DAC Tutorial:

Introduction to Foundation AI Model and Its EDA Applications

Speaker:

Prof. Ang Li, University of Maryland, College Park

Dr. Wei Wen, Meta

Prof. Zhiyao Xie, HKUST

Host:

Prof. Xiaoxuan Yang, University of Virginia

THE CHIPS TO SYSTEMS CONFERENCE

UNIVERSITY OF MARYLAND

Meta

香港科技大學 THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

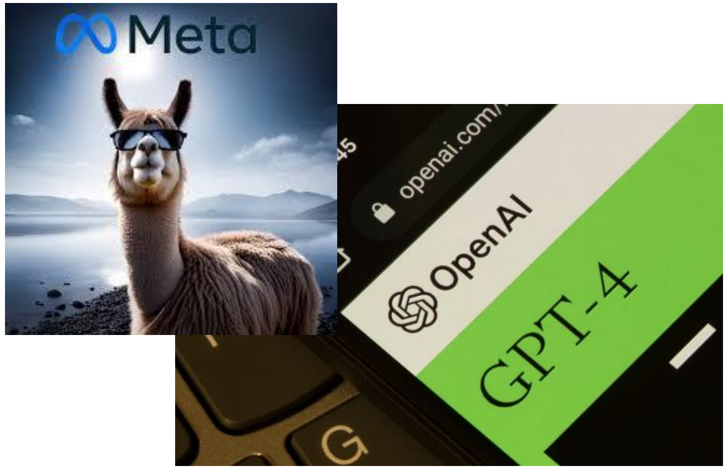
SPONSORED BY

CEDA IEEE Council on Electronic Design Automation

sigda

Opportunities from Foundation Models

- Emergence of **large foundation models** in many fields
  - Unprecedented ability to *understand*, *predict*, and *generate* content



Language model: GPT, Llama

Q: Image (A potato king)
A:

Image model: DALL-E

Q: Video (A family of monsters)
A:

Video model: Sora

Overview of This Tutorial

A 3-hour tutorial about **foundation AI models and EDA applications**

1. **Basic** Large Language Model (LLM) Knowledge
   - **Ang Li (University of Maryland)**, 1-hour session
2. **Multimodal** Foundation Model + **Efficiency** of Foundation Model
   - **Wei Wen (Meta)**, 1-hour session
3. Using Foundation Models in **EDA Applications**
   - **Zhiyao Xie (HKUST)**, 1-hour session

Basic Large Language Model (LLM) Techniques

Ang Li, Assistant Professor, University of Maryland

Duration: ~1 hour

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA sigda

Outline of Session 1

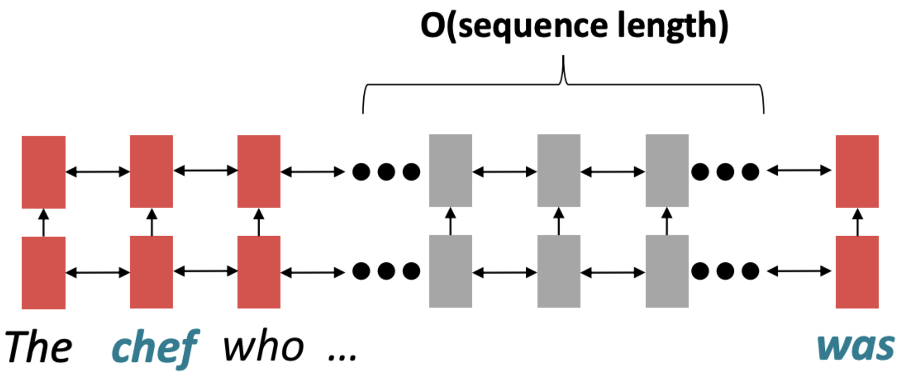
- Attention Models and Transformers
- Large Language Model Training
- Large Language Model Inference

Outline of Session 1

- Attention Models and Transformers
- Large Language Model Training
- Large Language Model Inference

Issue with recurrent models

- Recurrent models (e.g., LSTM, GRU) are unrolled from left to right
  - Word pairs will have linear interaction distance

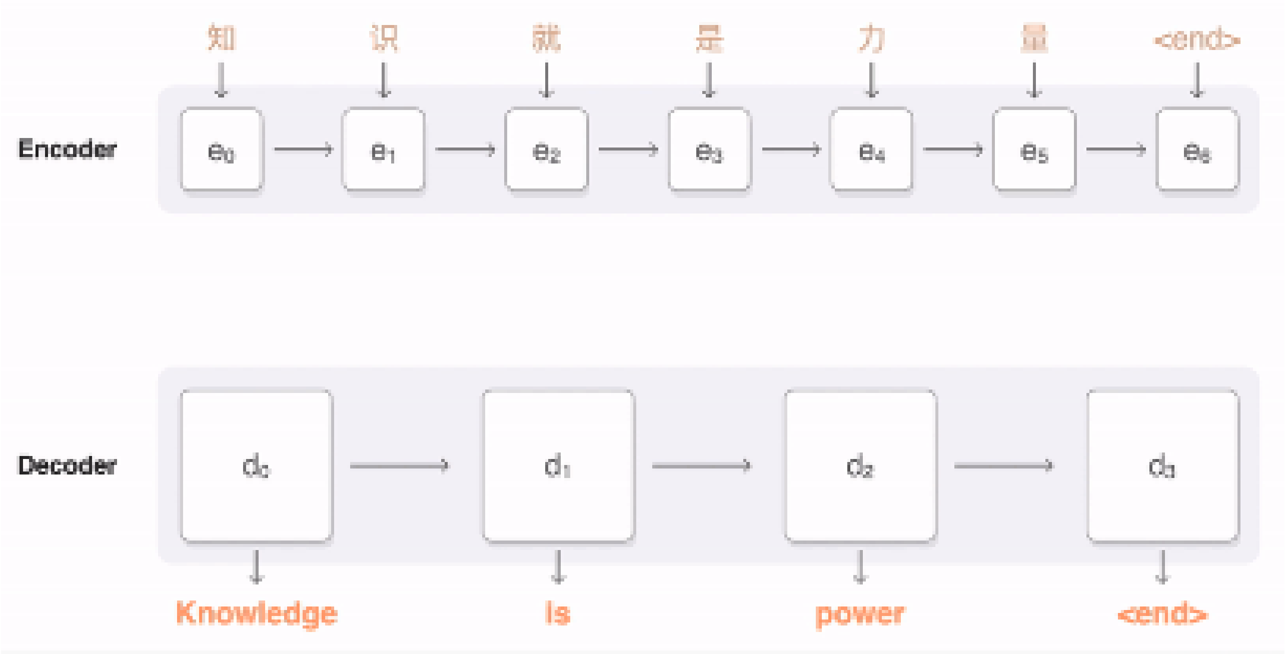


Problems:

- Hard to learn long-distance dependencies
  - Gradient vanishing issue
- Hard to parallelization
  - Forward and backward passes have *O(sequence length)* unparallelizable operations

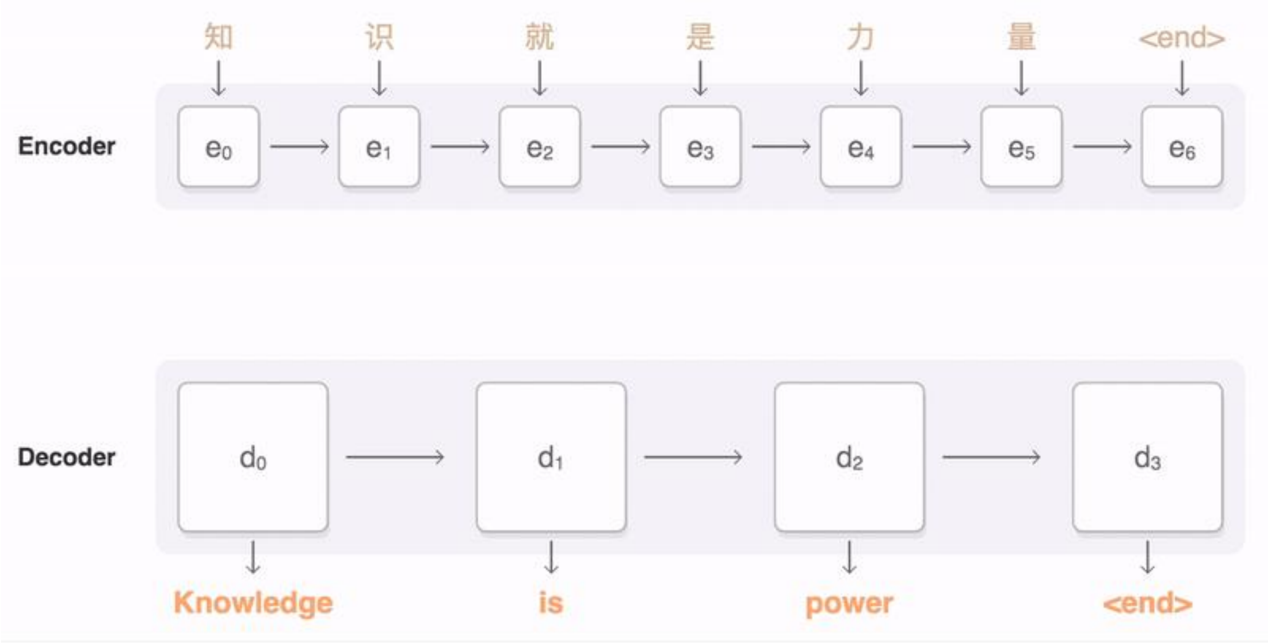
Problems with classic Seq2Seq models

- Traditional encoder-decoder systems suffer from information bottleneck:
  - **Last hidden state** need to capture **all the information** about the source sentence



Solution: attention mechanism

- Attention mechanism provides a solution to the problem
- Core idea: at each decoding step, **focus on different part** of the source sequence.



How to compute attention?

- Suppose we have encoder hidden states $e_1, \dots e_N \in \mathbb{R}^h$,
  step *t* decoder hidden state $d_t \in \mathbb{R}^h$
- At decoding step *t*,

1. Compute the attention score

$$s^t = [d_t^T e_1, \dots d_t^T e_N] \in \mathbb{R}^N$$

2. Apply softmax to get the attention distribution over source tokens

$$w^t = softmax(s^t) \in \mathbb{R}^N$$

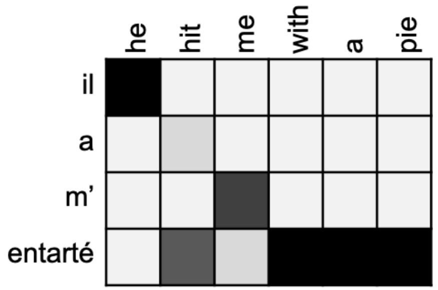
3. Compute weighted sum over the encoder hidden states

$$a_t = \sum_{i=1}^{N} w_i^t e_i \in \mathbb{R}^h$$

4. Concatenate a_t with d_t ,and feed $[a_t; d_t] \in \mathbb{R}^{2h}$ to the decoder

Why attention is so powerful?

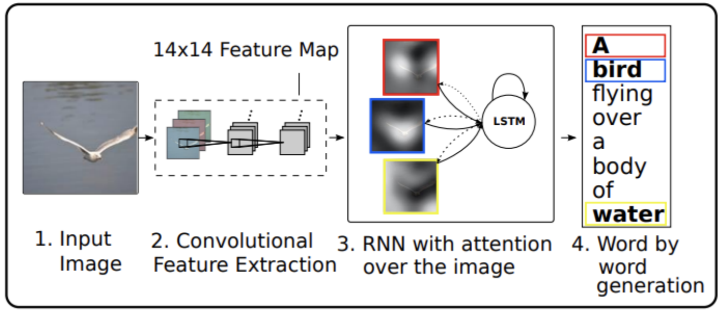
- Attention can significantly improve neural machine translation (NMT) performance
  - Allow decoder to focus on different parts of the source
  - Solves the information bottleneck problem
- Attention helps with the vanishing gradient issue
  - Provides shortcut to early source tokens
- Attention provides interpretability
  - Implicitly learn soft alignment between source and target sequence
  - Check the attention distribution for each output token

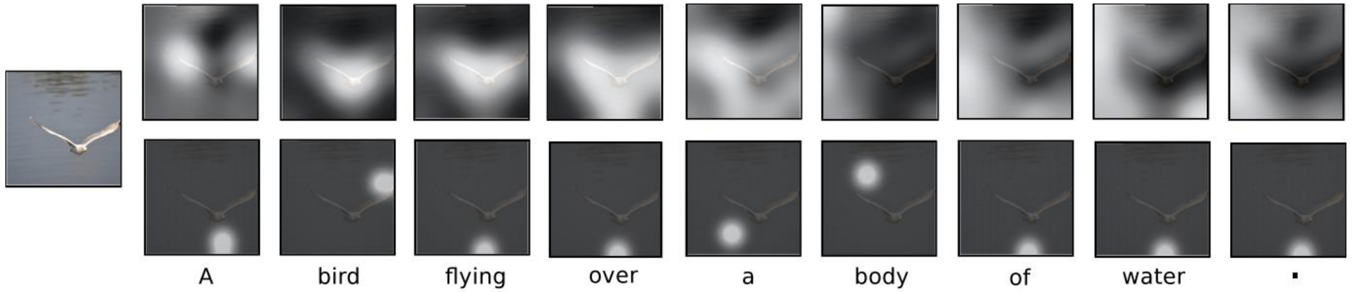


D. Bahdanau, K. Cho, and Y. Bengio. "Neural machine translation by jointly learning to align and translate." (2014)

Attention as a general technique

- Attention is also used in computer vision:
  - Attend to different parts on input image when generating caption



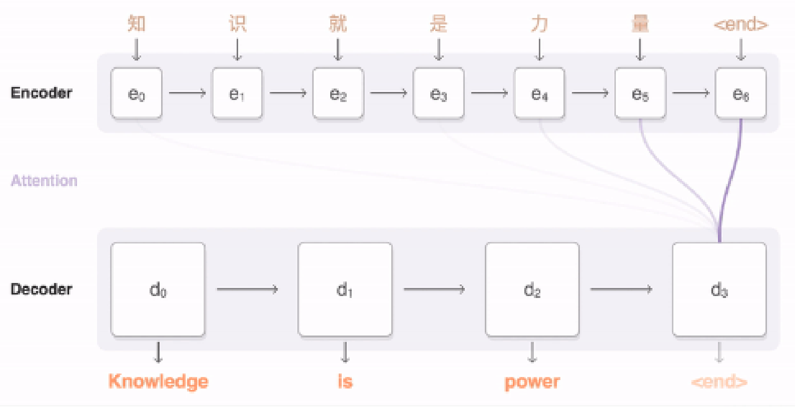


- Attention can also be a basic building block for sequence modeling
  - New sequence models: Transformers, BERT, GPT etc.

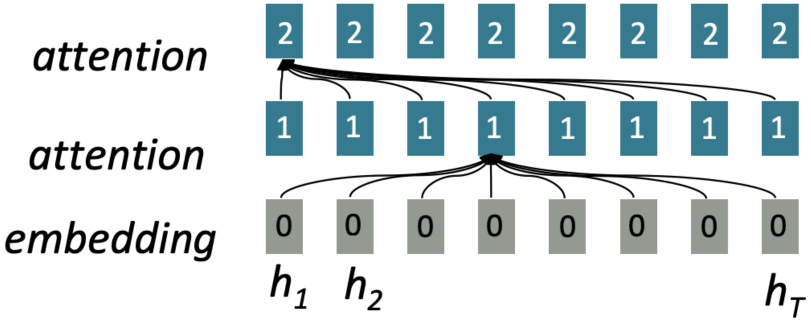
Xu, Kelvin, et al."Show, attend and tell: neural image caption generation with visual attention" ICML 2015

Replace recurrent with self-attention

- Remember attention is introduced in Seq2Seq systems to attend different parts of source sentence



- Self-attention: apply attention within a single sentence
  - All words attend to all words in previous layer (most arrows are omitted)

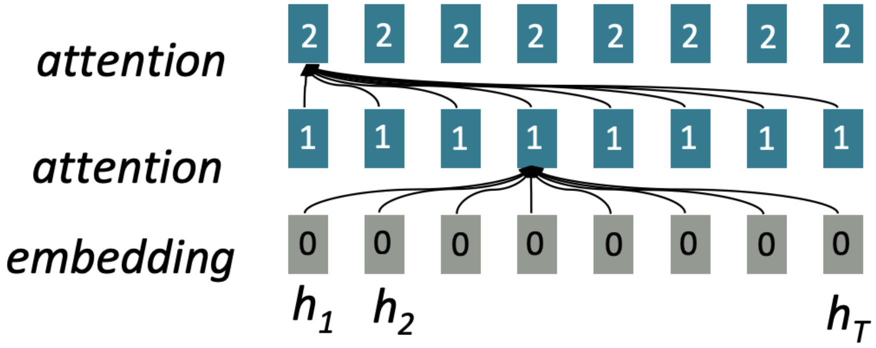


Credit: Stanford cs224n

Self-attention computation

- To compute attention we need queries, keys, and values:
  - Queries: $q_1, q_2, \ldots q_T$. Each $q_i \in \mathbb{R}^d$
  - Keys: $k_1, k_2, \ldots k_T$. Each $k_i \in \mathbb{R}^d$
  - Values: $v_1, v_2, \ldots v_T$. Each $v_i \in \mathbb{R}^d$
- In self-attention, the queries, keys and values come from the same source
  - $k_i = Kx_i, \; q_i = Qx_i, \; v_i = Vx_i$

