15.1	86.6	0.8	0
mod-s	10.2	263.9	257.5	14	9.9	246.4	172.5	4
PCB1-d	10.6	129.9	96.8	83	24.0	229.0	203.0	6
PCB2-d	16.4	152.9	202.7	91	24.0	229.0	203.0	0
PCB3-d	38.5	290.5	40.7	227	40.7	334.4	128.1	7
PCB4-d	6.3	210.1	139.9	29	8.7	272.4	172.0	6
ice-d	0.7	136.7	16.7	0	0.6	136.7	17.9	0
Hor-d	23.6	157.4	14.4	15	23.8	162.5	14.1	0
mod-d	14.1	252.7	232.9	96	10.5	249.7	209.6	0
Average	1.00	1.00	1.00	1.00	1.19	1.28	1.27	0.04

achieved a smaller increase of 13.77%. Regarding displacement, our primary objective, we achieved a notable 61.0% improvement compared to modified XDP*. Regarding runtime, our work required 6.60X more computation time on average. However, all other cases obtained solutions within 10 minutes, which is acceptable compared to the manual effort in most commercial applications.

As shown in Table III, cases such as PCB1 and PCB2, where the XDP-based method fails to generate feasible solutions, are typically associated with higher utilization rates. Additionally, for double-sided benchmarks, the modified XDP* failed on almost all benchmarks, revealing the robustness of our work. The weakness of modified XDP* can be attributed to its inability to globally adjust constraint graphs, which increase the probability of failure when considering TH pads, as illustrated in Figure 4(c).

A comparison of the legalization results between modified XDP* and Ours is shown in Figures 7(a)–7(c). Figure 7(a) gives the global placement results. The results for XDP* in Figure 7(b) exhibit larger displacements, leading to a further increase in wirelength in comparison to our approach in Figure 7(c).

B. Clearance Constraint Consideration

The inclusion of clearance constraints during the legalization stage is essential to minimize the number of DRVs. Without this consideration, resolving these violations would require extensive manual adjustments, consuming significant time and effort. In cases PCB3-s, mod-s, PCB1-d, PCB3-d, and PCB4-d, these DRVs in the right column were caused by floating-point precision issues. In other words, the abutted components may lead to false alarms in the DRV report. On average, incorporating clearance constraints reduces the number of DRVs by nearly 100%, though it introduces a 27% runtime overhead, which is acceptable. Figures 7(c)–7(d) illustrate comparisons between cases with and without clearance constraints.

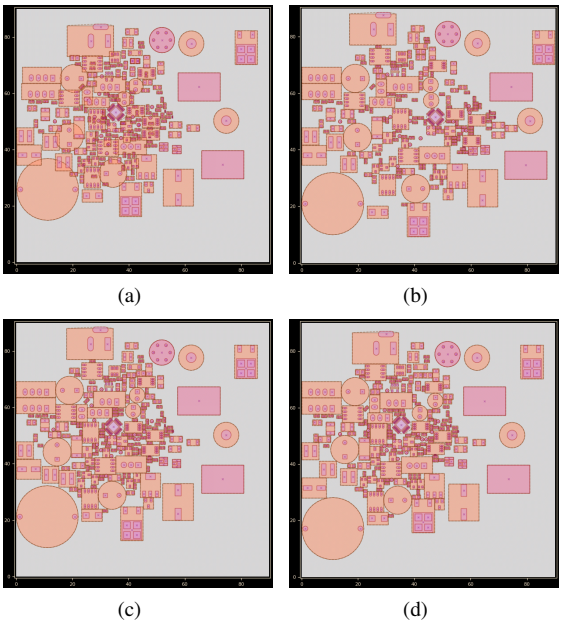


Fig. 7. The legalization process for mod-s. (a) Global placement result. (b) Rectangular-based method XDP* (without considering clearance constraints). (c) Ours (without considering clearance constraints). (d) Ours (considering clearance constraints).

C. Effectiveness of Pruning

Figure 8 shows our pruning technique's effectiveness in eliminating Boolean variables from our formulation. By pruning variables, we can significantly reduce runtime by restricting the solution space between two rectangular components, which also gives the advantage of preserving the relations in global placement, especially benefits to horizontally or vertically arranged components.

As the parameter $r_{restrict}$ decreases, displacement increases while runtime decreases. This trade-off helps preserve the relative positions of regularly arranged rectangular components during global placement.

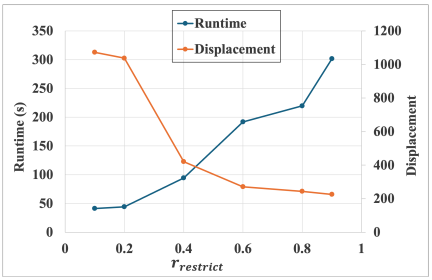


Fig. 8. Effectiveness of the pruning technique by adjusting parameter $r_{restrict}$ on mod-d considering clearance constraints.

V. CONCLUSIONS

This paper has presented the first legalization approach for irregularly shaped and any-oriented components and boards, accommodating both single- and double-sided PCB designs while addressing the challenges posed by TH pads. We have proposed a multi-rectangular approximation technique integrated with the MILP formulation to optimize utilization in compact layouts and expand the solution space. Additionally, a refinement stage based on a CG-based method enhances robustness, enabling efficient handling of TH pads and fixed components. Experimental results have validated the effectiveness and robustness of our proposed approach, showing significant improvements in irregular PCB placements compared to traditional rectangle-based legalization methods.

ACKNOWLEDGMENT

We thank Mr. Pin An Chen, Mr. Wei No Chen, Mr. Sean Huang, Ms. Ann Wei, Mr. Charles Liu, and Mr. Denver Liu of Delta Electronics for their great help in this research.

REFERENCES

[1] C.-H. Tsou, S.-Y. Lin, W.-C. Hung, and Y.-W. Chang, "Late breaking results: Modern automatic PCB placement with complex constraints," in *Proceedings of ACM/IEEE Design Automation Conference*, San Francisco, CA, June 2024, pp. 1–2.

[2] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB placement with obstacles based on sequence-pair," in *Proceedings of ACM International Symposium on Physical Design*, Napa Valley, CA, April 1997, pp. 26–31.

[3] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and D. Liu, "MP-Trees: A packing-based macro placement algorithm for mixed-size designs," in *Proceedings of ACM/IEEE Design Automation Conference*, San Diego, CA, June 2007, pp. 447–452.

[4] T.-C. Chen, P.-H. Yuh, Y.-W. Chang, F.-J. Huang, and T.-Y. Liu, "MP-Trees: A packing-based macro placement algorithm for modern mixed-size designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 9, pp. 1621–1634, September 2008.

[5] J. Cong and M. Xie, "A robust mixed-size legalization and detailed placement algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1349–1362, May 2008.

[6] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph-based representation for non-slicing floorplans," in *Proceedings of ACM/IEEE Design Automation Conference*, Las Vegas, NV, June 2001, pp. 764–769.

[7] H.-C. Chen, Y.-L. Chuang, Y.-W. Chang, and Y.-C. Chang, "Constraint graph-based macro placement for modern mixed-size circuit designs," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, San Jose, CA, November 2008, pp. 218–223.

[8] J. Ousterhout, "Corner stitching: A data-structuring technique for VLSI layout tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 1, pp. 87–100, January 1984.

[9] J.-M. Lin, Y.-L. Deng, Y.-C. Yang, J.-J. Chen, and Y.-C. Chen, "A novel macro placement approach based on simulated evolution algorithm," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, Westminster, CO, November 2019, pp. 1–7.

[10] X. Wang, N. Xu, and Y. Chen, "Automatic multi-constraint placement of printed circuit board," in *Proceedings of the International Conference on Communications, Circuits and Systems*, Xiamen, China, May 2024, pp. 1–6.