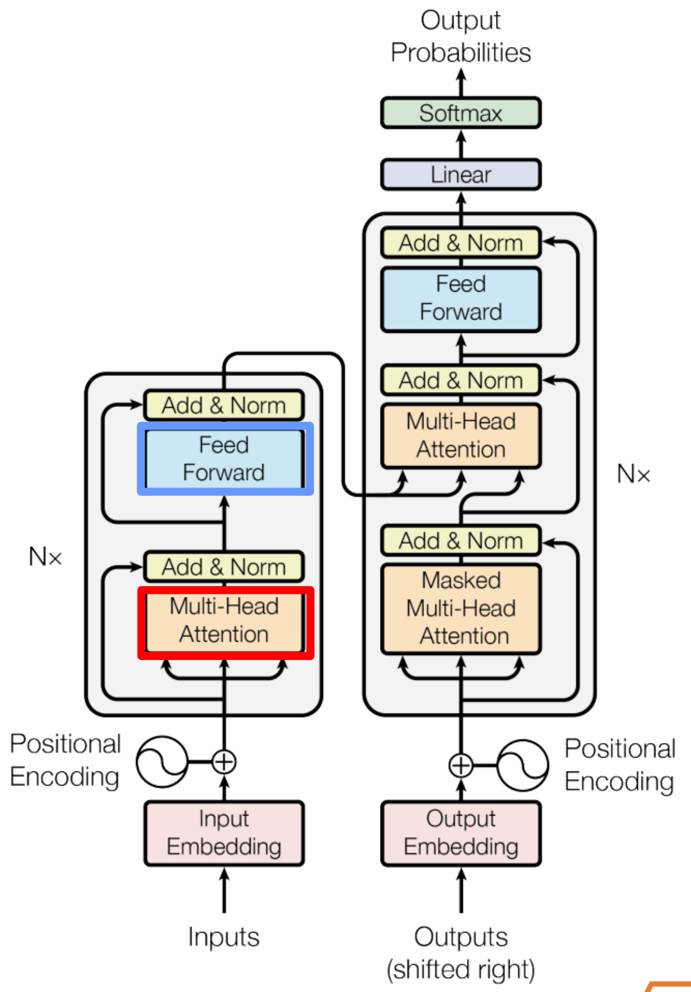
  where $K, Q, V \in \mathbb{R}^{d \times d}$ are linear transformation used for all x_i
- Self-attention generate new representations as follows:
  - score: $s_{ij} = q_i^T k_j$, $\;$ *attention:* $a_{ij} = \dfrac{\exp(s_{ij})}{\sum_{j'} \exp(s_{ij'})}$, *output*$_i = \sum_j a_{ij} v_j$



Credit: Stanford cs224n

Transformer

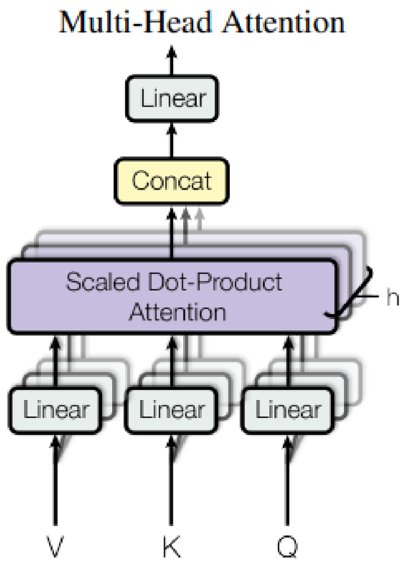
- Transformer structure:
  - Two parts: encoder & decoder (Seq2Seq model)
  - Basic block: self-attention + feed-forward
  - Stacked multiple blocks
  - Bunch of fixes/tricks



Vaswani, Ashish, et al. “Attention is all you need” Neurips 2017

Multi-head self-attention

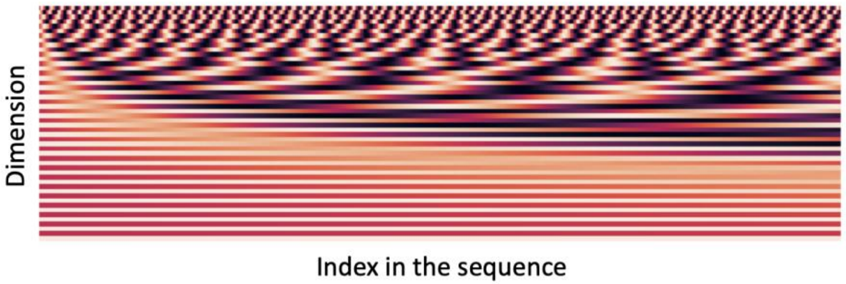
- Previously for each word i, we compute (**one**) attention over the words:
  - $k_i = Kx_i,\ q_i = Qx_i,\ v_i = Vx_i$ where $K, Q, V \in \mathbb{R}^{d\times d}$
  - score: $s_{ij} = q_i^T k_j$, attention: $a_{ij} = \frac{\exp(s_{ij})}{\sum_{j'} \exp(s_{ij'})}$, output$_i = \sum_j a_{ij} v_j$
- What if we want multiple attentions for each word?
  - We can define multiple attention "heads" by multiple K, Q, V matrices
  - Each head will look at different things and combine values differently!
- Define $K^l, Q^l, V^l \in \mathbb{R}^{d\times\frac{d}{h}}$, where h is the number of attention heads
  - For each head l: $k_i^l = K^l x_i,\ q_i^l = Q^l x_i,\ v_i^l = V^l x_i$
  - Use $k_i^l, q_i^l, v_i^l \in \mathbb{R}^{\frac{d}{h}}$ to compute score, attention and output$_i^l \in \mathbb{R}^{\frac{d}{h}}$
  - Combine all attention head outputs: output$_i = W_o[\text{output}_i^1; \dots; \text{output}_i^h]$ where $W_o \in \mathbb{R}^{d\times d}$



Encode sequence order

- Self-attention operation doesn't consider **the order information**
- Simple fix: we can represent the **sequence index** as a **vector**
  - Define positional embedding $p_i \in \mathbb{R}^d$, for $i \in \{1, 2, \ldots, T\}$
- Suppose $e_i \in \mathbb{R}^d$, for $i \in \{1,2,\ldots,T\}$ are the word embeddings, then we can add the positional embedding at layer 0: $x_i^0 = e_i + p_i$
- Options:
  - Sinusoidal position embedding:

$$p_i = \begin{pmatrix} \sin(i/10000^{2*1/d}) \\ \cos(i/10000^{2*1/d}) \\ \vdots \\ \sin(i/10000^{2*\frac{d}{2}/d}) \\ \cos(i/10000^{2*\frac{d}{2}/d}) \end{pmatrix}$$



  - Learned position embedding:

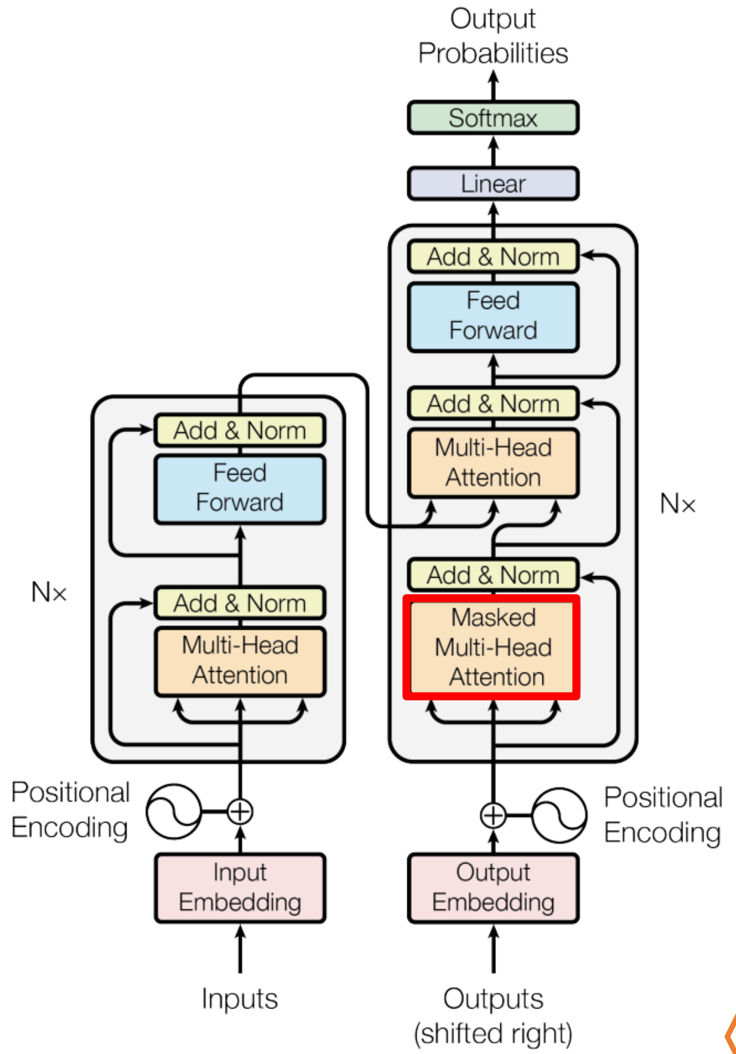
Just make all p_i as learnable parameters

Transformer decoder: self-attention

- To use self-attention in decoders, we need to ensure the decoder **cannot peek the future**
- Simple fix: we can mask the attention to future words by setting attention score as $-\infty$:

$$s_{ij} = \begin{cases} q_i^T k_j, & j < i \\ -\infty, & j \geq i \end{cases}$$

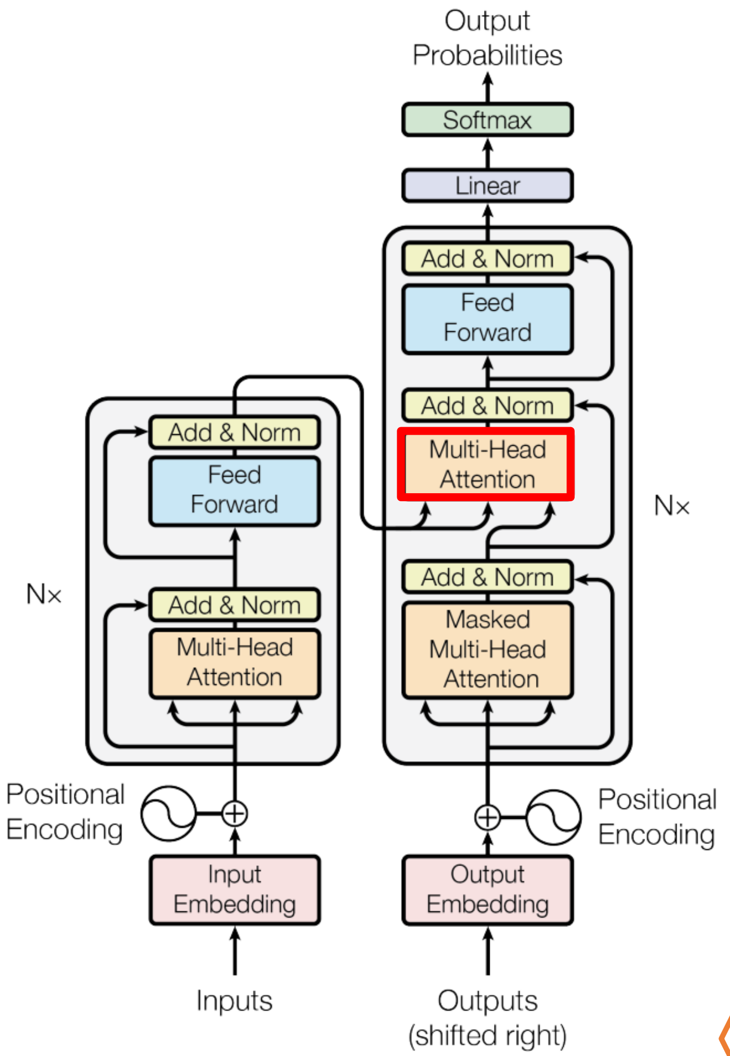
	[start]	The	boy	want
[start]	$-\infty$	$-\infty$	$-\infty$	$-\infty$
The		$-\infty$	$-\infty$	$-\infty$
boy			$-\infty$	$-\infty$
want				$-\infty$



Transformer decoder: encoder-attention

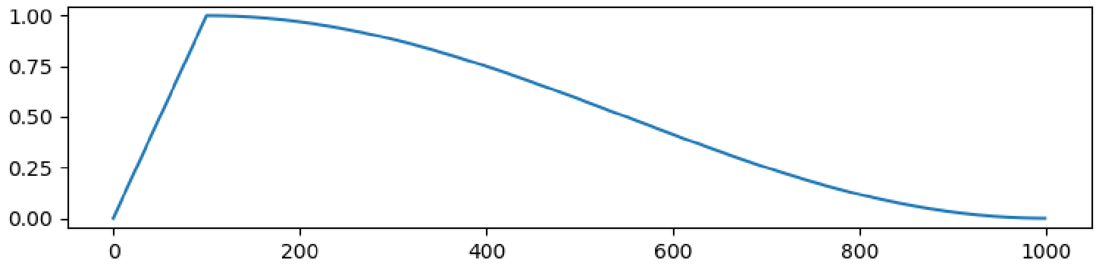
- In self-attention, keys, queries and values come from the same source
- However, on the decoder side, besides self-attention we also want to attend the states from encoder (Seq2Seq model)
- Simple fix: construct keys and values using encoder states
  - Define $x_1, \dots x_T \in \mathbb{R}^d$ as the output vectors from the **encoder**
  - Define $h_1, \dots h_N \in \mathbb{R}^d$ as the input vectors from the **decoder**
  - Compute key, value, query by:

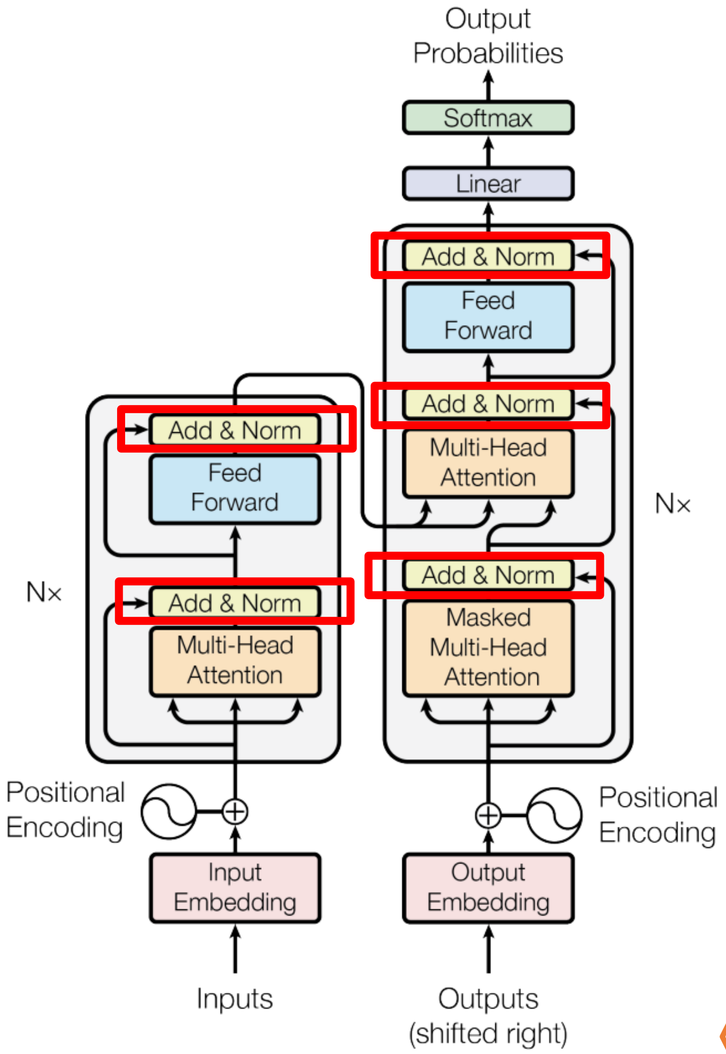
$$k_i = Kx_i, v_i = Vx_i, q_i = Qh_i$$



Other tricks in Transformer

- Residual connection and layer normalization:
  - Add after multi-head attention and feed-forward modules
  - Help models train faster
- Learning rate schedule:
  - warm-up stage: learning rate first increase then decrease
  - Converge to better sub-optimal

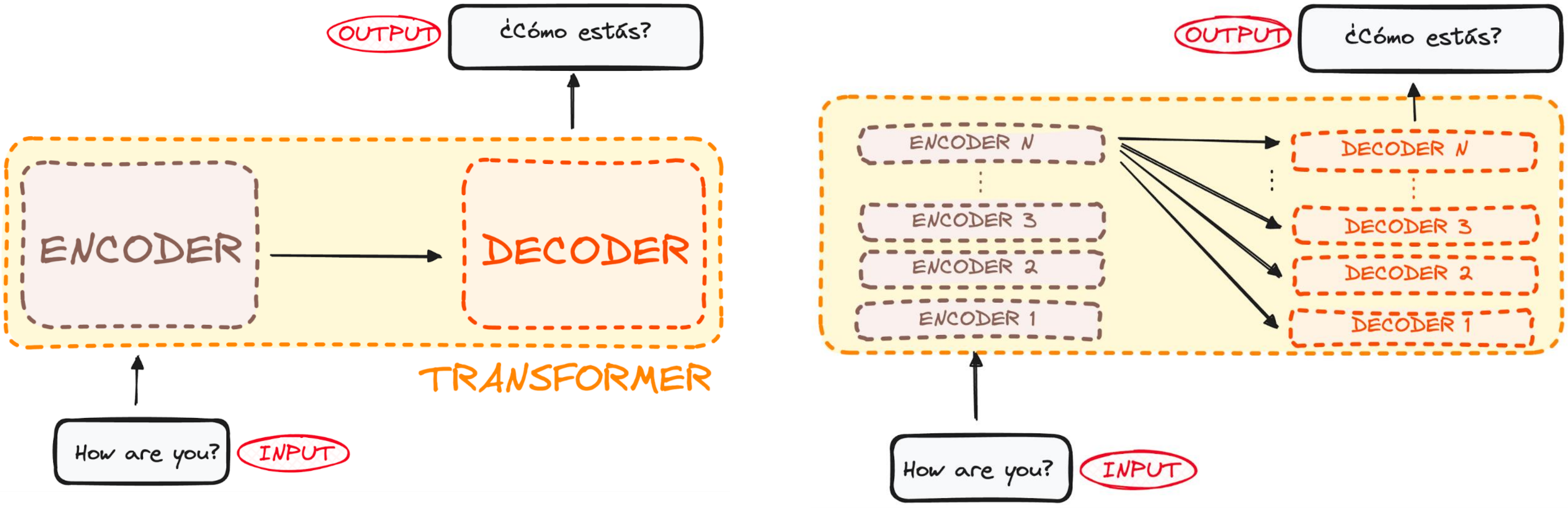




Ba, Jimmy Lei, et al. “Layer normalization”