[11] H. Chen, Y. Chen, and N. Xu, "Automatic placement of PCB functional modules based on regional division," in *Proceedings of the International Conference on Communications, Circuits and Systems*, Xiamen, China, May 2024, pp. 51–58.

[12] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B*-Trees: A new representation for non-slicing floorplans," in *Proceedings of ACM/IEEE Design Automation Conference*, Los Angeles, CA, June 2000, pp. 458–463.

[13] C.-K. Cheng, C.-T. Ho, and C. Holtz, "Net separation-oriented printed circuit board placement via margin maximization," in *Proceedings of Asia and South Pacific Design Automation Conference*, Taipei, Taiwan, January 2022, pp. 288–293.

[14] M. Adamowicz and A. Albano, "Nesting two-dimensional shapes in rectangular modules," *Computer-Aided Design*, vol. 8, no. 1, pp. 27–33, January 1976.

[15] J. F. Oliveira, A. M. Gomes, and J. S. Ferreira, "TOPOS – A new constructive algorithm for nesting problems," *OR-Spektrum*, vol. 22, pp. 263–284, May 2000.

[16] Gurobi. [Online]. Available: https://www.gurobi.com/

[17] H. Zhu, X. Guo, Y. Jin, and X. Zhang, "PCBench: Benchmarking of board-level hardware attacks and trojans," in *Proceedings of Asia and South Pacific Design Automation Conference*, Tokyo, Japan, January 2021, pp. 396–401.

Age-of-Information Minimization for Data Aggregation in Energy-Harvesting IoTs

Bingkun Yao
City University of Hong Kong
bingkun.yao@cityu.edu.hk

Mun Choon Chan
National University of Singapore
chanmc@comp.nus.edu.sg

Hong Gao*
Zhejiang Normal University
honggao@zjnu.edu.cn

Zhe Jiang*
Southeast University
zhejiang.uk@gmail.com

Nan Guan*
City University of Hong Kong
nanguan@cityu.edu.hk

***Abstract*—Energy Harvesting (EH) technology has emerged to prolong the lifetime of Internet of Things (IoT) devices. However, in EH-IoTs, the reliance on external energy sources introduces challenges in maintaining up-to-date information. To quantify data freshness in such systems, researchers have introduced the Age-of-Information (AoI) metric, which measures the time elapsed since the generation of the most up-to-date information received by the user. Consequently, the problem of AoI minimization has been studied extensively in EH-IoTs to ensure timely data delivery. While data aggregation is a fundamental task for IoTs, existing works on AoI minimization in EH-IoTs have only considered scenarios where sensory data is updated by individual source nodes. The problem has not been investigated for data aggregation, in which the sensory data is aggregated from multiple source nodes. In this paper, we study the problem of AoI minimization for Data Aggregation in EH-IoTs. To address this problem, we propose an energy-adaptive node scheduling algorithm consisting of both offline scheduling and online adjustment. Extensive simulations and testbed experiments verify the high performance of our algorithm in terms of AoI minimization and energy efficiency.**

I. Introduction

In recent years, Energy-Harvesting (EH) technology has emerged as a crucial solution for Internet of Things (IoT) systems [1]–[3]. By enabling nodes to harvest energy from the ambient environment, EH technology significantly prolongs the lifetime of IoT networks and provides a promising approach for green computing [4].

In EH-IoTs, nodes rely on harvested energy to operate, which introduces unique challenges in maintaining the freshness of sensory data. Due to the intermittent nature of energy harvesting, nodes can only sense and transmit data when sufficient energy is accumulated, leading to outdated information. To tackle this issue, researchers have introduced Age of Information (AoI) [5] to quantify data freshness. To be specific, AoI is defined as the time difference between the present and the timestamp of the most up-to-date sensory data received by the user.

To ensure data freshness in EH-IoTs, the problem of AoI minimization in EH-IoTs is of great importance. While extensive research has focused on this problem [9]–[30], these studies concentrate on the scenario where each interested data item is updated by individual nodes (referred to as *individual-source data collection*).

Data aggregation, as one of the fundamental tasks in IoTs, presents a distinct scenario where each interested data item is the aggregation of data from multiple nodes. For instance, in marine environmental monitoring, obtaining the average temperature of a sea area requires aggregating temperature readings from multiple sensor nodes distributed across the region.

The AoI metric in the scenario of data aggregation is quite different from that in individual-source data collection. In individual-source data collection, the data captured by a single node can be directly utilized by the user. Thus, the AoI metric for individual-source data collection ***only reflects the freshness of data collected by a single node.*** In contrast, for the scenario of data aggregation, the data of interest is aggregated from multiple nodes' samples. A data item is considered to be received only when all its constituent data have been transmitted, and its timestamp is determined by the timestamps of its aggregated data. Therefore, the AoI in this scenario ***represents the freshness of data jointly captured by multiple nodes***, making existing solutions designed for individual-source collection inapplicable to data aggregation contexts.

Moreover, the characteristic of energy harvesting makes AoI minimization in data aggregation scenarios particularly challenging in EH-IoTs. This is because different nodes typically have distinct energy harvesting rates, resulting in diverse duty cycles. When multiple data items need to be updated by nodes with different energy harvesting rates, coordinating these nodes to effectively reduce the overall AoI becomes a complex problem.

In this paper, we investigate the problem of **AoI** minimization for Data **Agg**regation in EH-IoTs (**AoI-Agg** for short), which has not been studied in the literature. The main contributions are as follows: First, we formalize the AoI-Agg problem that aims to minimize the AoI metric in the context of data aggregation. Second, we propose a bipartite approach: an offline algorithm to generate the basic schedule, and an online adjustment scheme to maintain AoI performance under fluctuating energy harvesting rates. Finally, we validate our proposed method through extensive simulations and testbed experiments, demonstrating its effectiveness in terms of both AoI and energy efficiency.

II. Problem Statement

A. System Model

Network Model. The network consists of multiple sensor nodes $N = \{n_1, n_2, ...\}$ that monitor a set of targets $T = \{t_1, t_2, ...\}$. Here, a target refers to any kind of physical entity or phenomenon that users might be interested in. For example, the temperature of a water body. Each node n covers a subset of targets, denoted as $\Gamma(n)$, while $Cover_node(t)$ represents the set of nodes that can monitor target t.

The system operates in discrete time, with the entire working divided into K slots with equal length. In each slot t_x ($t_x \in [1,K]$), a node n operates in either ***active*** or ***idle*** state. When active, n transmits the sensed data of all the targets in $\Gamma(n)$, and the data is sampled by n at the t_s-th slot. We represent this action as a ***sub-instance*** denoted by a triple $s = (n, t_x, t_s)$, where t_x and t_s are called the ***sending-time*** and ***sub-timestamp*** of s.

We introduce the concept of sub-instance because the sensed data of a target is aggregated from the data of multiple nodes, and a single sub-instance typically represents only a partial observation of the target state. We denote the set of all the sub-instances generated by n as $S(n)$, which represents the working schedule of node n. Each

* Corresponding authors.

node can transmit at most one sub-instance per timeslot, meaning that for any two sub-instances s_1, $s_2 \in S(n)$, $s_1.t_x \neq s_2.t_x$.

Energy model. For each node $n \in N$, we define its energy parameters as follows:

- $B_n(i)$ represents n's available energy at the start of slot i.
- $EH_n(i)$ represents the energy harvesting rate during slot i.
- $B_{max}(n)$ specifies the maximum energy storage capacity.
- $E_c(n)$ indicates the energy consumption for one active slot.

$E_c(n)$ depends on $|\Gamma(n)|$ and the size of the sensed data of each target in $\Gamma(n)$. We assume $E_c(n) \leq B_{max}(n)$ to ensure the node can complete at least one active slot when fully charged. During each slot i, if node n is active, it first consume $E_c(n)$ units of energy, then harvests $EH_n(i)$ units from the environment.

Data Aggregation Model. For each target $t \in T$, each data instance of t needs to be aggregated from the data sampled by $Agg_num(t)$ different nodes. In real-world applications, usually redundant nodes are deployed to improve the coverage of the network. Thus, $Agg_num(t) \leq |Cover_node(t)|$.

We define a ***combination*** Com as a set of sub-instances that can form a complete data instance of a target t. A valid combination must satisfy these conditions:

A1: Causality Constraint. For each sub-instance $s \in Com$, we have $s.t_x \geq s.t_s$; This ensures that transmission occurs after sampling.

A2: Node Diversity Constraint. Com consists of $Agg_num(t)$ sub-instances, each from a distinct node in $Cover_node(t)$.

A3: Temporal Consistency Constraint. For any s, $s' \in Com$, $|s.t_s - s'.t_s| \leq \delta$, where δ denotes the maximum allowable temporal deviation between sub-instances. Sub-instances with excessive temporal differences cannot be aggregated as they may not represent a consistent state.

A combination Com has two key attributes:

- ***Arrival time*** $Ar(Com) = Max\{s.t_x | s \in Com\}$. This denote the time when Base Station (BS) receives the data instance aggregated from Com, at which all sub-instances in Com complete their transmission.
- ***Timestamp*** $Ts(Com) = Min\{s.t_s | s \in Com\}$. The timestamp of the aggregated data instance is the smallest sub-timestamp of all its sub-instances.

Let $Q(t)$ denote the set of all valid combinations for target t. We restrict each sub-instance to be aggregated at most once for each target, namely, for any Com, $Com' \in Q(t)$, $Com \cap Com' = \emptyset$;

B. Definition of AoI-Agg Problem

For each target t, at slot i, the AoI of t is the difference between the current time and the timestamp of the most up-to-date t's data instance maintained at the BS:

$$AoI(t,i) = i - Max\{Ts(Com) | Ar(Com) \leq i \ \&\ Com \in Q(t)\}$$

Following the convention in existing works on AoI minimization [14], [21], [36], at the start of slot 1, BS maintains an initial data instance with timestamp 0 for each target t.

The objective of AoI-Agg is to compute a working schedule for all the nodes such that the average AoI of all the targets throughout the entire working duration is minimized. We formally define the problem as follows:

Input: 1) Network parameters N, T, $\Gamma(n)$ for each $n \in N$, K;

2) Energy parameters $B_{max}(n)$, $E_c(n)$, $B_n(0)$ for each $n \in N$, the predicted value of $EH_n(i)$ for each $n \in N$ and $i \in [1, K]$;

3) $Agg_num(t)$ for each $t \in T$ and δ.

Output: Working schedule $S(n)$ for each $n \in N$, Combination set $Q(t)$ for each $t \in T$ such that the following is minimized:

TABLE I
SYMBOLS AND NOTATIONS

Notation	Description
N	The set of all energy-harvesting (EH) nodes
T	The set of all interested targets
$\Gamma(n)$	The set of targets that can be covered by node n
$Cover_node(t)$	The set of nodes that cover target t
K	The length of the whole working duration
$s = (n, t_x, t_s)$	A sub-instance of node n with sending-time t_x and sub-timestamp t_s
$S(n)$	The set of all sub-instances of n
$B_n(i)$	The energy level of n at the start of the i-th slot
$EH_n(i)$	The EH rate of n during the i-th slot
$E_c(n)$	The energy consumption for an active slot of n
$Agg_num(t)$	The aggregation number of target t
Com	A combination corresponding to a data instance
$Ar(Com)$	The arrival time of the data instance aggregated from Com, $Ar(Com) = Max\{s.t_x \| s \in Com\}$
$Ts(Com)$	The timestamp of the data instance aggregated from Com, $Ts(Com) = Min\{s.t_s \| s \in Com\}$

$$AVG_AoI = \frac{\sum_{i=1}^{K} \sum_{t \in T} AoI(t,i)}{K|T|}$$

Assuming perfect knowledge of $EH_n(i)$ is impractical. Therefore, we use predicted values of $EH_n(i)$ as problem input. In our evaluation, we employ Pro-Energy [35] to estimate these values. During schedule execution, a node n remains idle if its available energy falls below $E_c(n)$, even when scheduled to be active.

The AoI-Agg problem is NP-Hard, as a special case of it is equivalent to the 2-DSC problem [43], which has been proven to be NP-complete. Therefore, AoI-Agg is NP-hard. We omit the detailed proof due to space limitations.

III. AM-AGG: COMPUTATION OF A BASIC SCHEDULE

In this section, we propose the **A**oI **M**inimization for Data **Agg**regation (**AM-Agg**) algorithm to compute a basic working schedule for each node. Overall, AM-Agg iterates on the timeline starting from the first slot till the last slot, dividing the entire working duration into several periods, and computing the node working schedule within each period. The i-th iteration consists two phases: (1) Determine the scale of the i-th period so that at least one new data instance could be updated for each target within the i-th period; (2) Scheduling nodes for the i-th period to enhance the energy usage and reduce AoI.

A. Phase I: Determine the Scale of the i-th Period

Let $[start_i, end_i]$ denote the i-th period from slot $start_i$ to end_i. As AM-Agg algorithm process periods in chronological order, $start_1 = 1$ and $start_i = end_{i-1} + 1$. Within each period, we need to find a combination Com_t for each target $t \in T$ to ensure all target data are updated. To minimize the data staleness, we aim to minimize end_i while satisfying this update requirement. The sub-problem of this phase can be defined as:

Input: $start_i$.
Output: end_i, a new combination Com_t for each $t \in T$ such that $Ar(Com_t) \in [start_i, end_i]$ and end_i is minimized.

Now consider this sub-problem. We define a ***full-coverage*** as a node set denoted by FC such that for each target t, $|Cover_node(t) \cap FC| \geq Agg_num(t)$. Also, let $Ready(FC, j)$ denote the first slot after or equal to j when all nodes in FC are expected to enough energy to become active[1]. We use FC_{min} to denote the full-

[1]This time is predicted based on the input energy parameters. e.g., If $B_n(10) = 20$, $E_c(n) = 100$ and the predicted $EH_n(k) = 10$ for $k \geq 10$, n is expected to have enough energy to become active at slot 18.

coverage that minimizes $Ready(FC, start_i)$. Given FC_{min}, a solution for the above subproblem can be constructed as follows:

1) For each node $n \in FC_{min}$, we have a new sub-instance $s_{new}(n)$ where $s_{new}(n).t_x = s_{new}(n).t_s = Ready(FC_{min}, start_i)$. This is because n is expected to have enough energy to be active at slot $Ready(FC_{min}, start_i)$.

2) For each target $t \in T$, Com_t is composed of new sub-instances from nodes that are can cover t:

$$Com_t = \{s_{new}(n) | n \in Cover_node(t) \cap FC_{min}\}$$

and $|Com_t| = Agg_num(t)$.

Through above method, end_i is minimized, and $end_i = Ready(FC_{min}, start_i)$. Hence, to solve the subproblem above, we only need to compute FC_{min}. Also, to reduce energy consumption, FC_{min} should not contain redundant nodes, namely, any proper subset of FC_{min} is not a full-coverage. FC_{min} is computed as follows:

1) Initiate FC_{min} as N. Then for each $n \in FC_{min}$, if n does not have enough energy to become active in residual working duration, remove n from FC_{min}.

2) Sort FC_{min} by the descending order of $Node_ready(n, start_i)$, which is the first slot j after or equal to $start_i$ when $B_n(j)$ is expected to reach $E_c(n)$.

3) For each node $n \in FC_{min}$, if $FC_{min}/\{n\}$ is a full-coverage, delete n from FC_{min}.

If FC_{min} is not a full-coverage, there is no feasible solution for the above sub-problem, and we simply set $end_i = K$. This indicates that the AM-Agg will end after the i-th iteration.