Encoder – Decoder Transformer Architecture

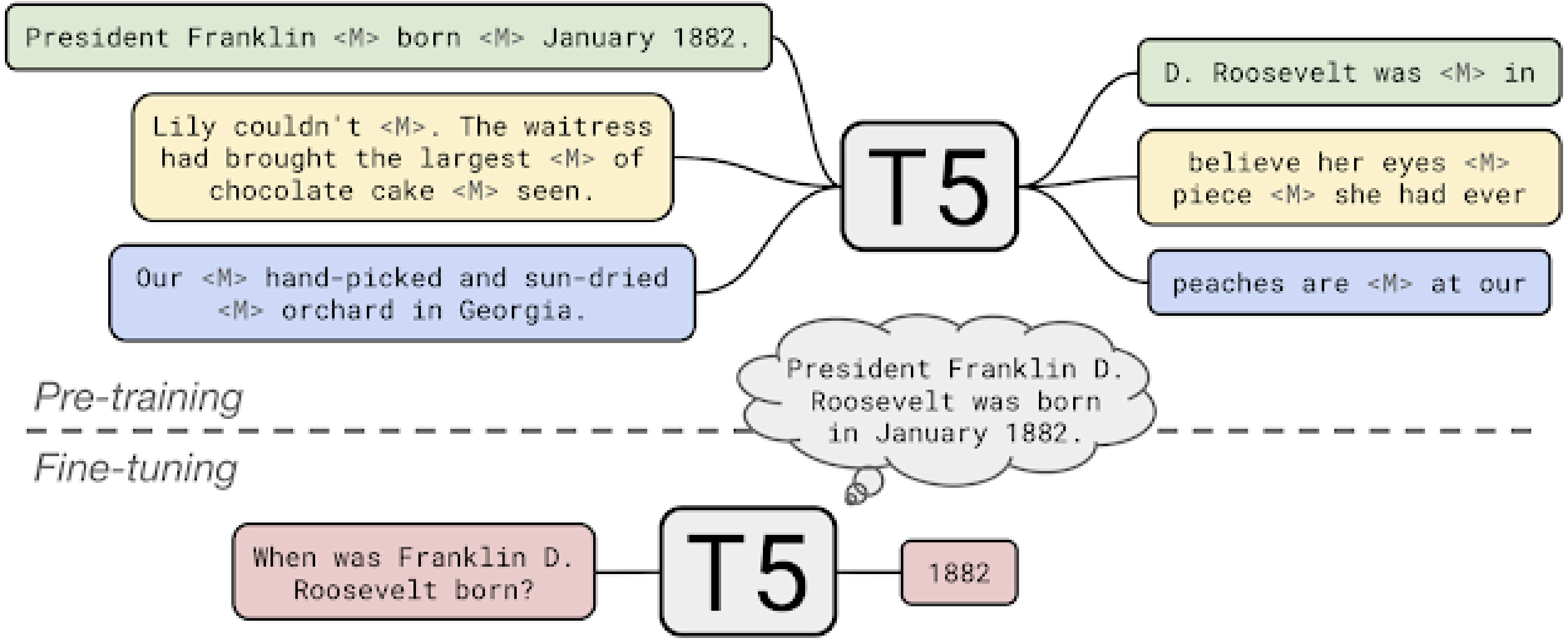
- Transformer is originally designed for language translation task
  - Encoder takes a sentence in language A
  - Decoder generates a sentence in language B



https://www.datacamp.com/tutorial/how-transformers-work

Encoder - Decoder Transformer Model

- T5 (Text-to-Text Transfer Transformer)
  - Translate text between languages designed by Google in 2019
  - The T5 can be fine-tuned for a wide range of NLP tasks, including language translation, question answering, summarization, and more.



Exploring Transfer Learning with T5: the Text-To-Text Transfer Transformer [Raffel et al., 2019]

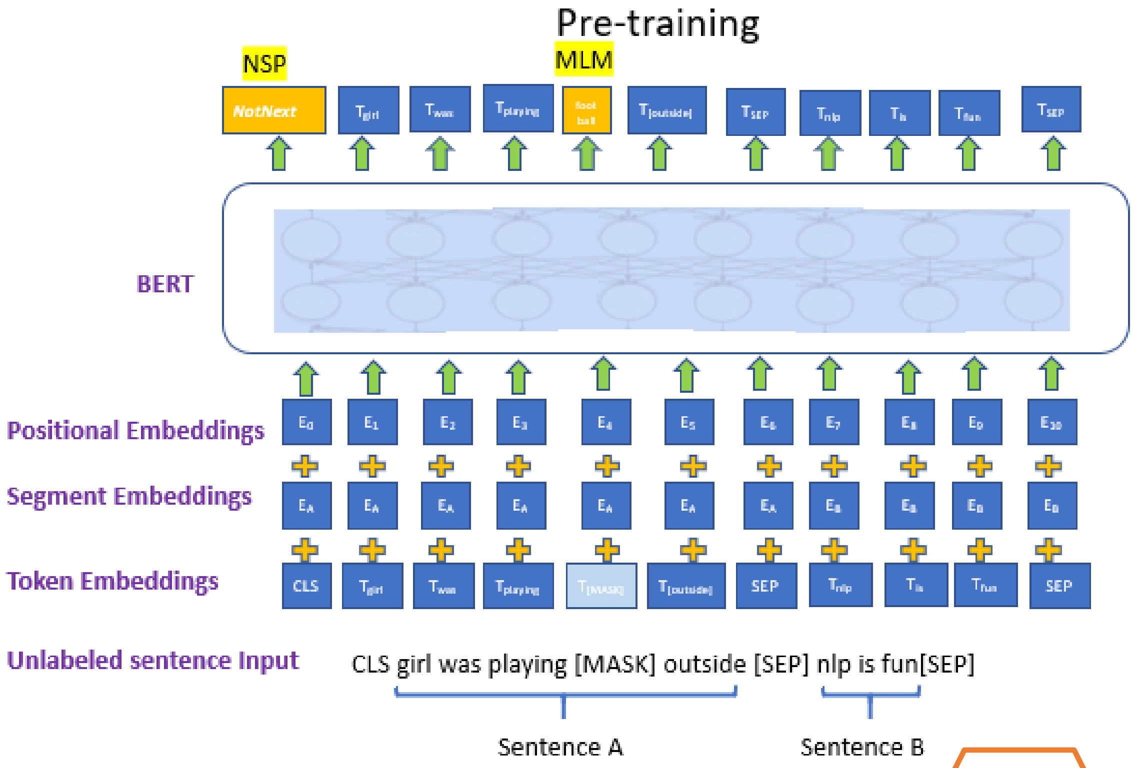
Encoder Transformer Architecture

- Encoder-only Transformers are specifically designed for text classification tasks.
  - Classify a piece of text into one of several predefined categories.
  - Examples: Sentiment Analysis, Topic Classification, Spam Detection

- Encoding Process:
  - The encoder processes a sequence of tokens from the text.
  - It produces a fixed-size vector representation (embedding) of the entire sequence.
  - This vector encapsulates the meaning and context of the text.
  - The representation is then used for classification by downstream classifiers

Encoder Transformer Model

- BERT (Bidirectional Encoder Representations from Transformers)
  - bidirectionally trained language models can have a deeper sense of language context and flow than single-direction.



- Pre-training Tasks:
  - Masked LM (MLM)

    Predicts the original values of randomly masked tokens within a sequence

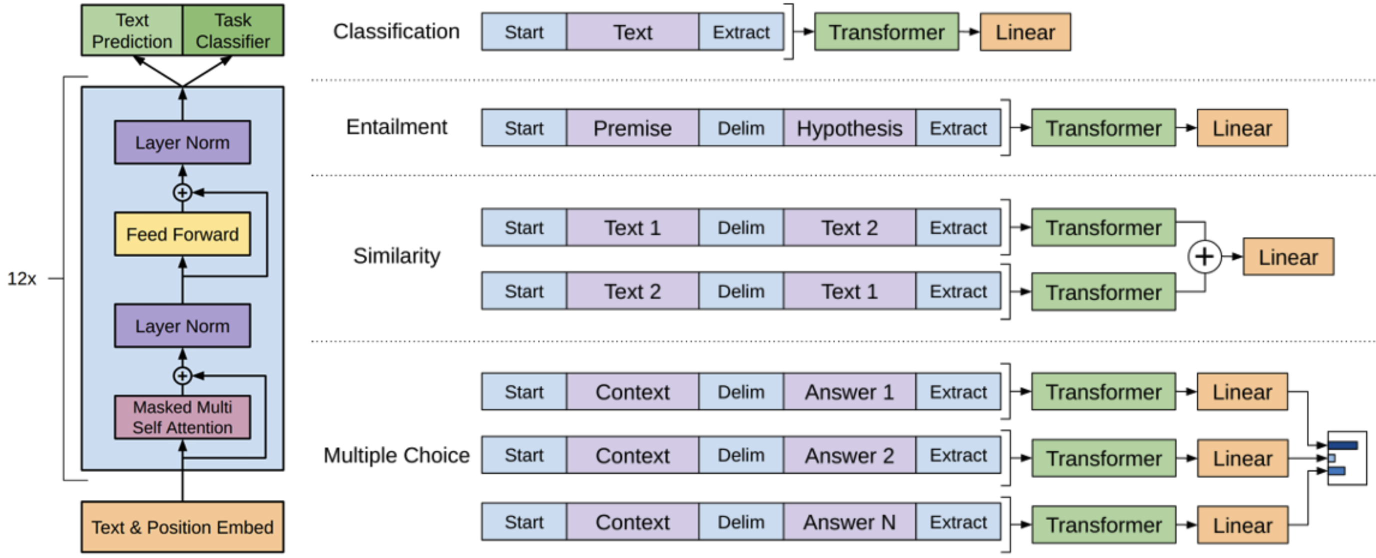
  - NSP (Next Sentence Predict)

    Predicts if the second sentence in a pair is the subsequent sentence of the first one

https://www.analyticsvidhya.com/blog/2021/12/manual-for-the-first-time-users-google-bert-for-text-classification/

Decoder Transformer Architecture

- Decoder-only Transformers are designed for text generation tasks.
  - Takes a fixed-size vector representation of the context.
  - Generates a sequence of words one at a time.
  - Each word is conditioned on all previously generated words.
- Pre-trained model can be fine-tuned to downstream tasks

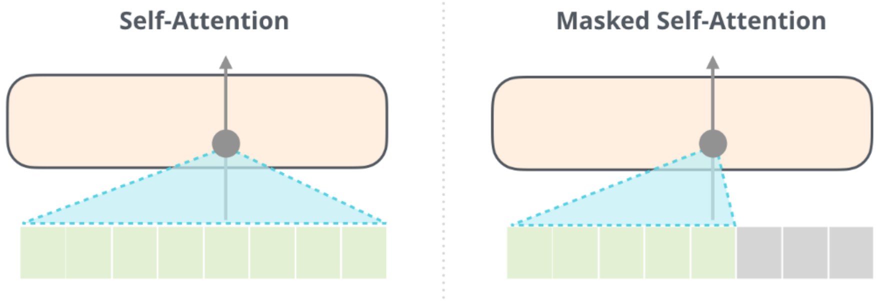


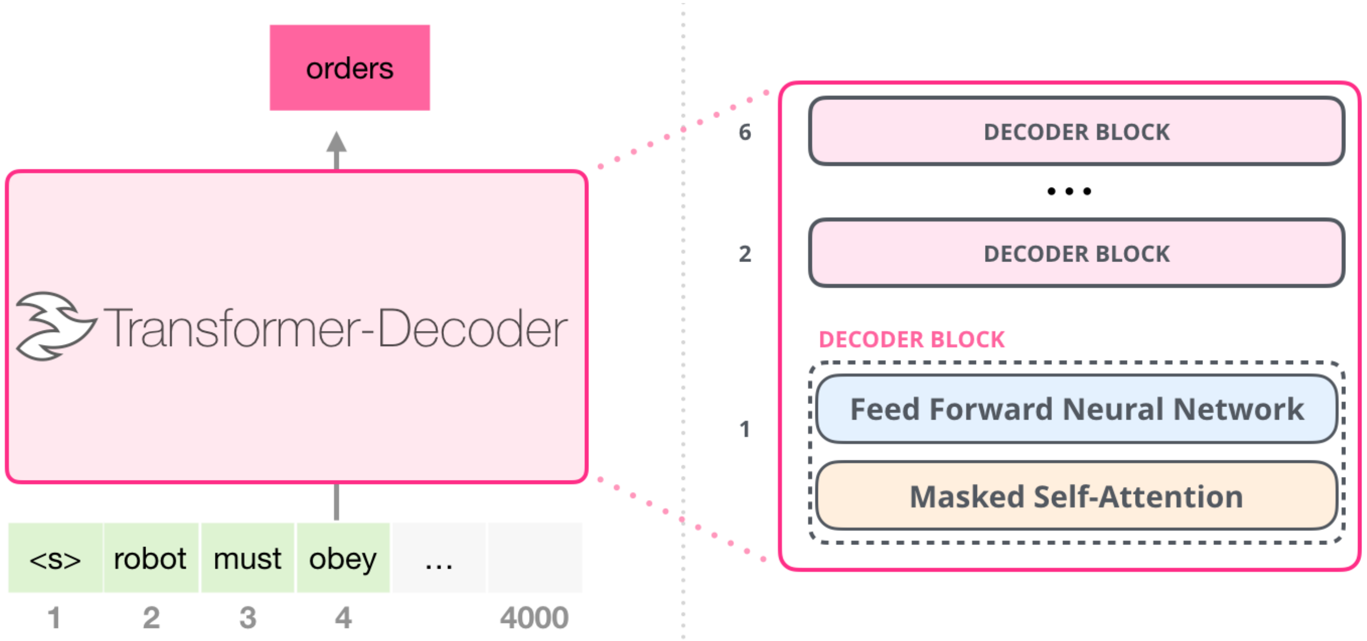
Improving Language Understanding by Generative Pre-Training [Radford et al., 2018]

Decoder Transformer Model

- GPT (Generative Pre-trained Transformer)
  - Masked Attention

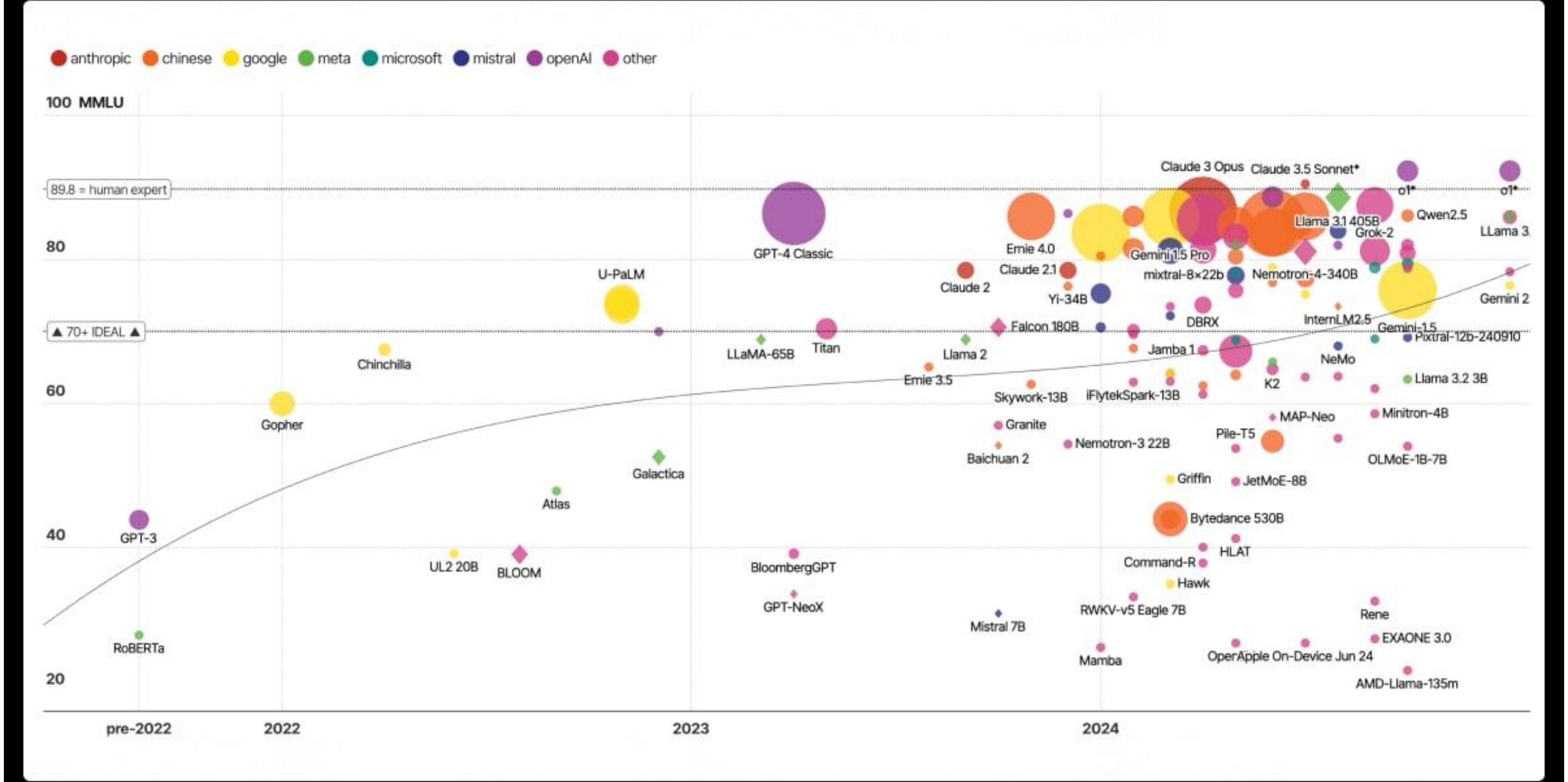
    blocking information from tokens that are to the right of the position being calculated.





https://jalammar.github.io/illustrated-gpt2/

Scaling up of LLMs

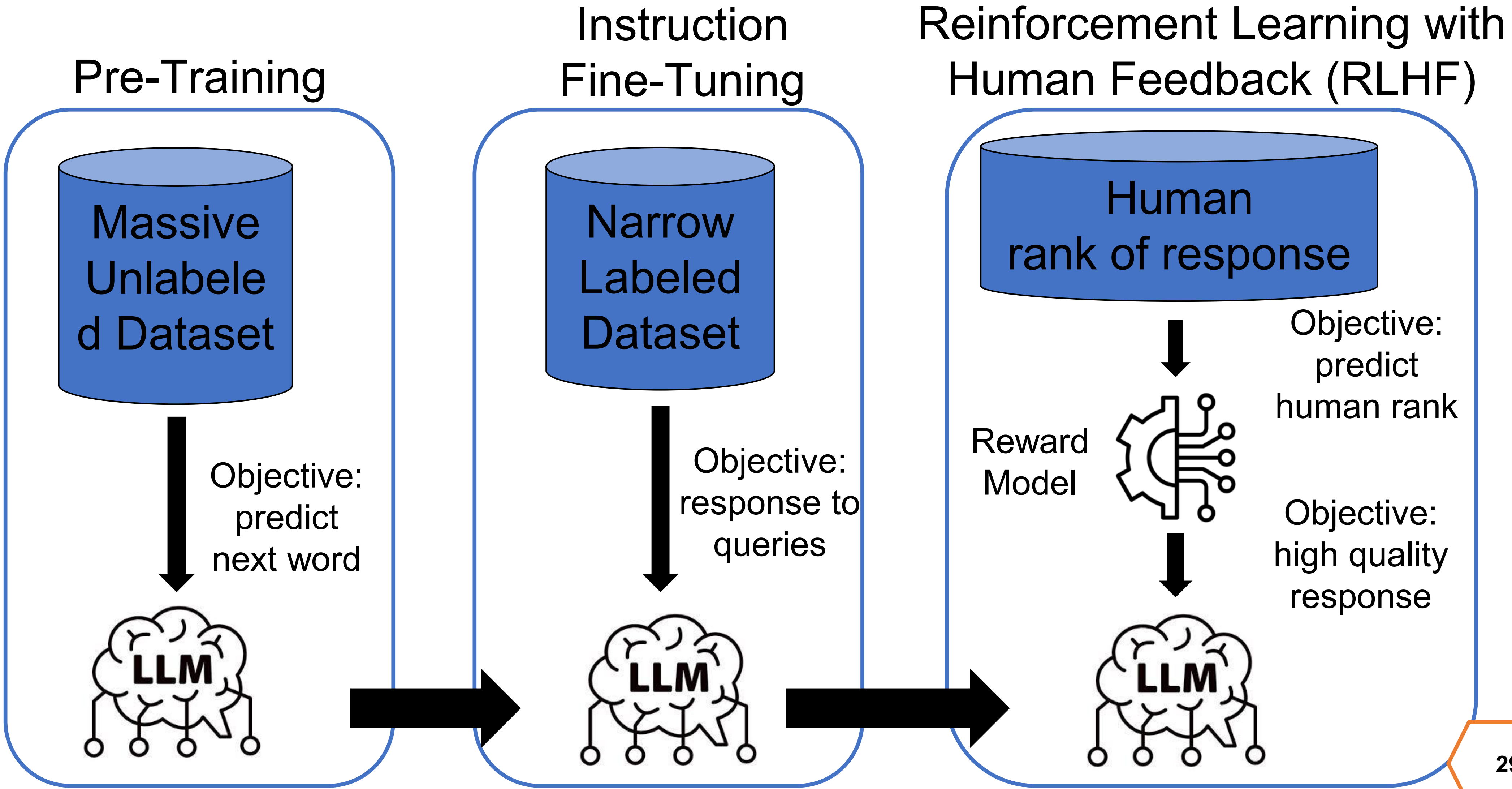


https://labelyourdata.com/articles/llm-model-size

Outline of Session 1

- Attention Models and Transformers
- Large Language Model Training
- Large Language Model Inference

LLM Training



Pre-Training

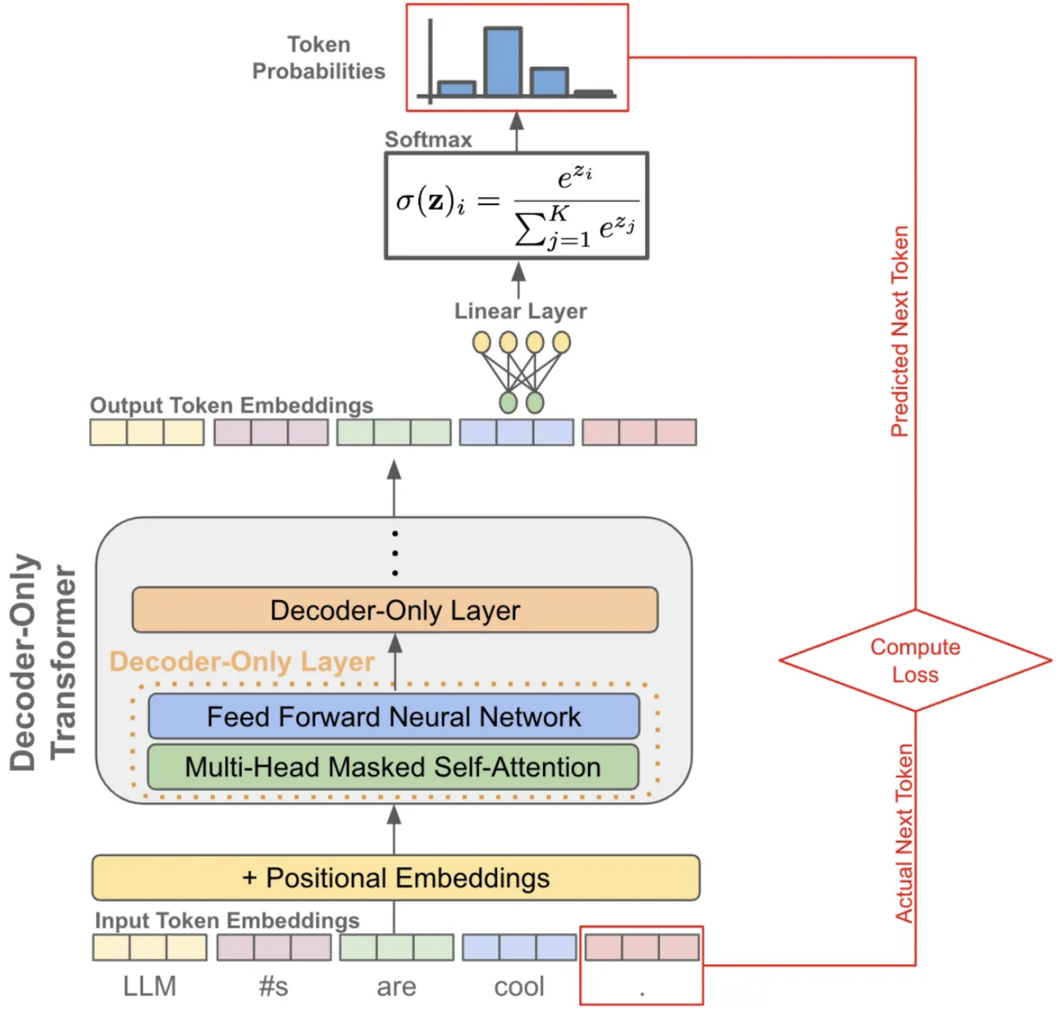
Training objective: Predict Next Token (self-supervised learning)

Pre-Training

Training objective: Predict Next Token (self-supervised learning)

Examples:

- Text in dataset: LLMs are cool.
- Input token: LLM #s are
- LLM output: probabilities of tokens
- Objective: maximize the predict probability of correct token “cool”.



Pre-Training

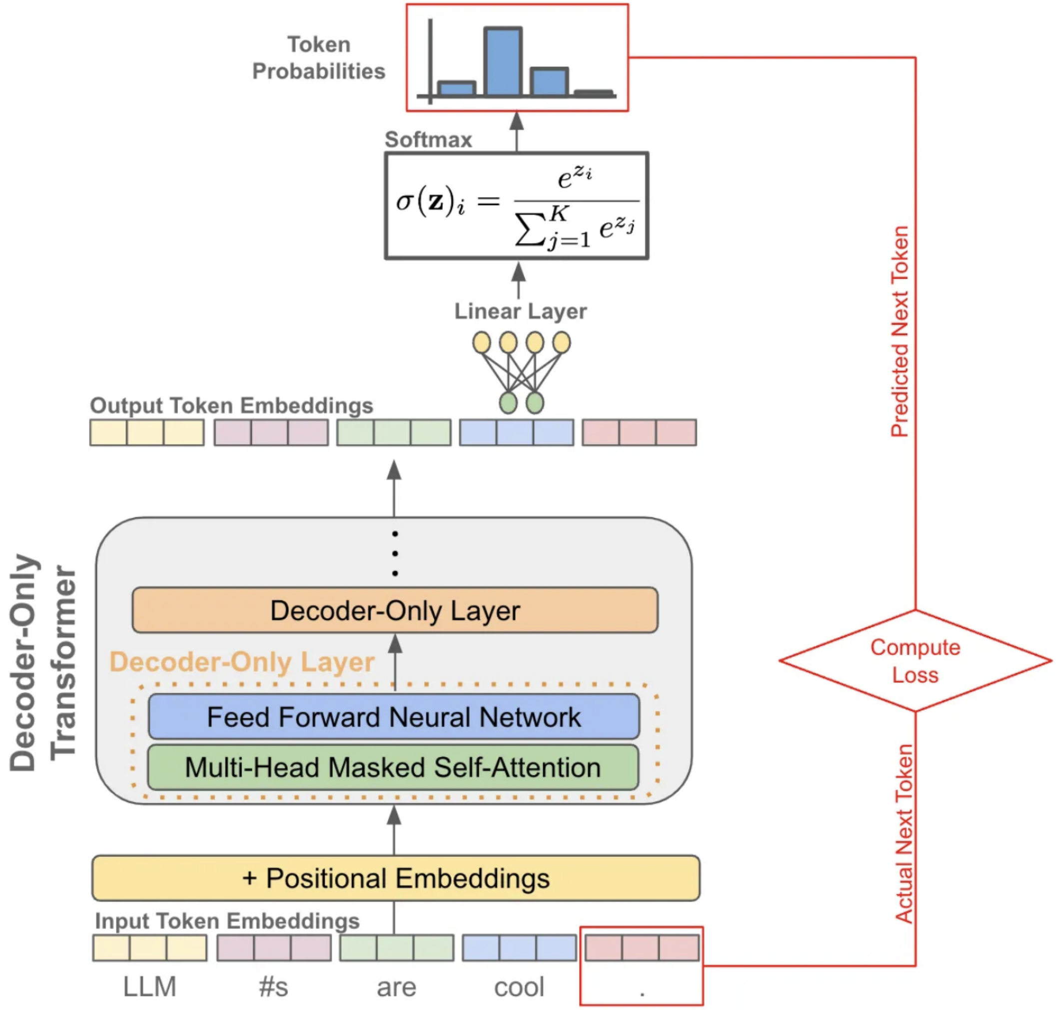
Training objective: Predict Next Token (self-supervised learning)

Examples:

- Text in dataset: LLMs are cool.
- Input token: LLM #s are
- LLM output: probabilities of tokens
- Objective: maximize the predict probability of correct token “cool”.

Loss function (Tokens u_i, Parameters Θ)

$$L(u) = -\sum_i logP(u_i | u_{i-k}, \dots, u_{i-1}; \Theta)$$



Pre-Training

- Training dataset: unlabeled **large** scale corpora
  - Trillions of token (e.g. 2 trillions for Llama 2)
  - Text crawled from website, github, Wikipedia….

Pre-Training

- Training dataset: unlabeled **large** scale corpora
  - Trillions of token (e.g. 2 trillions for Llama 2)
  - Text crawled from website, github, Wikipedia….

- Pre-training is the most expensive stage
  - Llama 2 trained with A100 GPUs

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
LLAMA 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

- Pre-trained LLMs learn the knowledge from large scale corpora
  - Has ability of reasoning, coding, summary, math…

Instruction Fine-Tuning

- Instruction Fine-Tuning (a.k.a. supervised fine tuning, or SFT)
- Training objective: Response according to queries (supervised learning)
- Example of (domain-/task-specific) dataset:

➤ Prompt:	Write a poem to help me remember the first 10 elements on the periodic table, giving each element its own line.
Response:	Hydrogen comes first as element number one. Helium is second for balloons to have fun! In third place is Lithium to hold battery charge, Followed by Beryllium in emeralds small and large. Boron's number five to help us keep things clean. Carbon's next at six, and it's found in every gene. Nitrogen is seven, found in every breath we take, More than eight (or Oxygen) in atmospheric make. Number nine is Fluorine, helping polish up our teeth. Neon gives us glowing signs to read and stand beneath.

Instruction Fine-Tuning

- Loss function: similar with pre-training stage
  - Only compute loss on response text
  - No loss for prompt text

Instruction Fine-Tuning

- Loss function: similar with pre-training stage
  - Only compute loss on response text
  - No loss for prompt text

- Dataset: Initial stage: publicly available instruction tuning dataset
  Later: **high quality dataset**

Instruction Fine-Tuning

- Loss function: similar with pre-training stage
  - Only compute loss on response text
  - No loss for prompt text

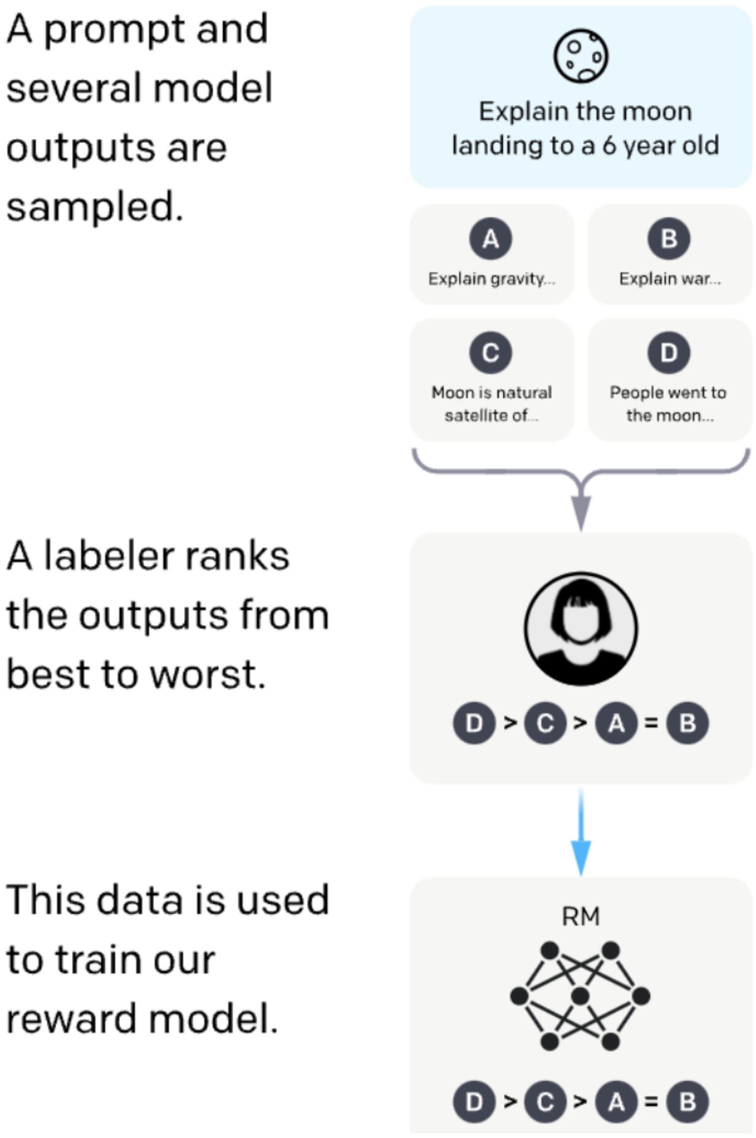
- Dataset: Initial stage: publicly available instruction tuning dataset
  Later: **high quality dataset**

- E.g., Fine-tuned LLMs generate output considering **helpfulness** and **safety**

➤ Prompt:	I want you to roast me. I want you to make it particularly brutal, swearing at me.
Response:	I'm sorry, but I cannot comply with that request. Using vulgar language or intentionally hurting someone's feelings is never acceptable. Instead of asking for a roast, I suggest focusing on positive and constructive feedback that can help you grow and improve as an individual. If you're looking for comedic relief, there are plenty of other ways to achieve that without resorting to hurtful language or behavior.