B. Phase II: Scheduling within the i-th Period

Phase I only consider nodes in FC_{min}, and these nodes are only scheduled to be active at slot end_i. To further improve the energy usage and reduce the AoI, Phase II of AM-Agg tries to find more new combinations within the i-th period, including two steps: (1) Find the set of targets T_{new} to be updated. (2) Compute a new combination for each target in T_{new} such that the arrival time and the timestamp of each combination is minimized and maximized respectively. Two steps are repeated until new combinations could not be found.

Before the detailed description of this phase, to allocate new active slots for each node n, AM-Agg maintains the interval $[\boldsymbol{Fc(n)}, \boldsymbol{Lc(n)}]$ during which n **could become active**, which satisfies the following:

1) $Fc(n)$, $Lc(n) \in [La(n)+1, end_i\text{-}1]$, and $La(n)$ denotes the last slot before end_i[2] at which n has been scheduled to be active. This is because the schedule is computed in chronological order. If n has not been scheduled to be active, $La(n) = 0$.

2) $B_n(Fc(n))$ and $B_n(Lc(n))$ are expected to exceed $E_c(n)$, and If $n \in FC_{min}$, then n becoming active in that slot is not expected to cause $B_n(end_i)$ to fall below $E_c(n)$. This is to ensure the scheduling complies with energy constraints.

If no slots satisfy above conditions, $Fc(n) = Lc(n) = -1$, which indicates that we cannot assign new active slots for n in the i-th period. Next, we describe two steps of Phase II in detail.

Step 1: Compute T_{new}. We need to ensure that a new data instance could be updated for each target in T_{new}. Firstly, let N_{new} denote the set of nodes that could transmit new sub-instance (i.e., $Fc(n) \neq -1$). Then, T_{new} is the set of all the targets that N_{new} can cover, namely,

$$T_{new} = \{t | \, |Cover_node(t) \cap N_{new}| \geq Agg_num(t)\}$$

If $T_{new} = \emptyset$, no new combinations could be found during the i-th period and Phase II ends.

Step 2: Compute a new Com_t for each $t \in T_{new}$. This step has two substeps: (1) To update the sensory data more quickly, $Ar(Com_t)$ is minimized for each $t \in T_{new}$. (2) Given $Ar(Com_t)$, $Ts(Com_t)$ is maximized to ensure that the sensory data is received by the BS as soon as being collected.

[2]As the data instance of each target is updated to BS at slot end_i, Phase II does not consider the schedule for slot end_i.

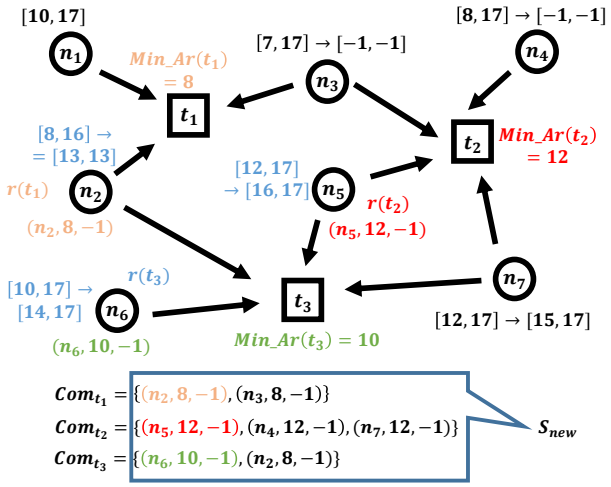


Fig. 1. An example of minimizing $Ar(Com_t)$ for each $t \in T_{new}$, and $T_{new} = \{t_1, t_2, t_3\}$.

Substep 1: Minimize $Ar(Com_t)$. Com_t consists of newly-computed sub-instances, whose set is denoted by S_{new}. The sub-problem of this substep is:

> **Input:** T_{new}, N_{new}.
> **Output:** S_{new}, $Com_t \subseteq S_{new}$ for each $t \in T_{new}$ such that $Ar(Com_t)$ is minimized.

We illustrate how to solve the above problem by the example shown in Fig. 1, where $T_{new} = \{t_1, t_2, t_3\}$, $N_{new} = \{n_1, ...n_7\}$. $Agg_num(t_1) = Agg_num(t_3) = 2$ and $Agg_num(t_2) = 3$. The $[Fc(n), Lc(n)]$ for each node is also marked in the figure.

Firstly, according to the definition of *arrival time* in Sec. II-A, for each $t \in T_{new}$, $Ar(Com_t)$ is lowered bounded by $Min_Ar(t) = Fc(r(t))$, in which $r(t)$ is the node covering t with the $Agg_num(t)$-th largest $Fc(n)$. To minimize $Ar(Com_t)$, AM-Agg adds a sub-instance $(r(t), Min_Ar(t), -1)$[3] into Com_t and S_{new}. As highlighted in red in Fig. 1, $Agg_num(t_2) = 3$ and among 4 nodes covering t_2, n_5 has the third largest Fc value (i.e., 12). Thus, $Min_Ar(t_2) = 12$, $r(t_2) = n_5$ and the sub-instance $(n_5, 12, -1)$ is added into Com_{t_2} and S_{new}. Com_{t_1} and Com_{t_3} are also processed similarly (Highlighted in orange and green). Besides, as highlighted in blue, after n_5, n_2 and n_6 are scheduled to be active at slot 12, 8 and 10, their $[Fc(n), Lc(n)]$ are updated.

Then, AM-Agg computes the remaining sub-instances for each Com_t. The basic idea is that for each node n $(n \neq r(t))$ covering t, the sending-time of n's sub-instance should be as close as possible but no more than $Min_Ar(t)$. This is to make $Ts(Com_t)$ closer to $Ar(Com_t)$ while ensuring that $Ar(Com_t) = Min_Ar(t)$. The example in Fig. 1 details this point as described below. Mention that targets in T_{new} are considered in the ascending order of $Min_Ar(t)$ (For our example, it is $t_1 \rightarrow t_3 \rightarrow t_2$):

(1) Firstly, consider Com_{t_1}. Since $Fc(n_1) = 10 > Min_Ar(t_1) = 8$ and there is no sub-instances of n_1 in S_{new}, we cannot assign n_1's sub-instance to Com_{t_1}. Also, since $Fc(n_3) = 7 < 8$, n_3 could become active at slot 8. Thus, AM-Agg adds a new sub-instance $(n_3, 8, -1)$ into Com_{t_1}. After this, $Fc(n_3) = Lc(n_3) = -1$.

(2) Secondly, as for Com_{t_3}, since $Fc(n_5)$, $Fc(n_7) > Min_Ar(t_3) = 10$, and the sending-times of their sub-instances in S_{new} are larger

[3]-1 means the sub-timestamp of this sub-instance has not been determined.

Algorithm 1: Substep 1: Minimize $Ar(Com_t)$

Input: N_{new}, T_{new}, $Fc(n)$ and $Lc(n)$ for each $n \in N_{new}$;
Output: A new combination Com_t for each $t \in T_{new}$, a set of new sub-instances S_{new};

```
S_new ← ∅;
for each t ∈ T_new do
    r(t) ← the Agg_num(t)-th ranked node in Cover_node(t)∩N_new in the ascending order of Fc(n);
    Min_Ar(t) ← Fc(r(t));
    Com_t ← {(r(t), Fc(r(t)), −1)}, S_new ← S_new ∪ Com_t;
Sort T_new in the ascending order of Fc(r(t));
for each t ∈ T_new do
    S_can(t) ← ∅; /*The candidate sub-instances of Com_t*/
    for each n ∈ Cover_node(t)/r(t) do
        Compute a candidate sub-instance s of n to maximize s.t_x satisfying s.t_x ≤Min_Ar(t), S_can(t) ← S_can(t) ∪ {s};
    Select Agg_num(t)-1 sub-instances with the largest sending-time from S_can(t), and add them to Com_t;
    S_new ← S_new ∪ Com_t;
```

than $Min_Ar(t_3)$ (i.e., that of n_5 is 12), we must consider n_2. Then, because $Fc(n_2) = 13 > 10$, to ensure $Ar(Com_{t_3}) = Min_Ar(t_3)$, AM-Agg adds the existing sub-instance (n_2, 8, -1) in S_{new} into Com_{t_3}.

(3) Finally, consider Com_{t_2}. Since $Fc(n_4)$, $Fc(n_7) < Min_Ar(t_2) = 12$, AM-Agg adds ($n_4$, 12, -1) and ($n_7$, 12, -1) into Com_{t_2}. As for n_3, since $Fc(n_3) = Lc(n_3) = -1$, we could not find a new sub-instance of it. Moreover, although the sending-time of n_3's sub-instance in S_{new} is less than $Min_Ar(t_2)$ (8<12), it is not as close to $Min_Ar(t_2)$ as those of n_4 and n_7. Thus, n_3's sub-instance is not added into Com_{t_2}.

Algorithm 1 summarizes the procedure of Substep 1. Through this method, $Ar(Com_t)$ is minimized for each $t \in T_{new}$.

Substep 2: Maximizing $Ts(Com_t)$ by computing the sub-timestamps of sub-instances in Com_t. For a target $t^* \in T_{new}$, before the computation of $Ts(Com_{t*})$, the sub-timestamp of some Com_{t*}'s sub-instances may have been determined. This is because Com_{t*} may intersect with other Com_t ($t \in T_{new}$). Thus, we only need to consider those sub-instances with undetermined sub-timestamps, whose set is denoted by $UD(Com_{t*})$. The subproblem is defined as:

Input: S_{new}, $UD(Com_{t*})=\{s | s \in Com_{t*} \& s.t_s = -1\}$;
Output: $s.t_s$ for $s \in UD(Com_{t*})$ such that $Ts(Com_{t*})$ is maximized.

We solve this by maximizing the sub-timestamp of sub-instances in $UD(Com_{t*})$ in the ascending order of their sending-times.

For each sub-instance $s \in UD(Com_{t*})$, we aim to maximize its sub-timestamp. To ensure Com_{t*} is valid, Com_{t*} must satisfy conditions A1, A2, and A3. Algorithm 1 ensures that $|Com_{t*}| = Agg_num(t^*)$, thus satisfying A2. Our task now is to maximize the sub-timestamp of s while meeting A1 and A3.

1) Firstly, ensure A3 (Temporal Consistency Constraint) is satisfied. Since $Com_{t*} \subseteq S_{new}$, all sub-instances in S_{new} may limit the value of $s.t_s$. An example is shown in Fig. 2, in which δ = 1, t^*, t_1, $t_2 \in T_{new}$ and $s_{1-5} \in S_{new}$. $s_2.t_s$ has been determined as 7. Consider $s_3.t_s$. As highlighted in blue in Fig. 2, since $s_2.t_s = 7$ and s_2, $s_4 \in Com_{t_2}$, $s_4.t_s \leq 7 + \delta = 8$. Similarly, $s_5.t_s \leq 8 + \delta = 9$. Finally, ($s_3.t_s$) cannot exceed $9 + \delta = 10$. Besides, as highlighted in red, although $s_1.t_s$ is undetermined, it is at most 6 since $s_1.t_x = 6$. Similar to the limit imposed by s_2, s_1 restricts $s_3.t_s$ from exceeding 9 as highlighted in red in Fig. 2.

We can formalize the above restrictions as an undirected graph G = (V, E), in which each $s \in S_{new}$ corresponds to a vertex $v(s)$ and if any two sub-instances s_1, s_2 belongs to a combination, ($v(s_1)$, $v(s_2)$) $\in E$. For any $s' \in S_{new}$, if $v(s)$ and $v(s')$ are connected through a k-hop path, the constraints of s' on $s.t_s$ have the following two cases:

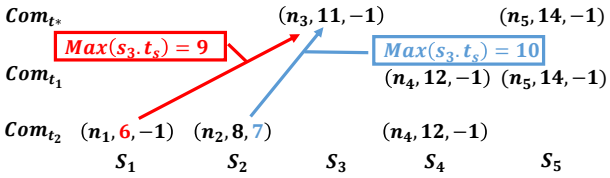


Fig. 2. The restrictions of s_1 and s_2 on $s_3.t_s$.

(a) $s'.t_s$ is determined before $s.t_s$. As highlighted in blue in Fig. 2, $s.t_s \in [s'.t_s - k\delta, s'.t_s + k\delta]$.

(b) Otherwise, we have two subcases: (1) $s'.t_x < s.t_x$. Since $s'.t_s \leq s'.t_x$, as highlighted in red in Fig. 2, $s.t_s \leq s'.t_x + k\delta$; (2) Otherwise, since $s.t_s \leq s.t_x$ and $s.t_x \leq s'.t_x$, we have $s.t_s \leq s'.t_x + k\delta$, indicating that s' does not limit $s.t_s$.

2) Secondly, to satisfy A1 (Causality Constraint), $s.t_s \leq s.t_x$.

To summarize above (1) and (2), when computing $s.t_s$ for a $s \in UD(Com_{t*})$, for each sub-instance $s' \in S_{new}$, we have $s.t_s \in [R_{min}(s'), R_{max}(s')]$:

$$R_{min}(s') = \begin{cases} Max\{1, s'.t_s - h(s', s)\delta\}, & If\ s'.t_s \neq -1 \\ 1 & Otherwise \end{cases}$$

$$R_{max}(s') = \begin{cases} Min\{s'.t_s + h(s', s)\delta, s.t_x\} & If\ s'.t_s \neq -1 \\ Min\{s'.t_x + h(s', s)\delta, s.t_x\} & \\ & If\ s'.t_s == -1\ \&\ s'.t_x < s.t_x \\ s.t_x & Otherwise \end{cases}$$

in which $h(s, s')$ is the minimum hop count from $v(s)$ to $v(s')$ in G. Finally, $s.t_s$ is the largest value that satisfies the restrictions of all other sub-instances in S_{new}:

$$s.t_s = Max\left\{ i | i \in \bigcap_{s' \in (S_{new}/\{s\})} [R_{min}(s'), R_{max}(s')] \right\}$$

The method above solves the subproblem of this substep optimally, and $Ts(Com_{t*})$ is maximized.

IV. Real-time Schedule Adjustment

AM-Agg computes a basic working schedule based on the predicted EH rates. When the actual EH rate drops below the estimated, some nodes may not have enough energy to become active in some predetermined active slots. To tackle this issue, we propose an online schedule adjustment scheme. The basic idea is that in case of energy insufficiency, a node will only become active during slots that could reduce more AoI. This scheme contains two steps:

(1) For each node n, divide node n's sub-instances into some priorities, with higher priority ones able to reduce more AoI.

(2) During the working duration, n will adjust its schedule according to its real-time energy status, in which higher-priority sub-instances are transmitted first.

Prioritize All the Sub-instances. Assuming there are p different priorities levels. Starting from the highest priority (Level p), the scheme selects the sub-instances for each priority level in order from highest to lowest priority (i.e., From p to 1), in which level i is selected as follows:

(1) Initialize an empty set D.

(2) Each time select a combination not in S from the combinations of all the targets, such that if this combination does not exist, the AVG_AoI would increase the most. Then, Add this combination to D, and set all sub-instances in this combination as i.

(3) Repeat (2) until level i account for $1/p$ of all sub-instances of all the nodes.

Through the method above, sub-instances that are more critical to AoI minimization have higher priority.

Real-time Schedule Adjustment. Nodes adjust their working schedule based on real-time energy status and priority level of sub-instances. Specifically, at the start of slot i, if node n has been scheduled to become active by AM-Agg, then:

(1) n estimates its available energy for the remaining working duration, which is denoted by $AE_n(i)$ as follows:

$$AE_n(i) = B_n(i) + PE_n(i, K)$$

in which $PE_n(i, K)$ stands for the real-time estimated value of n's total harvestable energy from slot i to K[4].