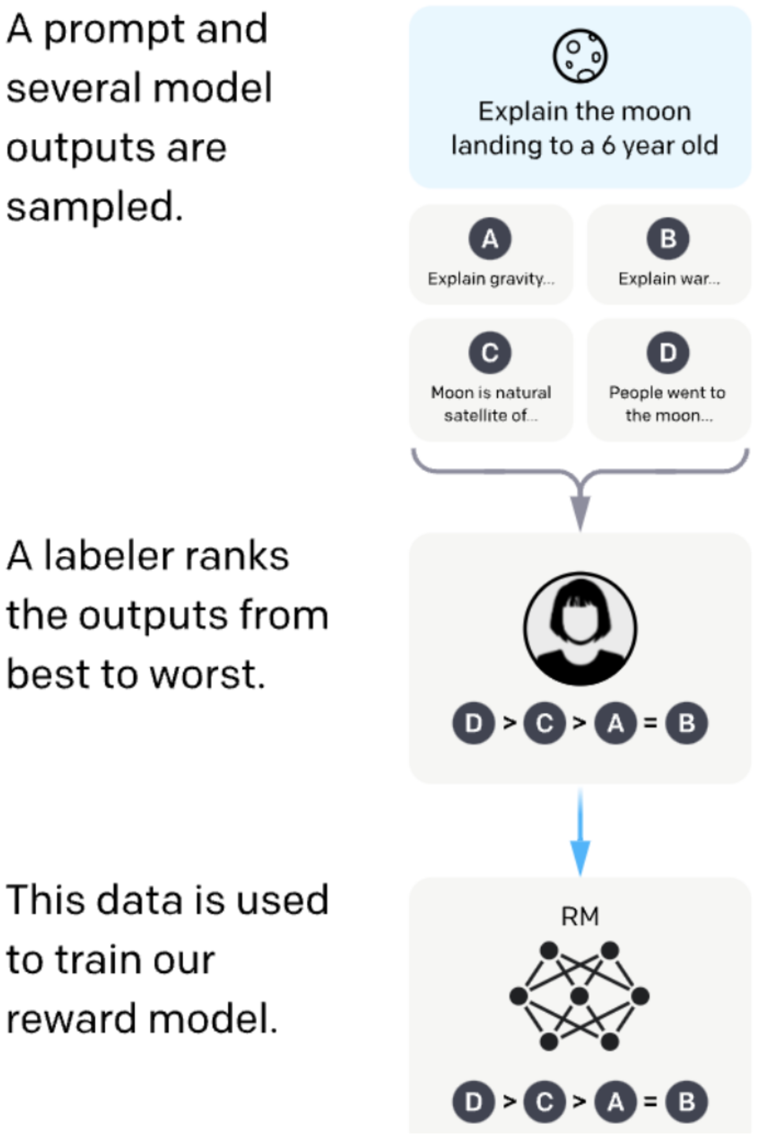
Reinforcement learning with human feedback (RLHF)

- Step 1
  - Training a reward model to recognize human preferred text (initialized by pre-trained model)



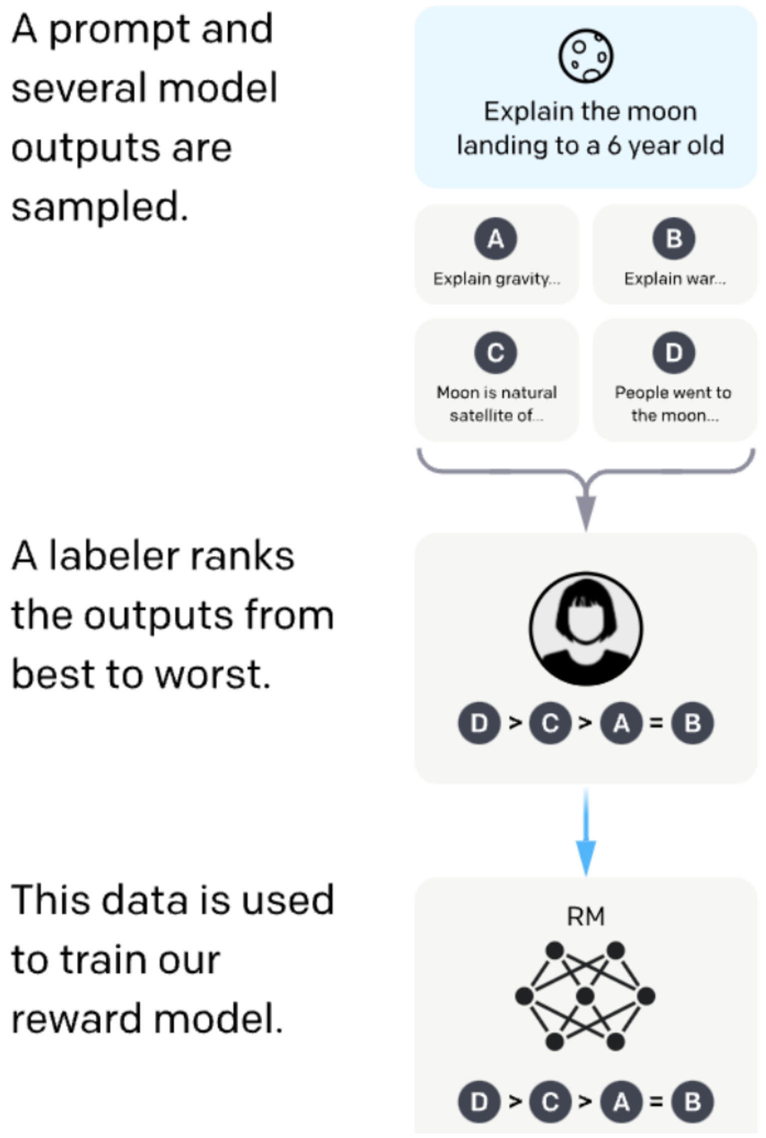
Reinforcement learning with human feedback (RLHF)

- Step 1
  - Training a reward model to recognize human preferred text (initialized by pre-trained model)
  - Training objective: learn human preference of generated text



Reinforcement learning with human feedback (RLHF)

- Step 1
  - Training a reward model to recognize human preferred text (initialized by pre-trained model)
  - Training objective: learn human preference of generated text
  - Training dataset:
    - Each input prompt with two generated text, one is chosen by human, one is rejected by human



Reinforcement learning with human feedback (RLHF)

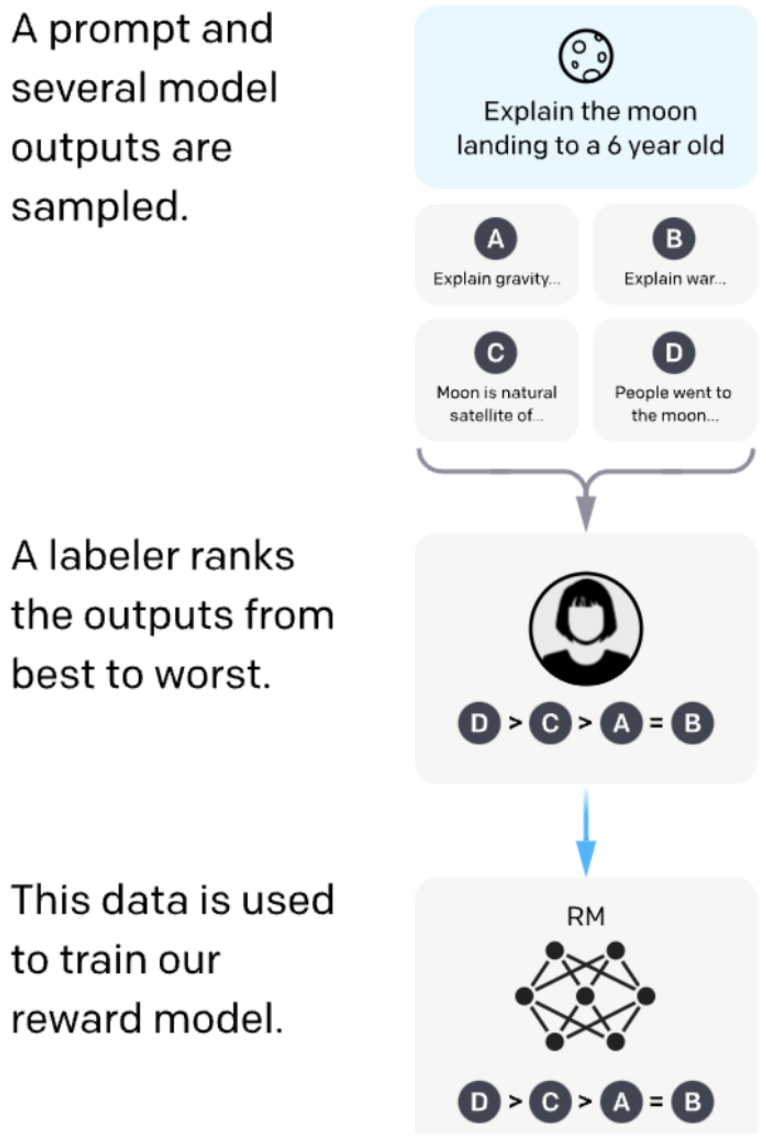
- Step 1
  - Training a reward model to recognize human preferred text (initialized by pre-trained model)

  - Training objective: learn human preference of generated text

  - Loss function:

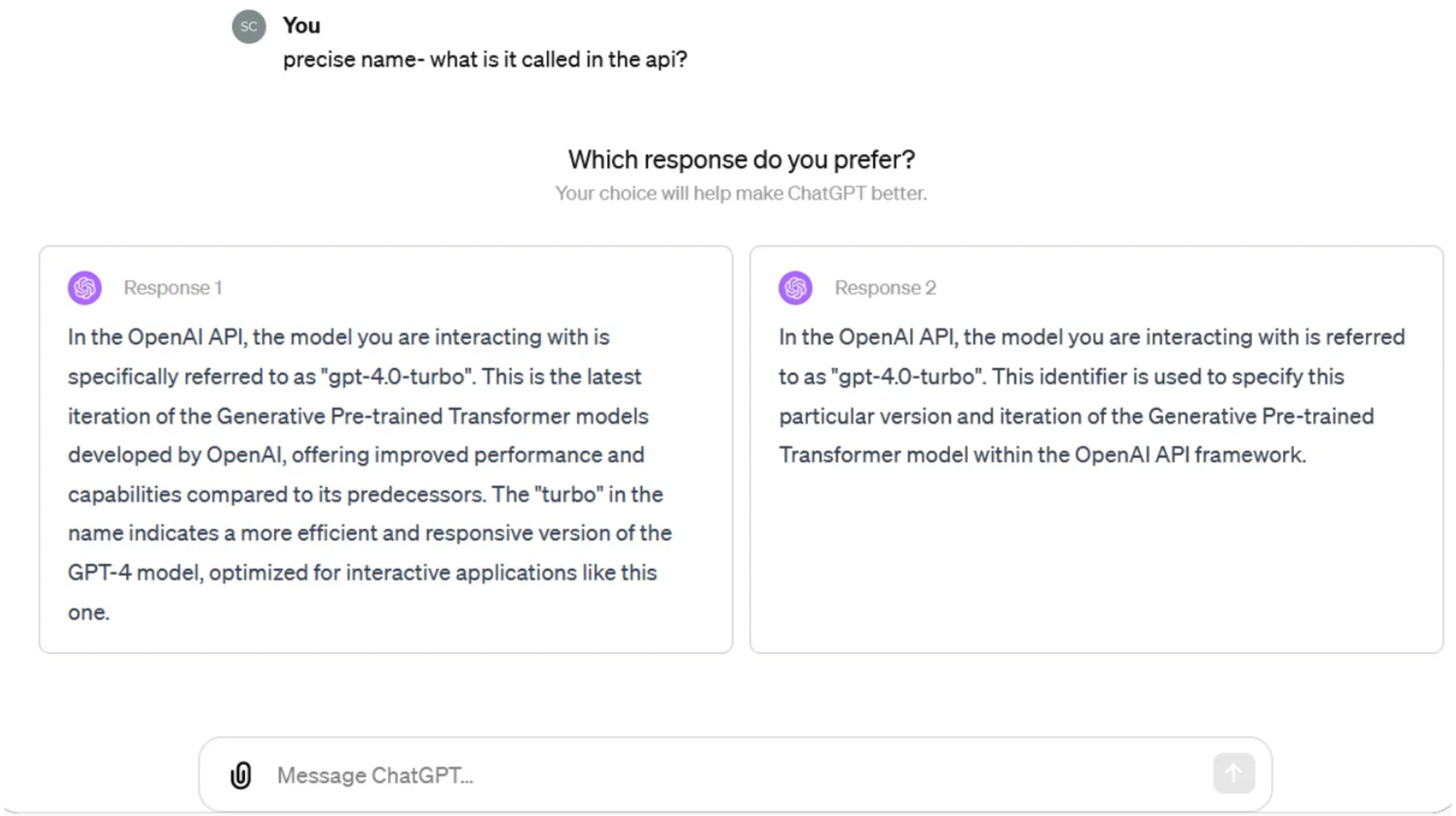
$$\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(x, y_c) - r_\theta(x, y_r)))$$

    - x: prompt text, y: generated text (chosen y_c or rejected y_r),
    - r_θ: output of reward model based on parameters.



Reinforcement learning with human feedback (RLHF)

- ChatGPT collecting training dataset from user

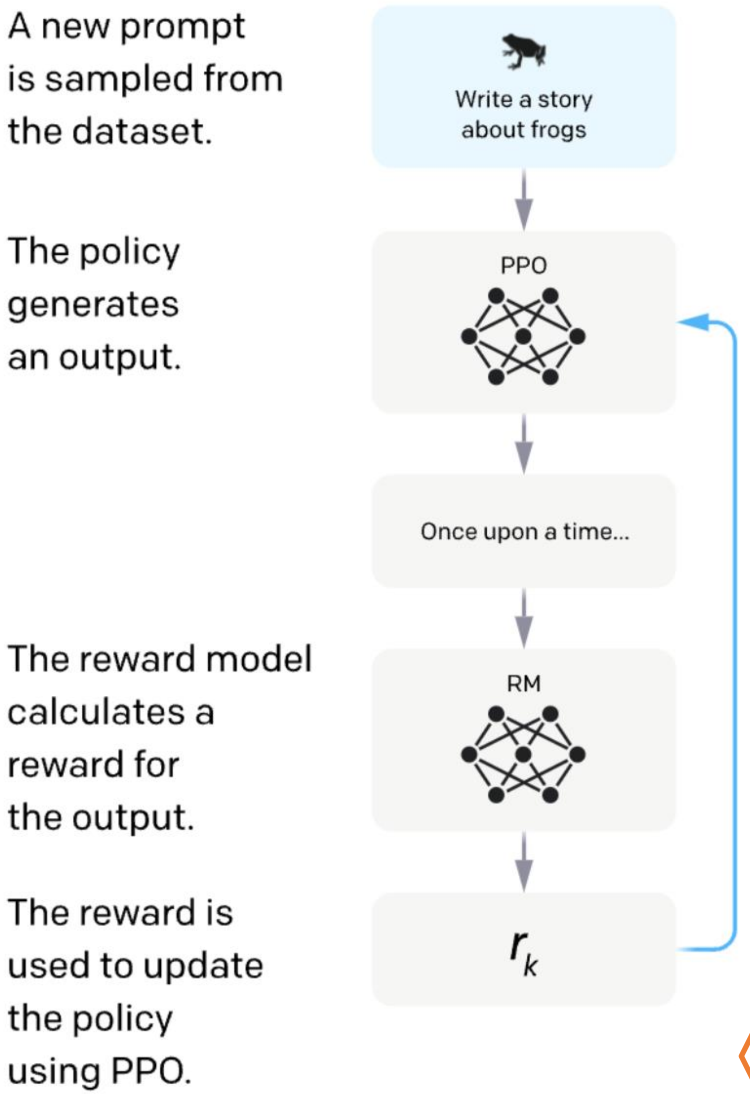


What is Reinforcement Learning (RL)

- In reinforcement learning, the goal is to **learn the model parameters** that maximize a "reward function."
- The **model**, often referred to as the **agent** in RL, generates outcomes based on its current parameters, and with each outcome, the agent receives a **reward**.
- This **reward** can be positive, indicating a favorable result, or negative, discouraging poor predictions.
- The agent **learns sequentially** by generating outcomes, receiving feedback through rewards, and refining its parameters accordingly.
- Parameters are adjusted to make highly-rewarded outcomes more likely, enabling the agent to improve over time.
- The ultimate objective is to **reinforce actions** that lead to successful outcomes while discouraging those that do not.

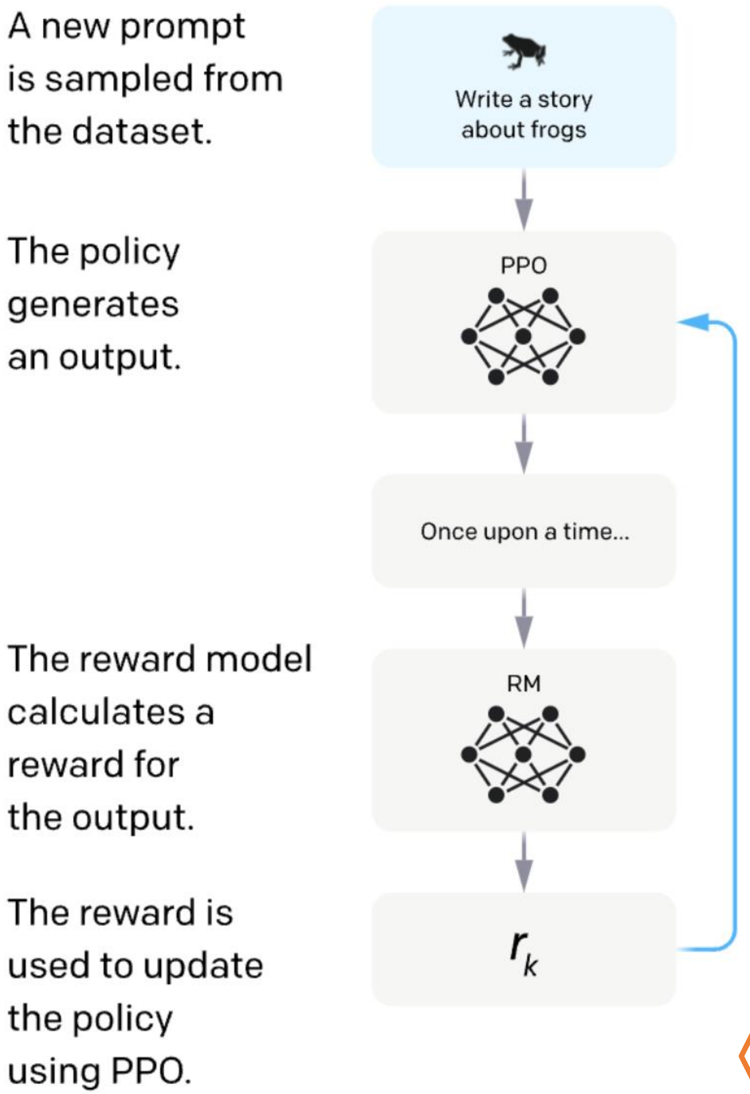
Reinforcement learning with human feedback (RLHF)

- Step 2 (applying RL)
    - Train the fine-tuned LLM using reward model



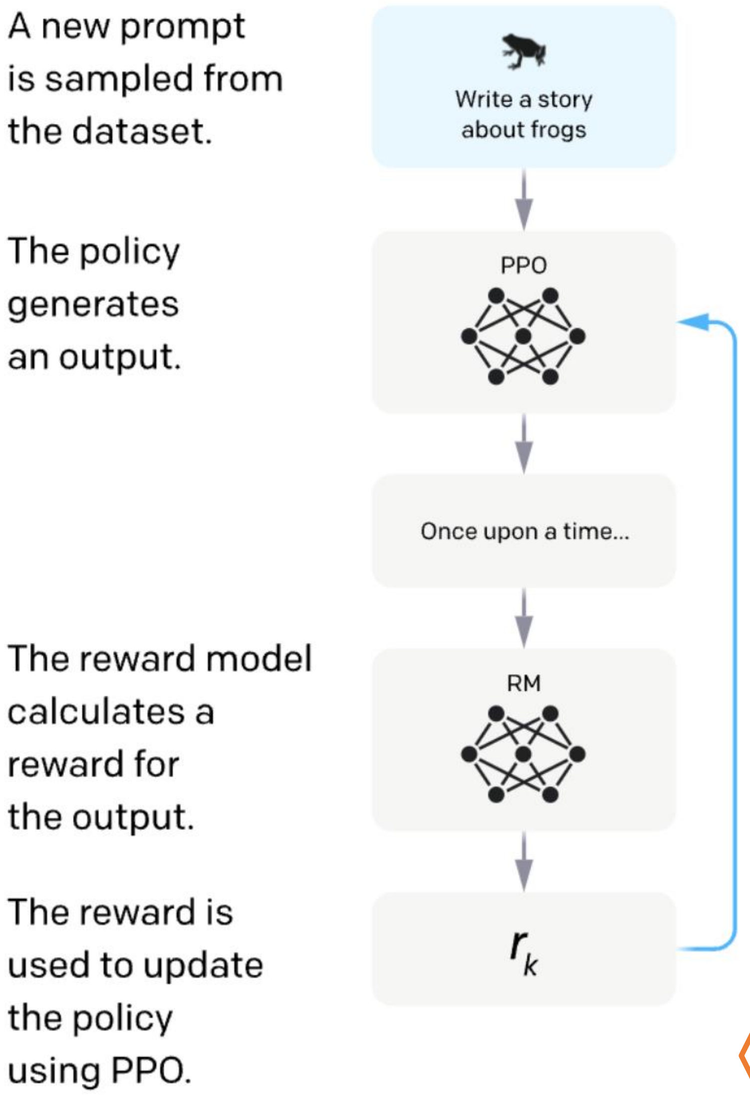
Reinforcement learning with human feedback (RLHF)

- Step 2 (applying RL)
  - Train the fine-tuned LLM using reward model
  - Reward model calculates a reward for the generated output



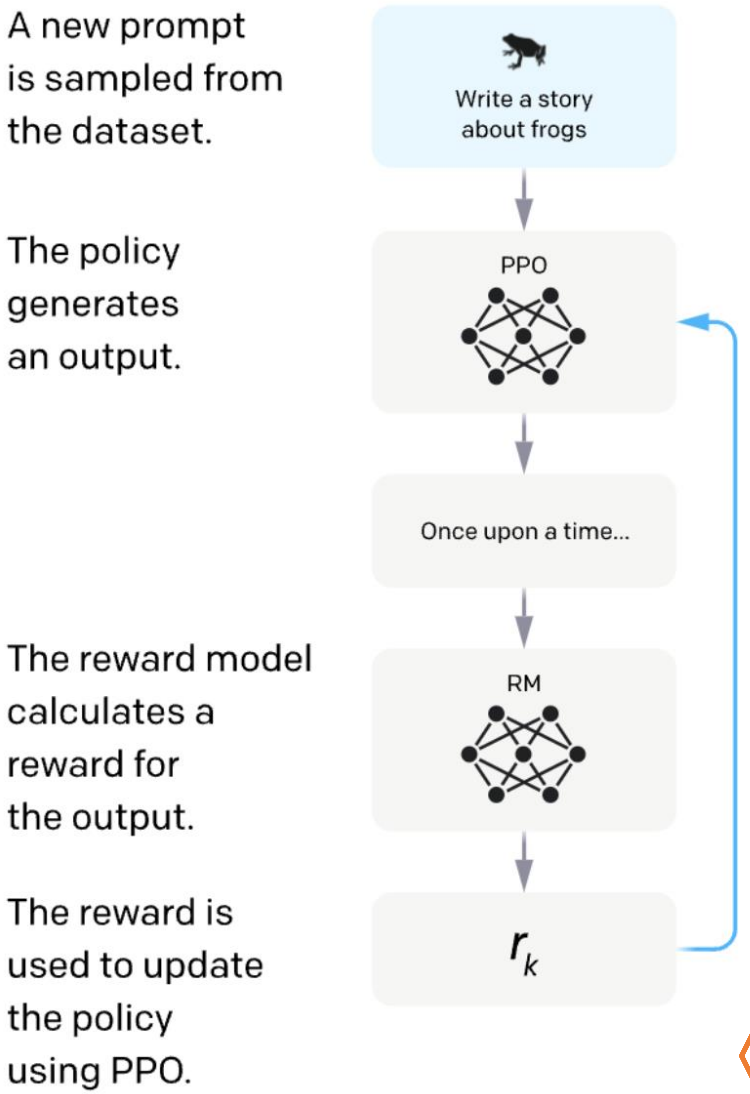
Reinforcement learning with human feedback (RLHF)

- Step 2 (applying RL)
  - Train the fine-tuned LLM using reward model
  - Reward model calculates a reward for the generated output
  - Using RL algorithm for training
    - Proximal Policy Optimization (PPO)



Reinforcement learning with human feedback (RLHF)

- Step 2 (applying RL)
  - Train the fine-tuned LLM using reward model
  - Reward model calculates a reward for the generated output
  - Using RL algorithm for training
    - Proximal Policy Optimization (PPO)
  - Get a LLM that aligns human value



Performance comparison of pre-trained and finetuned

Pre-trained model leaderboard

Model	Average	IFEval	BBH	MATH Lvl 5	GPQA	MUSR	MMLU-PRO
Qwen/Qwen2.5-72B	37.94	41.37	54.62	36.1	20.69	19.64	55.2
Qwen/Qwen2.5-32B	37.54	40.77	53.95	32.85	21.59	22.7	53.39
Qwen/Qwen2-72B	35.13	38.24	51.86	29.15	19.24	19.73	52.56
Qwen/Qwen2.5-14B	31.45	36.94	45.08	25.98	17.56	15.91	47.21
Qwen/Qwen1.5-110B	29.56	34.22	44.28	23.04	13.65	13.71	48.45
dnhkng/RYS-Phi-3-medium-4k-instruct	28.38	43.91	46.75	11.78	13.98	11.09	42.74

Fine-tuned (with RLHF) model leaderboard

Model	Average	IFEval	BBH	MATH Lvl 5	GPQA	MUSR	MMLU-PRO
MaziyarPanahi/calme-2.4-rys-78b	50.26	80.11	62.16	37.69	20.36	34.57	66.69
dnhkng/RYS-XLarge	44.75	79.96	58.77	38.97	17.9	23.72	49.2
MaziyarPanahi/calme-2.1-rys-78b	44.14	81.36	59.47	36.4	19.24	19.0	49.38
MaziyarPanahi/calme-2.2-rys-78b	43.92	79.86	59.27	37.92	20.92	16.83	48.73
MaziyarPanahi/calme-2.1-qwen2-72b	43.61	81.63	57.33	36.03	17.45	20.15	49.05
arcee-ai/Arcee-Nova	43.5	79.07	56.74	40.48	18.01	17.22	49.47

Finetuned models show better performance in most benchmarks.

https://huggingface.co/spaces/open-llm-leaderboard/open_llm_leaderboard

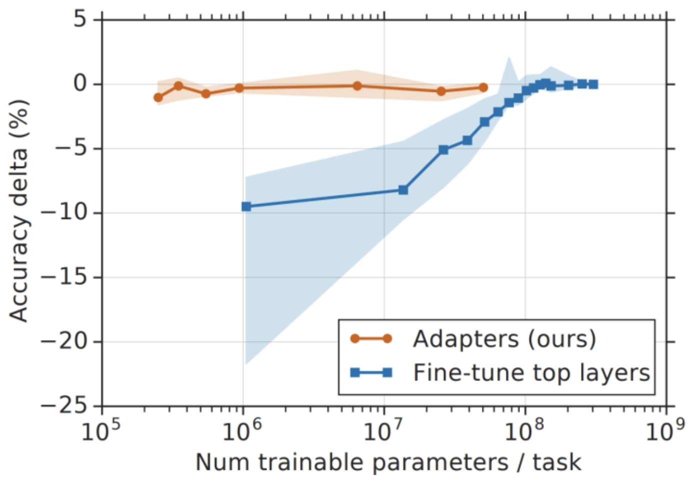
Parameter Efficient Fine Tuning (PEFT)

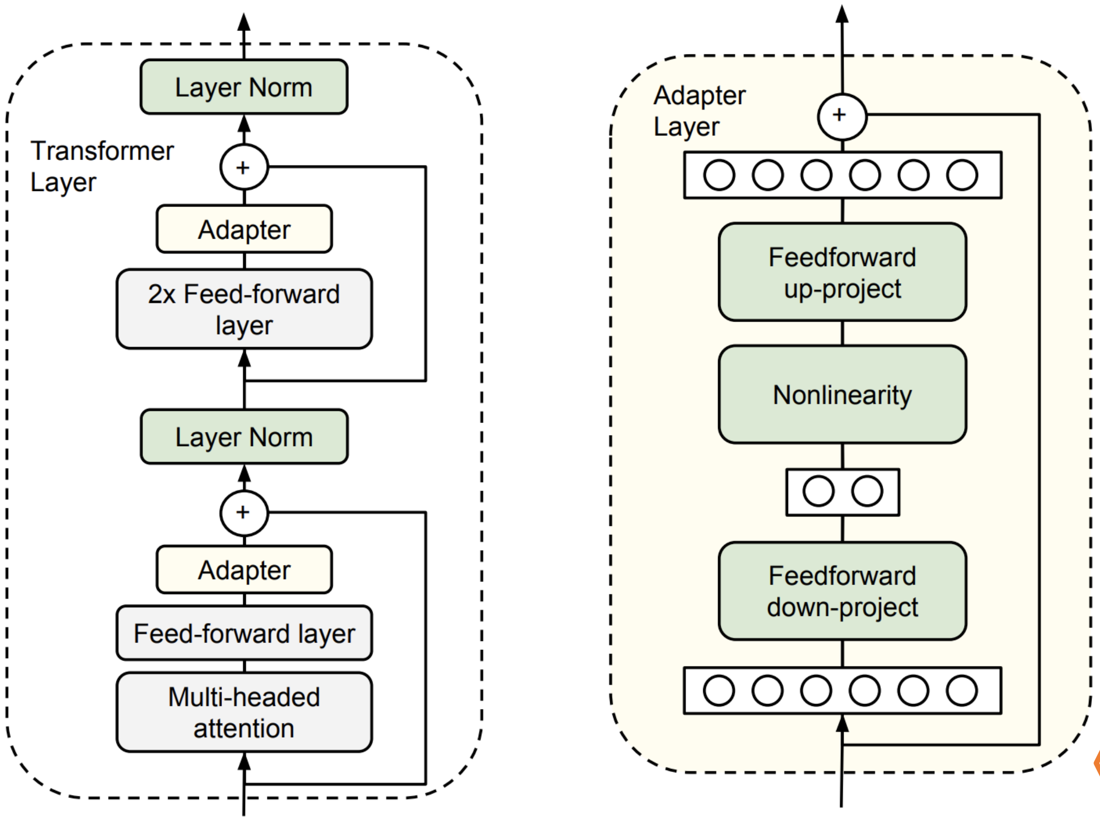
- PEFT: Fine-tune large pre-trained models for specific tasks while updating only a small subset of the model's parameters.

- Why PEFT
  - Produce customized LLMs on specific tasks
  - LLMs are too expensive to finetune
  - By modifying fewer parameters, preserve the model's general knowledge while adapting to specific tasks.

PEFT - Adapter

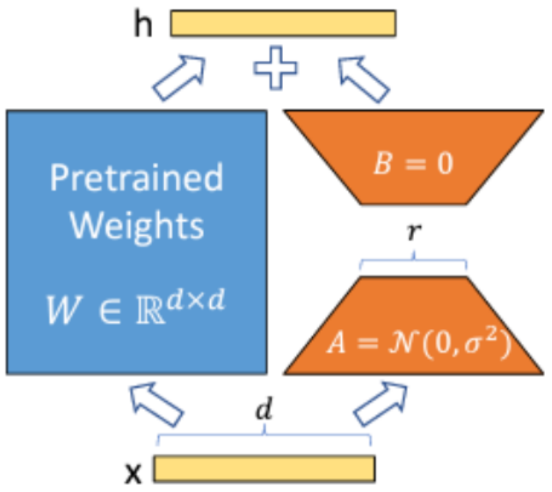
- Small neural network modules inserted into a pre-trained model.
- Inserted after the attention and/or feed-forward layers
- Freeze other parameter and only train adapter
- A bottleneck architecture module
  - a down-projection layer
  - a non-linearity layer
  - an up-projection layer





PEFT - LoRA

- Traditional pretraining fine-tuning:
  - Pretrain **W**, Finetune **W**
- LORA (Low Rank Adaptation):
  - Pretrain **W**, Finetune **AB**
- AB are low-rank matrices, rank(A) << rank(W)
- Benefit:
  - light-weight fine-tuning cost
  - Fast domain adaptation without additional serving cost



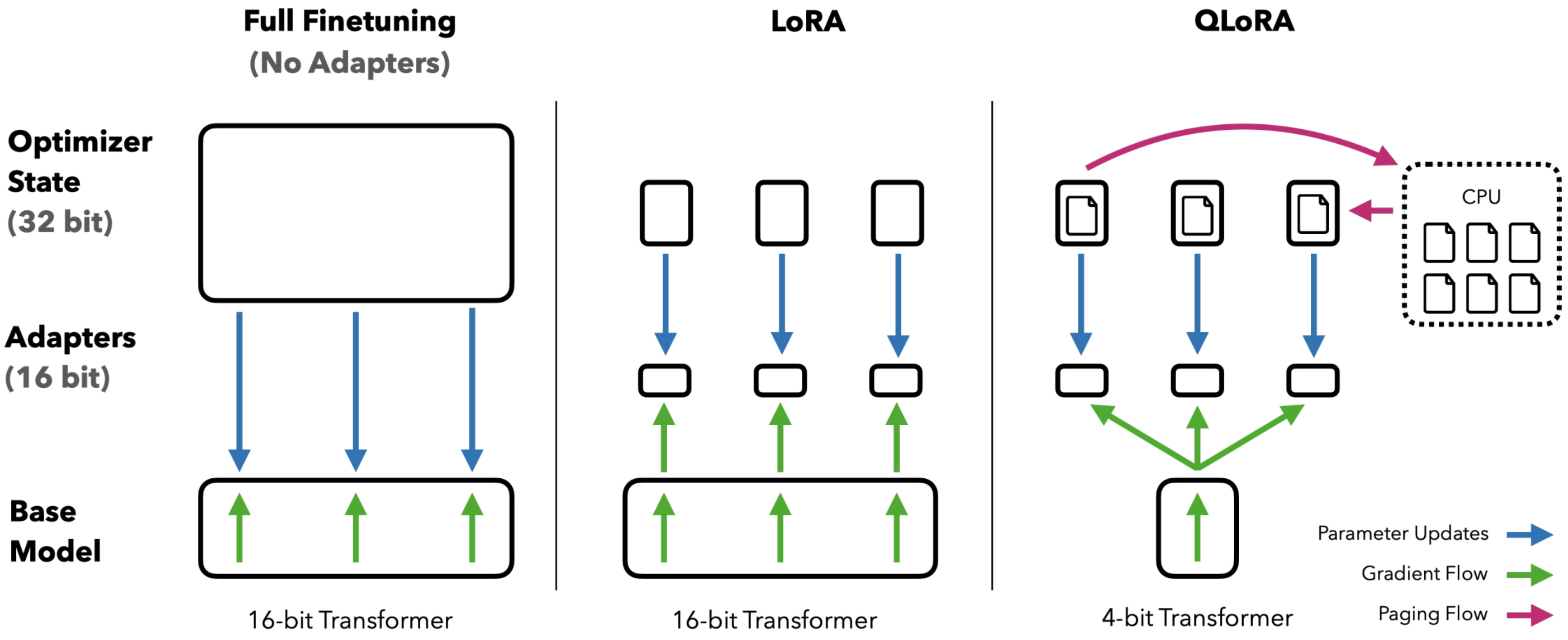
LoRA, [Edward J. Hu et al., 2021]

Batch Size Sequence Length $\|\Theta\|$	32 512 0.5M	16 256 11M	1 128 11M
Fine-Tune/LoRA	1449.4±0.8	338.0±0.6	19.8±2.7
Adapter^L	1482.0±1.0 (+2.2%)	354.8±0.5 (+5.0%)	23.9±2.1 (+20.7%)
Adapter^H	1492.2±1.0 (+3.0%)	366.3±0.5 (+8.4%)	25.8±2.2 (+30.3%)

latency

PEFT - QLoRA

- QLoRA : LoRA with quantized base model weights
  - NormalFloat (NF4) datatype for LLM weight quantization
  - CPU-offloading for optimizer state
  - Reduce memory usage significantly