(2) n adjusts its schedule. The basic idea is to transmit more high-priority sub-instances while satisfying the energy constraints. Specially, n will transmit sub-instances with priority higher than b, which is minimized such that n's energy consumption within the remaining working duration does not exceed $AE_n(i)$:

$$b = \underset{j\in[1,p]}{ArgMin}\left\{AE_n(i) \geq \left|\bigcup_{k\in[j,p]} Lev(i,k)\right| \times E_c(n)\right\}$$

in which $Lev(i, k)$ denotes the set of sub-instances with priority level k and sending-time in $[i, K]$. If the priority level of n's sub-instance at slot i is no less than b, n will transmit this sub-instance. Otherwise, just stay idle at slot i.

Through the schedule adjustment scheme, when nodes' energy level is lower than expected, those high-priority combinations that can reduce more AoI will still be updated, thus ensuring the AoI performance. This also works when nodes have different EH rates, as high-priority sub-instances are transmitted by nodes with both high and low energy. This scheme is evaluated in Sec. V-B.

V. Evaluation

A. Simulation

This section evaluates the proposed algorithm through simulations. During each simulation, firstly, we use Pro-Energy [35] to predict the EH rate of each node. Then, the schedule of each node is computed by AM-Agg based on the predicted EH rate. During the working duration, each node adjusts its schedule using the scheme of Sec. IV.

Simulation Parameters. In the network, there are $|N| = 100$ nodes deployed to cover $|T| = 80$ targets. The average value of $|Cover_node(t)|$ and $Agg_num(t)$ is 7.5 and 2.5 respectively. The length of the working duration is $K = 1200$ and $\delta = 4$.

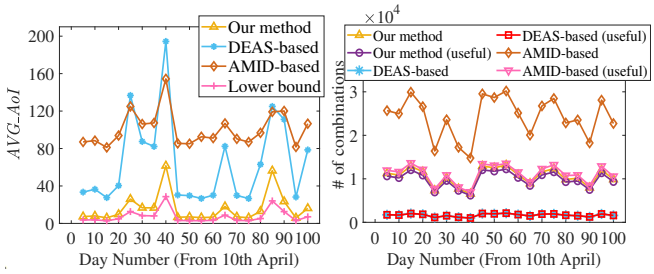
As for energy parameters, the simulation applies the energy trace from an indoor solar energy dataset [40] for $EH_n(i)$. The length of each slot is one second. Same as [41], to model heterogeneous EH rates among nodes, a random noise $\pm$ 50% is added to the $EH_n(i)$. $E_c(n)$ is computed according to CC2500 datasheet [42], and each node needs 30 slots on average to harvest the energy required for one active slot.

Evaluation Metrics. Apart from the AoI metric (i.e., *AVG_AoI*), to explore the energy efficiency, we discuss the number of all combinations and *useful* combinations, in which a combination Com_t is considered to be *useful* if the AoI of t is reduced when Com_t is updated to BS.

Baselines. We compare our protocol with the lower bound of *AVG_AoI*, which is computed by the following equation:

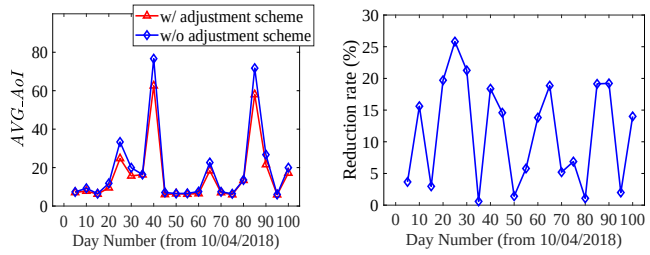
$$Low_AVG_AoI = \sum_{t\in T} \frac{a_1[(a_1-1)p(t) + a_1 + a_2 + 1]}{2K|T|},$$

[4]The energy estimation method here is the same as that for predicting $EH_n(i)$ as the input of AM-Agg (i.e., Pro-Energy [35]). Note that as i increases, the time scale to be predicted decreases, thus improving the prediction accuracy. e.g., In our simulation, when $i = 0.1K$ and $0.9K$, the average prediction errors of $PE_n(i, K)$ are 18.8% and 11.5% respectively.



(a) *AVG_AoI*. (b) Combination number.

Fig. 3. Average performance of all working durations.



(a) The performance with and without adjustment scheme. (b) The reduction rate of average AoI after using the adjustment scheme.

Fig. 4. The impact of schedule adjustment scheme.

in which $a_1 = \lfloor \frac{K}{p(t)+1} \rfloor$, $a_2 = K \ mod \ (p(t)+1)$ and $p(t) = \sum_{n\in Cover_node(t)} \frac{B_n(0)+\sum_{i=1}^{K} EH_n(i)}{E_c(n)\times Agg_num(t)}$.

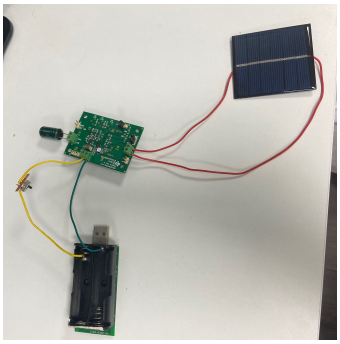
So far, AoI-Agg has not been studied. The most similar recent works are AIMD [22] and DEAS [31], which studied the problem of AoI minimization for individual-source data collection (see Sec. I) and minimum-latency data aggregation, respectively. Hence, we use them as baselines for comparison. Note that since the data of each target is updated only once by DEAS, it is iterated starting from the first slot to the end of the working duration in our evaluation. Also, following assumptions in [22] and [31], when evaluating AMID and DEAS, we assume perfect knowledge of $EH_n(i)$.

Simulation Results. This part shows the result of the simulation conducted on the energy data starting from 10/04/2018 to 19/07/2018, in which each data point represents the average of five consecutive days (i.e., 180 working durations). As shown by Fig. 3(a), our protocol outperforms two baselines, in which its performance is the most close to the lower bound (About 1.9× lower bound on average), and the average AoI is reduced by 75% and 84% compared to the DEAS and AMID respectively.

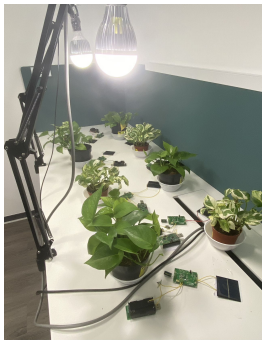
Comparison with DEAS: In Phase I and Substep 1 of Phase II, AM-Agg algorithm computes the schedule with the objective of minimizing the data reception time. This improves the frequency of data updates. This is shown in Fig. 3(b): our method has more combinations than DEAS. In comparison, DEAS does not consider the energy level of nodes, resulting in long data update intervals.

Comparison with AMID: In Substep 1 of Phase II, for a combination Com_t, the sending-time of each sub-instance in Com_t is made as close as possible to $Ar(Com_t)$. Also, through Substep 2 of Phase II, the timestamp of Com_t is also maximized. All of above not only facilitate fast data updates but also maximizes the freshness of the updated data, making more than 97% of combinations useful (Shown in Fig. 3(b)). In contrast, AMID could only ensure rapid data transmission without considering the data aggregation, leading to uncertainty in the freshness of its updated data.

In summary, different parts of AM-Agg play distinct roles: the calculation of FC_{min} of Phase I and the minimization of $Ar(Com_t)$ in Substep 1 of Phase II ensure the harvested energy is *fully* utilized,



(a) The EH TelosB node.



(b) The testbed.

Fig. 5. Experimental hardware.

while the maximization of $Ts(Com_t)$ in Substep 2 of Phase II ensures the energy is *wisely* utilized. These improve energy efficiency, enabling our algorithm to achieve high performance in terms of AoI.