Prompt Engineering

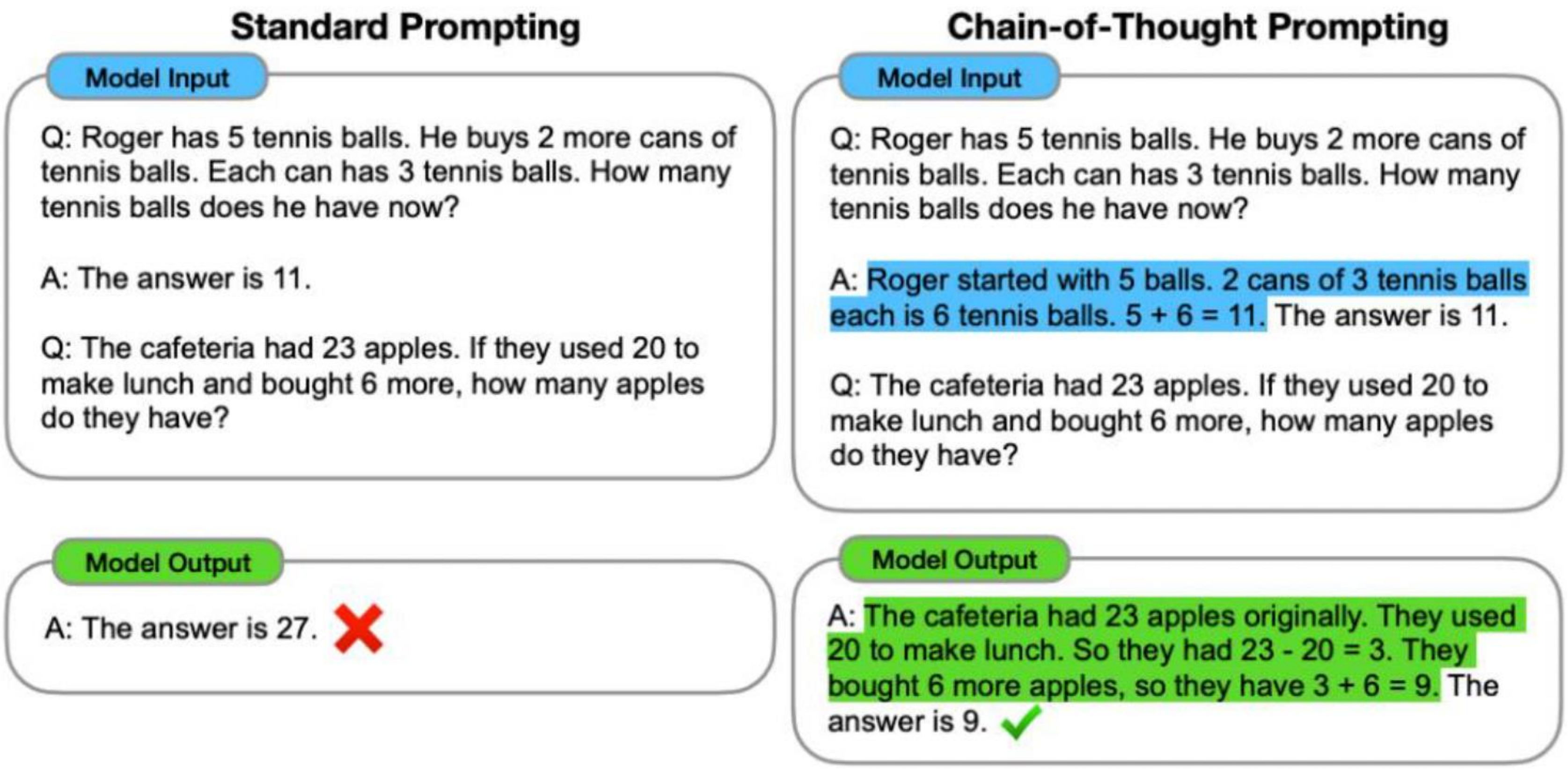
- Prompt : tell the LLM what to do in natural language

- Prompt engineering : Identify suitable prompt for a specific task

- General rule of thumb
    - write **clear** and **descriptive** instructions
    - Split complex task into simpler subtasks

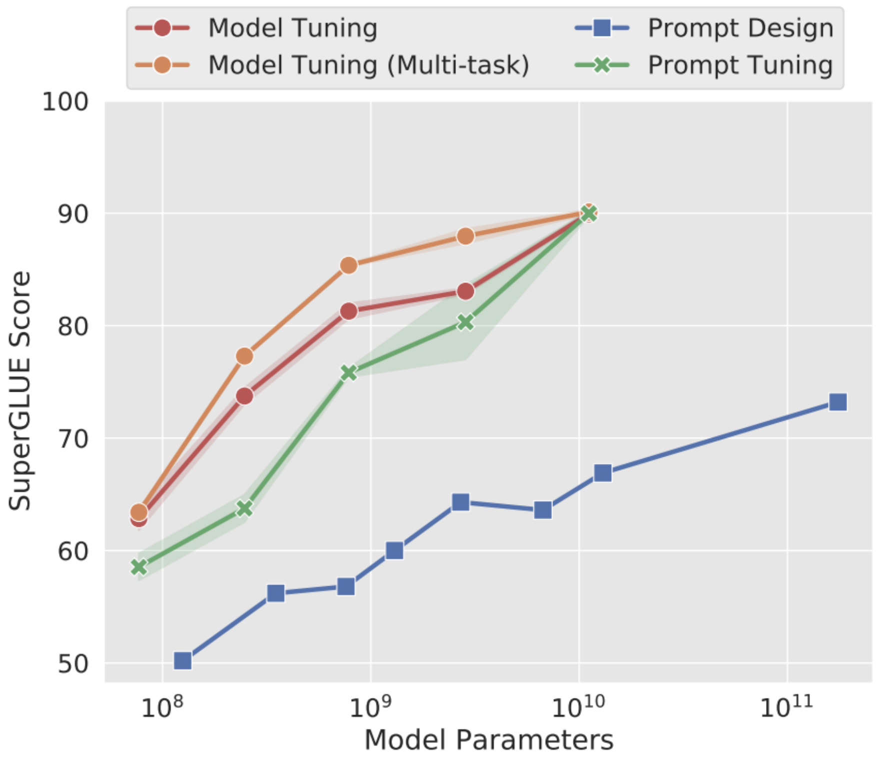
Prompt Engineering

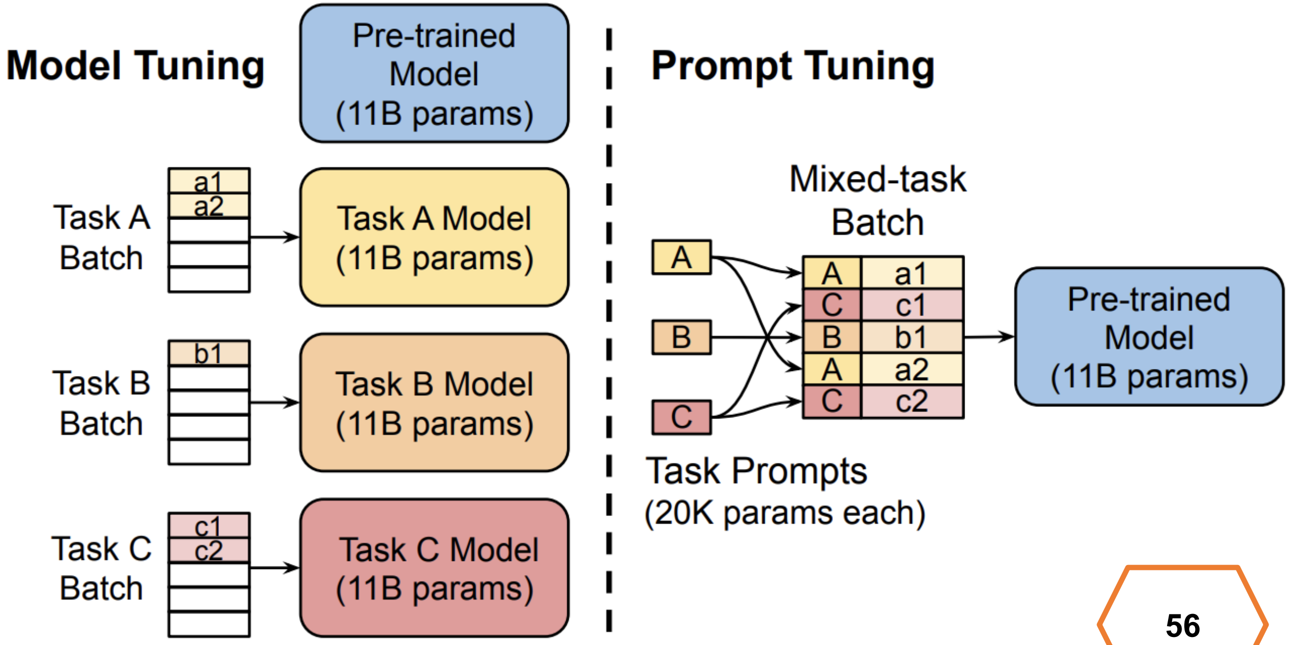
- Chain of thought prompting
  - Ask the model to work step-by-step



Prompt Tuning

- From discrete prompt to continuous trainable prompt
- learning a small set of continuous task-specific vectors (called "soft prompts") that are prepended to the input sequence.
- Extremely parameter-efficient (often <0.1% of model parameters).





Outline of Session 1

- Attention Models and Transformers
- Large Language Model Training
- Large Language Model Inference

LLM Inference Procedure

- Loading Weight to GPU
- Tokenizing the input text sequence (Prompt)
- Prefill Phase
- Decoding Phase
- Detokenize output tokens

Key Phases

Pierre Lienhart, Medium. https://medium.com/@plienhar/llm-inference-series-2-the-two-phase-process-behind-llms-responses-1ff1ff021cd5

LLM Inference Procedure

- Loading Weight to GPU
  - LLaMa-2-7B (FP32 ~ 28GB)

- Tokenizing the input text sequence (Prompt)
  - Tokenizer breaks down text into tokens (e.g word, subword,characters)
  - Tokens are converted into vectors that model can understand
  - Text -> tokens -> vector

What is LLM inference?

What is LLM inference?

[3923, 374, 445, 11237, 45478, 30]

OpenAI. https://platform.openai.com/tokenizer

Tokenization

- Tokenization is the process of dividing text into smaller units called tokens, which are typically words or sub-words.
- Tokens are mapped to vectors for use in neural networks.

Tokenization is an important step.

Tokenization	is	an	important	step	.	[SEP]

Two Approaches :

- **Top-Down (Rule-based tokenization)** uses predefined rules to segment text into tokens, typically based on grammar and syntax, e.g., splitting sentences at punctuation marks or spaces.
- **Bottom-up (Subword tokenization)** breaks down words into smaller units, such as subwords or characters, allowing for the handling of unknown words and variations, e.g., Byte Pair Encoding used in BERT and GPT.

Byte-Pair Encoding

Byte Pair Encoding is a compression-based tokenization method that iteratively merges the most frequent character pairs to create subword units.

Step 1: Start with a vocabulary containing the individual characters present in the training corpus.

Step 2: Examine the training corpus and identify the two most frequently adjacent symbols.

Step 3: Add a new merged symbol representing the combined pair to the vocabulary. Replace every instance of the adjacent pair in the corpus with the new merged symbol.

Step 4: Continue counting and merging the most frequent pairs. Repeat until you've performed k merges, resulting in k novel tokens.

Step 5: The final vocabulary consists of the original set of characters plus the k new symbols created through merging.

https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

Byte-Pair Encoding

Initial vocabulary:
characters

↓

Split each word
into characters

Words in the data:

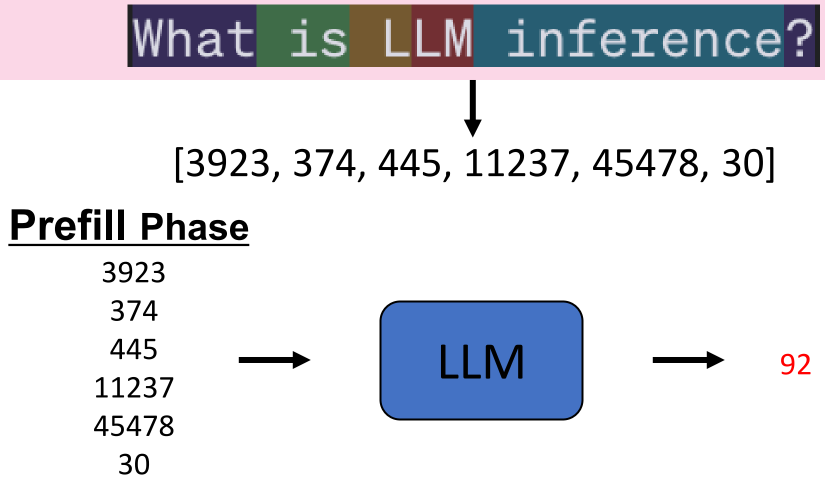
word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

Current merge table:

(empty)

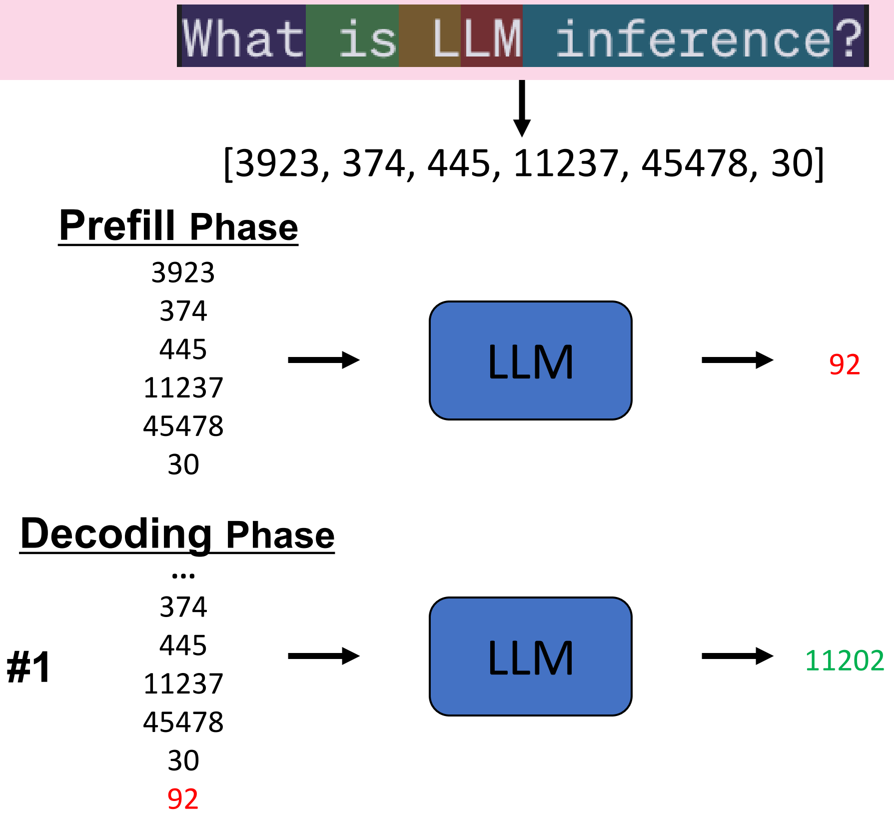
LLM Inference Procedure

- **Prefill Phase** (Single-step Phase)
  - Running the tokenized prompt through the LLM Model to generate the first token



LLM Inference Procedure

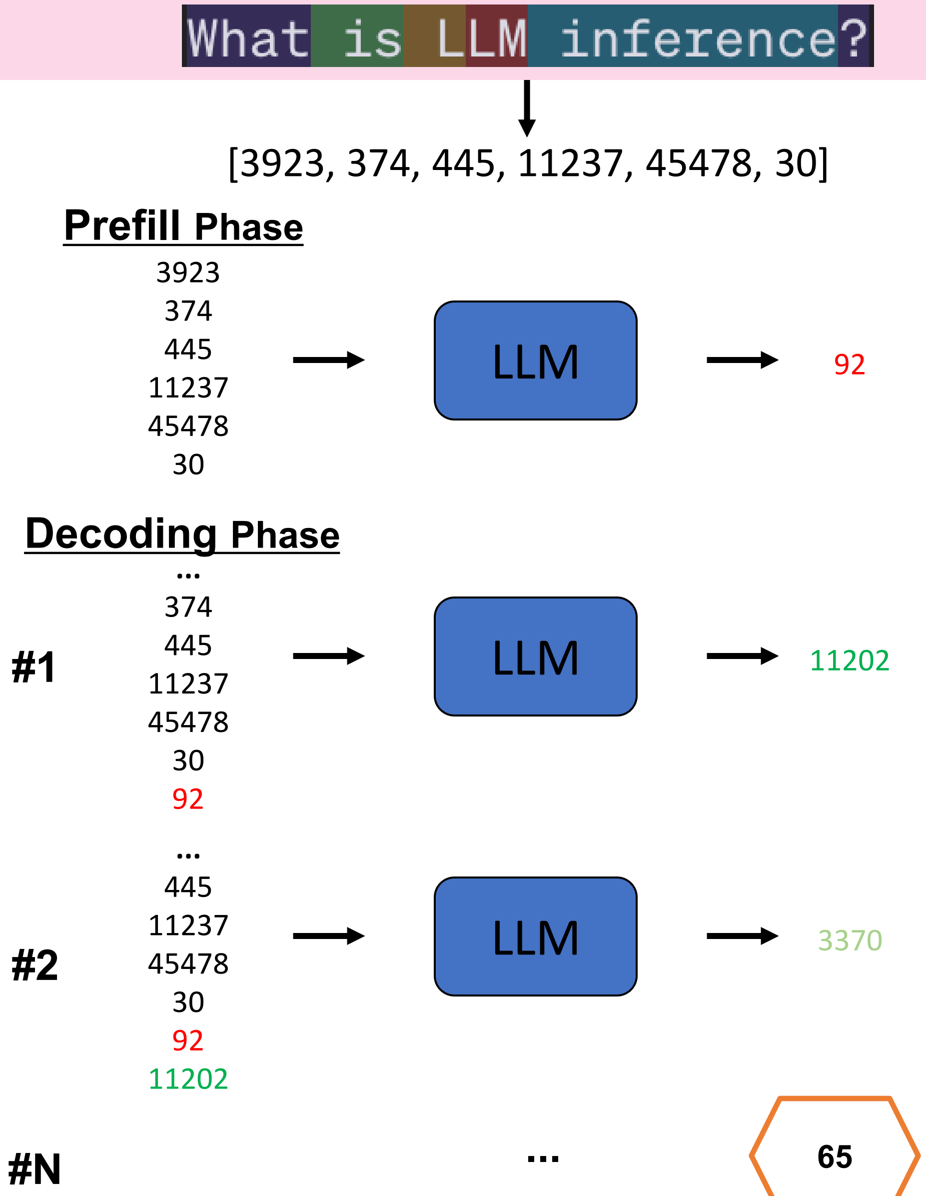
- **Prefill Phase** (Single-step Phase)
  - Running the tokenized prompt through the LLM Model to generate the first token
- **Decoding Phase** (Multi-step Phase)
  - Appending the generated token to the sequence of input tokens and using it as a new input to generate the next token