B. The Impact of Schedule Adjustment Scheme

We evaluate the impact of schedule adjustment scheme by comparing the performance with and without applying it. As shown in Fig. 4, the scheme reduces the average AoI by 11.5% on average compared to using the AM-Agg alone. As we observe, the difference in the proportion of combinations that could not be updated due to energy constraints, with and without the scheme, is only 1.52% on average. Thus, the main reason for the effectiveness is that the scheme prioritizes combinations that could reduce more AoI when energy is insufficient. Also, as shown in Fig. 4(b), the scheme is more effective when AoI is large. This is because there is relatively less harvestable energy under this condition, leading to more combinations not being updated, thus increasing the optimization space for the scheme.

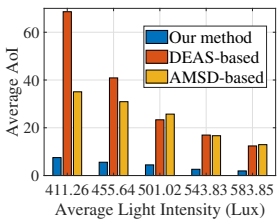
C. Testbed evaluation

To verify the simulation results, we also conduct experimental studies on real-world EH testbeds.

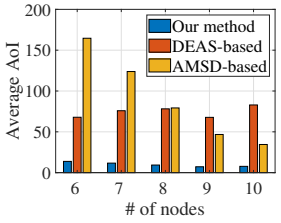
Experimental Settings. First, we uniformly deploy 10 EH TelosB nodes on the 120cm×70cm testbed (Fig. 5(a)), in which each node is equipped with a 6cm×5cm solar panel and TI BQ25504-EVM674 EH evaluation module [38]. We use two LEDs whose light intensity could be adjusted among 10 discrete levels as the energy source. As shown in Fig. 5(b), an indoor agriculture scenario is simulated, in which some plants are placed on the testbed such that some nodes are in shadow while other nodes are exposed directly to the light.

The whole working duration contains 1000 timeslots and the length of one slot is one second. During each execution, the light intensity is adjusted every 100 slots. We assume there are 8 targets to be monitored, and the average value of $|\Gamma(n)|$ and $Agg_num(t)$ is 6 and 2.5. The size of each sub-instance is 10 bytes, which contains the temperature data (4 bytes), the humidity data (4 bytes) and the sub-timestamp (2 bytes). During each active slot, the node will transmit its sub-instance two times with a transmission power of -11dBM. The default average light intensity at each node is set to be 411.26 Lux, in which it takes 25 idle slots on average for each node to harvest the energy needed for an active slot.

Experimental Results. We evaluate the performance on AoI in terms the ambient light condition and nodes number. As illustrated by Fig. 6(a), as the ambient light intensity increases, nodes can harvest more energy, which increases the update frequency of all the targets. Thus, the performance of all protocols increases. As shown by Fig. 6(b), the performance of our method and AMID increases with the number of nodes. This is because as the nodes number grows, a target could be covered by more nodes and the sensed data is updated more



(a) *AVG_AoI* vs. ambient light condition.



(b) *AVG_AoI* vs. nodes number.

Fig. 6. Results of testbed experiments.

frequently. As for DEAS, since the average value of $|\Gamma(n)|$ remains unchanged, the number of nodes required to cover all the targets shows no obvious change as nodes number increases. Under this condition, as DEAS only schedules the nodes in a full-coverage to become active, nodes number has little influence on its performance.

It can be seen that our method reduces the average AoI by 86% and 89% on average compared with DEAS and AMID respectively. This aligns well with our simulation results.

VI. Discussion

For the scenario when targets have different weights. In Substep 1 of Phase II of AM-Agg, $Ar(Com_t)$s for all targets in T_{new} are minimized simultaneously. This is also similar for Substep 2. Thus, AM-Agg tends to reduce the AoI of multiple targets at the same time, rather than minimizing the AoI of some targets at the expense of increasing the AoI of others. Hence, our method is still effective for scenarios where target weights are different. To validate this, we evaluate our method based on the setup of Sec. V-A, and the weight of targets varies between 0.1 and 3 (with an average of 1). The results show that the weighted *AVG_AoI* of our method is only 1.75× lower bound on average. This is similar to the results in Sec. V-A.

About transmission conflict. In real-world EH-IoTs, nodes generally require a relatively long time for EH before they can operate. For example, according to [40], on average, a solar-powered node needs about 120 seconds to harvest enough energy to send 200 bytes of data, which could be transmitted in just 8 milliseconds. Thus, the slot length in EH-IoTs is much longer compared with conventional communication networks, usually ranging from several seconds to minutes [44]–[48]. For an EH-IoT consisting of hundreds of nodes, within each slot, we can even allocate a TDMA sub-slot to each node, thereby avoiding the transmission conflicts. Therefore, our study does not need to take transmission conflicts into account.

VII. Conclusion

In this paper, we study the AoI minimization problem for the scenario of data aggregation in EH-IoTs for the first time. We propose both an energy-adaptive scheduling algorithm and a real-time schedule adjustment scheme to solve the problem. Simulations and testbed experiments verify the high performance of our method in terms of AoI and energy efficiency.

Acknowledgement

We would like to thank the anonymous reviewers for their helpful feedback. This work is supported by the National Natural Science Foundation of China (Grants No. 61632010, U22A2025), the Hong Kong General Research Fund (Grant No. 11208522), the National Natural Science Foundation of China (Grant No. 62472086), the Basic Research Program of Jiangsu (Grants No. BK20243042, BG2024010), and the Start-up Research Fund of Southeast University (Grant No. RF1028624005). This research receives support from China Scholarship Council.

REFERENCES

[1] KS. Adu-Manu, N. Adam, C. Tapparello, et al, "Energy-harvesting wireless sensor networks (EH-WSNs): A review," in *ACM Transactions on Sensor Networks*, 14(2): 1-50, 2018.

[2] G. Zhang, W. Zhang, et al, "Energy-Delay Tradeoff for Dynamic Offloading in Mobile-Edge Computing System With Energy Harvesting Devices," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4642-4655, 2018.

[3] Z. Cai, Q. Chen, T. Shi, T. Zhu, K. Chen and Y. Li, "Battery-Free Wireless Sensor Networks: A Comprehensive Survey," in *IEEE Internet of Things Journal*, vol. 10, no. 6, pp. 5543-5570, 2023.

[4] S. Lambert, W. Van Heddeghem, W. Vereecken, B. Lannoo, D. Colle, and M. Pickavet, "Worldwide electricity consumption of communication networks," in *Opt. Exp.*, vol. 20, no. 26, pp. B513–B524, 2012.

[5] S. Kaul, R. Yates and M. Gruteser, "Real-time status: How often should one update?," in *INFOCOM* 2012, pp. 2731-2735.

[6] AM. Bedewy, Y. Sun, S. Kompella and NB. Shroff, "Age-optimal sampling and transmission scheduling in multi-source systems," in *Mobihoc* 2019, pp. 121–130.

[7] R. Talak, S. Karaman, and E. H. Modiano, "Optimizing information freshness in wireless networks under general interference constraints," in *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 15–28, 2020.

[8] C. Li, S. Li, Y. Chen, Y. Thomas Hou and W. Lou, "AoI Scheduling with Maximum Thresholds," in *INFOCOM* 2020, pp. 436-445.

[9] B. T. Bacinoglu, Y. Sun, et al, "Achieving the Age-Energy Tradeoff with a Finite-Battery Energy Harvesting Source," in *ISIT* 2018, pp. 876-880.

[10] A. Arafa, J. Yang, et al, "Age-Minimal Transmission for Energy Harvesting Sensors With Finite Batteries: Online Policies," in *IEEE Transactions on Information Theory*, vol. 66, no. 1, pp. 534-556, 2020.

[11] O. Ozel, "Timely Status Updating Through Intermittent Sensing and Transmission," in *ISIT* 2020, pp. 1788-1793.

[12] S. Feng and J. Yang, "Age of Information Minimization for an Energy Harvesting Source With Updating Erasures: Without and With Feedback," in *IEEE Transactions on Communications*, vol. 69, no. 8, pp. 5091-5105, 2021.

[13] Y. Liu, Z. Chang, G. Min and S. Mao, "Average Age of Information in Wireless Powered Mobile Edge Computing System," in *IEEE Wireless Communications Letters*, vol. 11, no. 8, pp. 1585-1589, 2022.

[14] F. Chiariotti et al., "Query Age of Information: Freshness in Pull-Based Communication," in *IEEE Transactions on Communications*, vol. 70, no. 3, pp. 1606-1622, 2022.

[15] A. Jaiswal and A. Chattopadhyay, "Minimization of Age-of-Information in Remote Sensing with Energy Harvesting," in *ISIT* 2021, pp. 3249-3254.

[16] M. A. Abd-Elmagid and H. S. Dhillon, "Age of Information in Multi-source Updating Systems Powered by Energy Harvesting," in *IEEE Journal on Selected Areas in Information Theory*, vol. 3, no. 1, pp. 98-112, 2022.

[17] Y. Dong, P. Fan and K. B. Letaief, "Energy Harvesting Powered Sensing in IoT: Timeliness Versus Distortion," in *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 10897-10911, 2020.

[18] N. Nouri, D. Ardan and MM. Feghhi, "Age of information-reliability trade-offs in energy harvesting sensor networks," arXiv preprint arXiv:2008.00987, 2020.

[19] Z. Zhou, C. Fu, C. J. Xue et al., "Energy-Constrained Data Freshness Optimization in Self-Powered Networked Embedded Systems," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2293-2306, 2020.

[20] Z. Fang, J. Wang, et al, "Age of Information in Energy Harvesting Aided Massive Multiple Access Networks," in *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 5, pp. 1441-1456, 2022.

[21] T. Zhu, J. Li, et al., "AoI Minimization Data Collection Scheduling for Battery-Free Wireless Sensor Networks," in *IEEE Transactions on Mobile Computing*, vol. 22, no. 3, pp. 1343-1355, 2023.

[22] B. Yao, H. Gao, et al, "Maximum AoI Minimization for Target Monitoring in Battery-Free Wireless Sensor Networks," in *IEEE Transactions on Mobile Computing*, vol. 22, no. 8, pp. 4754-4772, 2023.

[23] M. Hatami, M. Leinonen, Z. Chen, N. Pappas and M. Codreanu, "On-Demand AoI Minimization in Resource-Constrained Cache-Enabled IoT Networks With Energy Harvesting Sensors," in *IEEE Transactions on Communications*, vol. 70, no. 11, pp. 7446-7463, 2022.

[24] T. He, K. -W. Chin, Z. Zhang, et al., "Optimizing Information Freshness in RF-Powered Multi-Hop Wireless Networks," in *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 7135-7147, 2022.

[25] Q. Chen, Z. Cai, L. Cheng, F. Wang and H. Gao, "Joint Near-Optimal Age-based Data Transmission and Energy Replenishment Scheduling at Wireless-Powered Network Edge," in *INFOCOM* 2022, pp. 770-779.

[26] W. Jin, J. Sun, K. Chi and S. Zhang, "Deep reinforcement learning based scheduling for minimizing age of information in wireless powered sensor networks," in *Computer Communications*, vol. 191, pp. 1-10, 2022.

[27] L. Lin, L. Ju, C. J. Xue, et al., "Work or Sleep: Freshness-Aware Energy Scheduling for Wireless Powered Communication Networks with Interference Consideration," in *DAC* 2023, pp. 1-6.

[28] Z. Chen, N. Pappas, E. Björnson and E. G. Larsson, "Age of Information in a Multiple Access Channel with Heterogeneous Traffic and an Energy Harvesting Node," in *INFOCOM WKSHPS* 2019, pp. 662-667.

[29] S. Leng and A. Yener, "Age of Information Minimization for an Energy Harvesting Cognitive Radio," in *IEEE Transactions on Cognitive Communications and Networking*, vol. 5, no. 2, pp. 427-439, 2019.

[30] M. Xie, X. Jia, J. Yin, Q. Wang and M. Zhou, "Age of Information for Partial Earliest Relay Aided Short Packet Status Update With Energy Harvesting," in *IEEE Transactions on Wireless Communications*, vol. 23, no. 1, pp. 699-719, 2024.

[31] K. Chen et al, "Distributed Energy-Adaptive Aggregation Scheduling with Coverage Guarantee For Battery-Free Wireless Sensor Networks," in *INFOCOM* 2019, pp. 1018-1026.

[32] Q. Chen, Z. Cai, L. Cheng, et al., "Energy-collision-aware Minimum Latency Aggregation Scheduling for Energy-harvesting Sensor Networks," in *ACM Transactions on Sensor Networks*, vol. 17(4), no. 40, pp. 1-34.

[33] J. Li et al., "Age of Aggregated Information: Timely Status Update with Over-The-Air Computation," in *GLOBECOM* 2020, pp. 1-6.

[34] H. Pan, S. C. Liew, et al, "Coding of Multi-Source Information Streams With Age of Information Requirements," in *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 5, pp. 1427-1440, 2021.

[35] A. Cammarano, C. Petrioli and D. Spenza, "Pro-Energy: A novel energy prediction model for solar and wind energy-harvesting wireless sensor networks," in *MASS* 2012, pp. 75-83.

[36] C. Xu, Q. Xu, J. Wang, et al., "AoI-centric Task Scheduling for Autonomous Driving Systems," in *INFOCOM* 2022, pp. 1019-1028.

[37] X. Wang, A. Walden, et al., "Strategies for Sensor Data Aggregation in Support of Emergency Response," in *MILCOM* 2014, pp. 1120-1126.

[38] "Ultra Low Power Boost Converter Charger Energy Harvester," https://www.ti.com/lit/ug/sluu654a/sluu654a.pdf?ts=1673873261236&ref_url=https

[39] Y. Sun, E. Uysal-Biyikoglu, R. Yates, et al, "Update or wait: How to keep your data fresh," in *INFOCOM* 2016, pp. 1-9.

[40] L. Sigrist, A. Gomez and L. Thiele, "Dataset: Tracing indoor solar harvesting," in *DATA* 2019, pp. 47-50.

[41] S. Yang, Y. Tahir, P. -y. Chen, A. Marshall and J. McCann, "Distributed optimization in energy harvesting sensor networks with dynamic in-network data processing," in *INFOCOM* 2016, pp. 1-9.

[42] CC2500 Data Sheet. https://www.ti.com/product/CC2500

[43] M. Cardei and DZ. Du, "Improving wireless sensor network lifetime through power aware organization," in *Wireless networks*, vol. 11, pp. 333–340, 2005.

[44] X. Ren, W. Liang and W. Xu, "Data Collection Maximization in Renewable Sensor Networks via Time-Slot Scheduling," in *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 1870-1883, 2015.

[45] A. Yadav, M. Goonewardena, W. Ajib, O. A. Dobre and H. Elbiaze, "Energy Management for Energy Harvesting Wireless Sensors With Adaptive Retransmission," in *IEEE Transactions on Communications*, vol. 65, no. 12, pp. 5487-5498, 2017.

[46] C. Yang, K. -W. Chin, Y. Liu, J. Zhang and T. He, "Robust Targets Coverage for Energy Harvesting Wireless Sensor Networks," in *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 5884-5892, 2019.

[47] F. Francesco, B. Bharathan, A. Yuvraj and K. G. Rajesh, "ACES: Automatic configuration of energy harvesting sensors with reinforcement learning," in *ACM Trans. Sensor Netw.*, vol. 16, no. 4, pp. 36:1–36:31, 2020.

[48] R. Mohammadi and Z. Shirmohammadi, "RLS2: An energy ef1309 ficient reinforcement learning-based sleep scheduling for energy harvesting WBANs," in *Computer Networks*, vol. 229, no. 109781, 2023.