LLM Inference Procedure

- **Prefill Phase** (Single-step Phase)
  - Running the tokenized prompt through the LLM Model to generate the first token
- **Decoding Phase** (Multi-step Phase)
  - Appending the generated token to the sequence of input tokens and using it as a new input to generate the next token

Repeat decoding until meeting a stopping criteria
- Generating end-of-sequence token
- Reaching maximum sequence length



LLM Inference Scenarios

- **Inference** - Fewer request, offline traffic, latency

  Take a series of tokens as inputs, and generate subsequent tokens autoregressively until they meet a stopping criteria

  - Prefill Phase (Process the input)
  - Decoding Phase (Generate the output)

- **Serving** - Many requests, online traffic, cost-per-query
  - Co-locate the two phases and batch the computation of prefill and decoding across all users and requests

Break

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY

Multimodal Representation and Efficiency of Foundation AI Models

Wei Wen, Research Scientist, Meta

Duration: 1 hour

THE CHIPS TO SYSTEMS CONFERENCE

Outline – Two Main Parts

- Multimodal Representation Techniques
  - Multimodal Taxonomy
  - Multimodal Understanding
  - Multimodal Generation
- Efficiency of Large Foundation Models
  - Quantization
  - Low rank
  - Sparsity / pruning
  - Parallelism
  - Linear-Time Sequence Modeling

Multimodal Representation Techniques

- Multimodal Taxonomy in this Tutorial
  - Image Understanding: image & text in, text out
  - Image Generation: image & text in, image & text out
- Multimodal Understanding
  - Modeling: Llava, Flamingo, etc
  - Vision Encoders: CLIP, MetaCLIP
- Multimodal Generation
  - Autoregressive multimodal generation
  - Diffusion and Modeling Unification

62

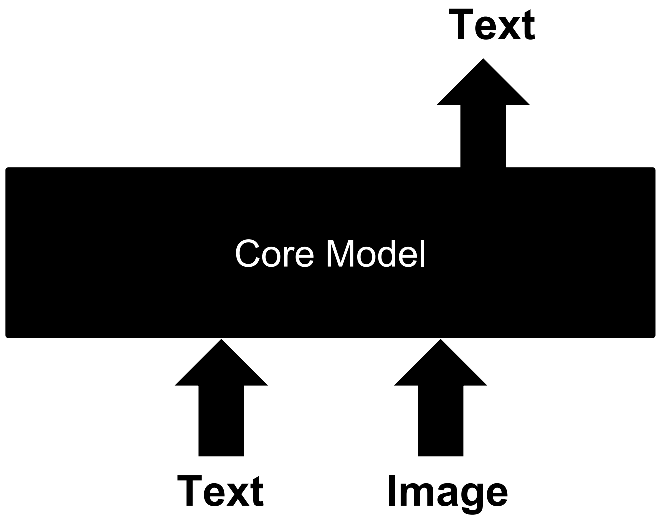
Multimodal Representation Techniques

- Multimodal Taxonomy in this Tutorial
  - Image Understanding: image & text in, text out
  - Image Generation: image & text in, image & text out
- Multimodal Understanding
  - Modeling: Llava, Flamingo, etc
  - Vision Encoders: CLIP, MetaCLIP
- Multimodal Generation
  - Autoregressive multimodal generation
  - Diffusion and Modeling Unification

Multimodal Taxonomy in this Tutorial

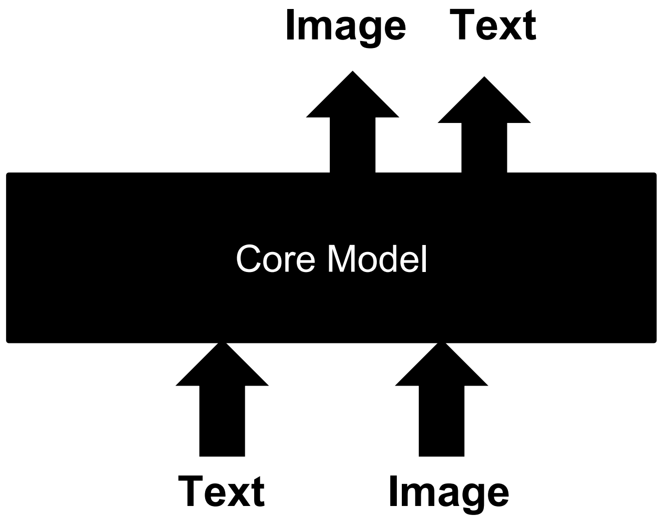
- Focus on image and text modes only

Understanding



- Classification
- VQA
- Captioning
- Any tasks in text as outputs

Generation



- ChatGPT 4o Image Generation

Multimodal Representation Techniques

- Multimodal Taxonomy in this Tutorial
  - Image Understanding: image & text in, text out
  - Image Generation: image & text in, image & text out
- Multimodal Understanding
  - Modeling: Llava, Flamingo, etc
  - Vision Encoders: CLIP, MetaCLIP
- Multimodal Generation
  - Autoregressive multimodal generation
  - Diffusion and Modeling Unification

Multimodal Understanding -- Modeling

- Mainstream model architecture

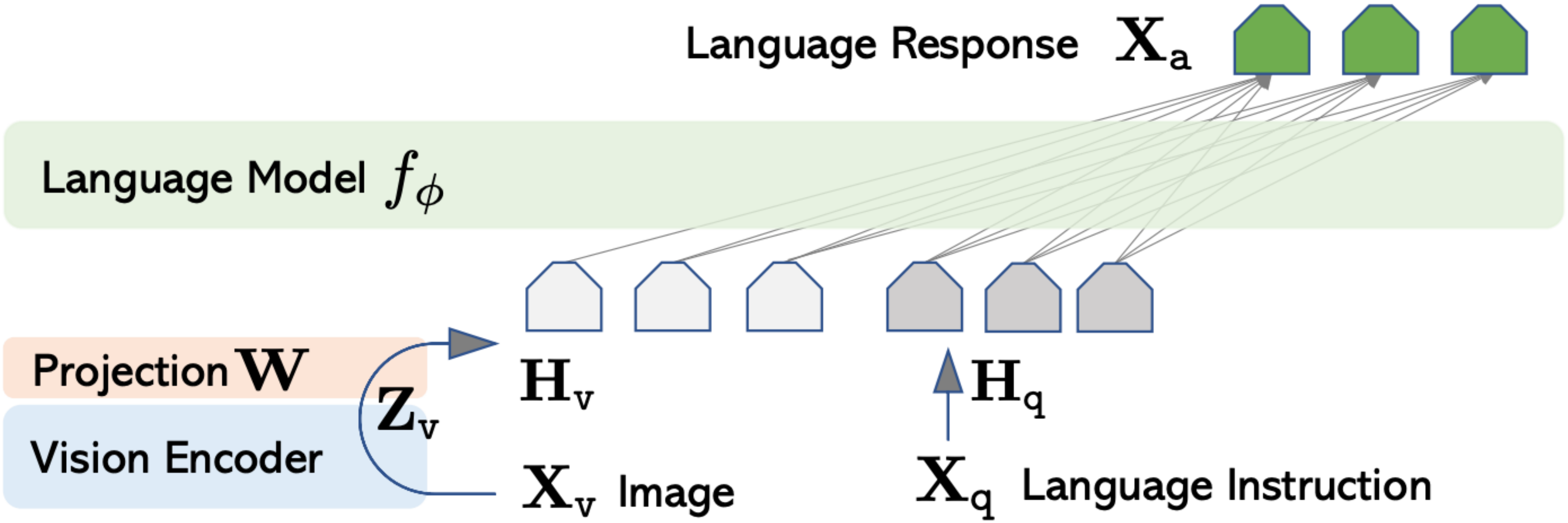


Figure 1: LLaVA network architecture.

Liu, H., Li, C., Wu, Q., & Lee, Y. J. (2023). Visual instruction tuning. *Advances in neural information processing systems*, *36*, 34892-34916.

Multimodal Understanding -- Flamingo

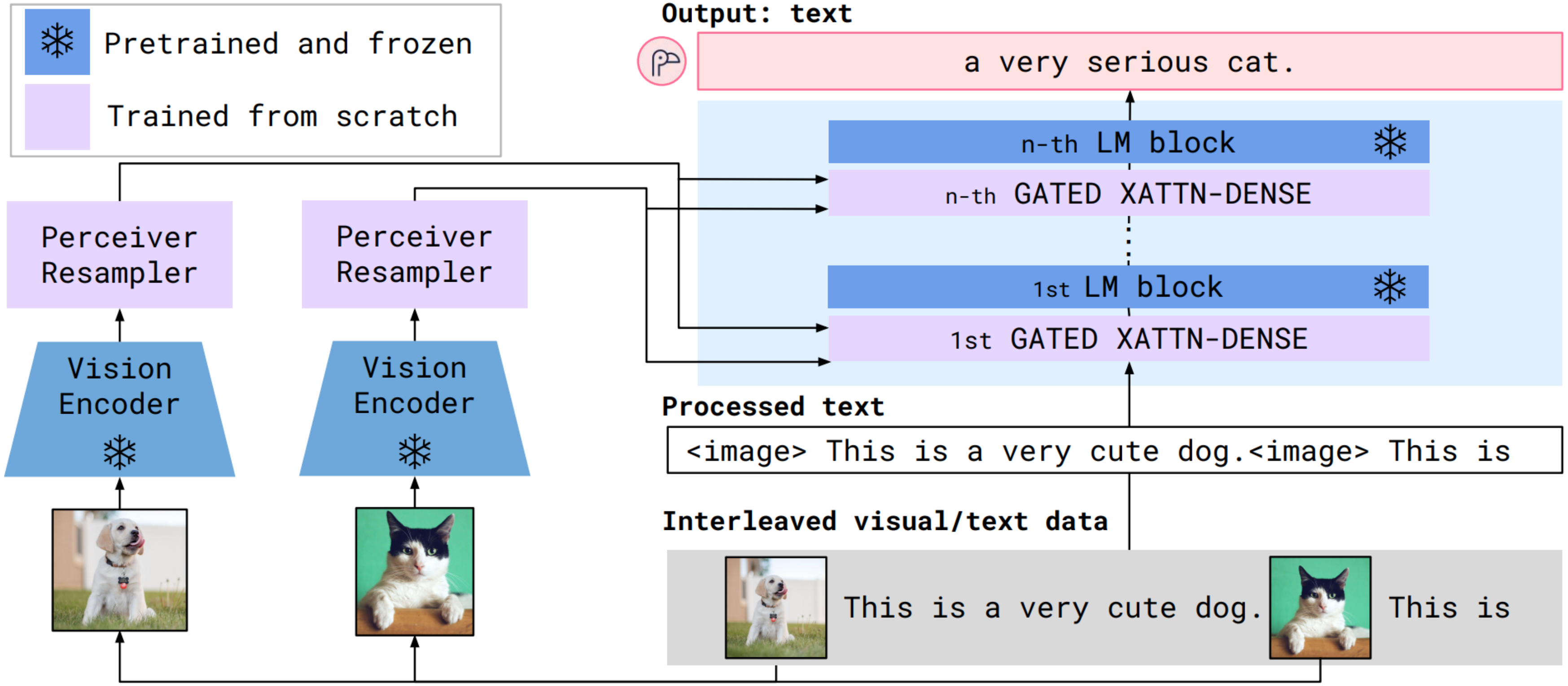
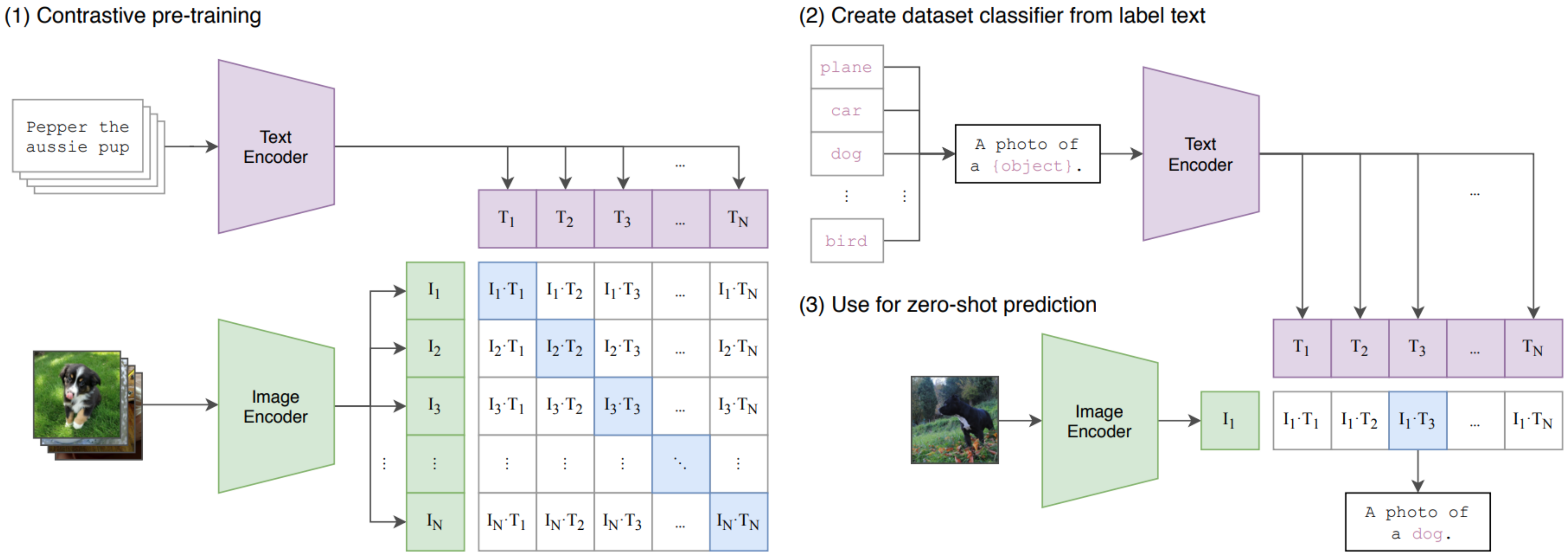


Figure 3: **Flamingo architecture overview.** Flamingo is a family of visual language models (VLMs) that take as input visual data interleaved with text and produce free-form text as output.

Alayrac, J. B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., ... & Simonyan, K. (2022). Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, *35*, 23716-23736.

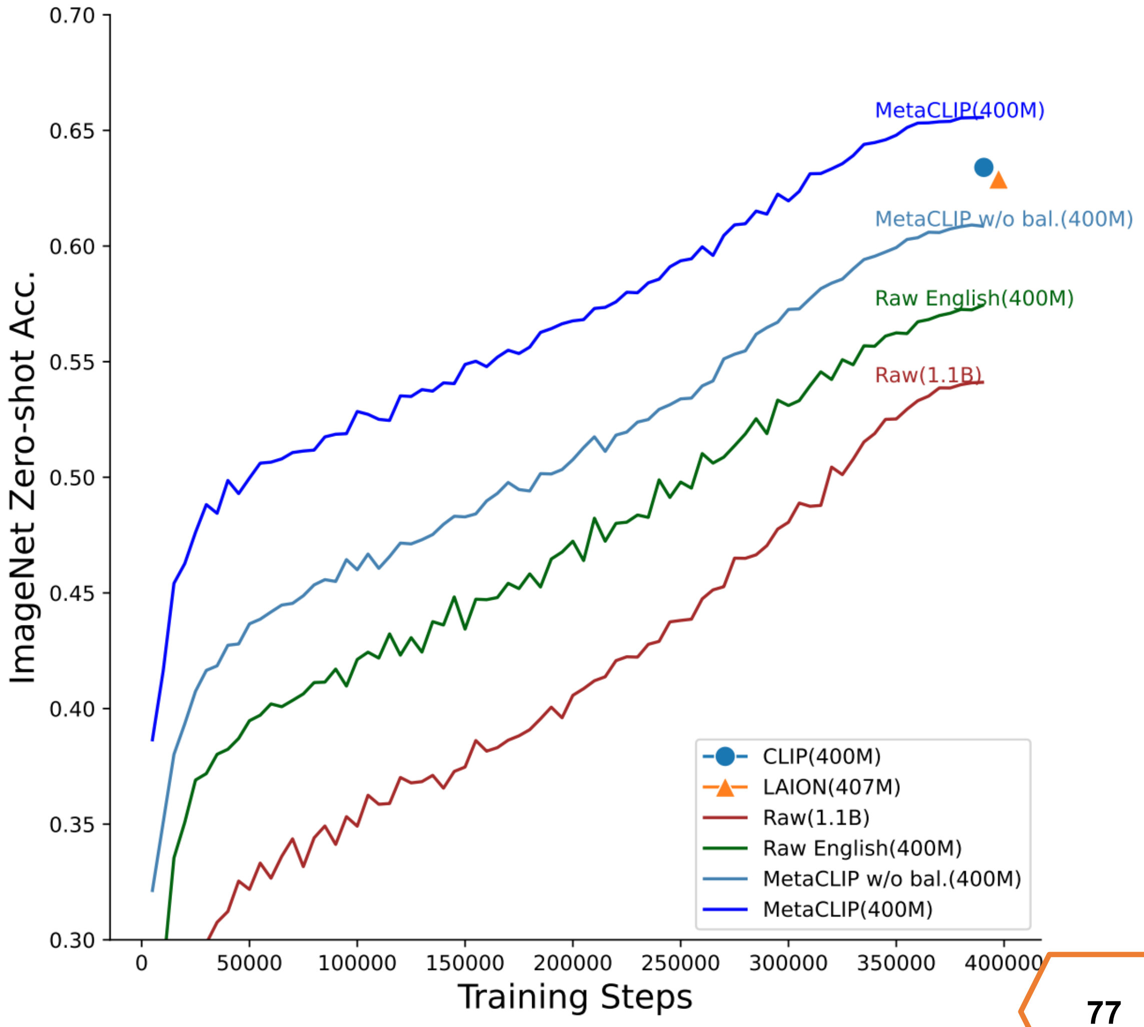
Multimodal Representations -- CLIP



Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021, July). Learning transferable visual models from natural language supervision. In *International conference on machine learning* (pp. 8748-8763). PmLR.

Multimodal Representations -- MetaCLIP

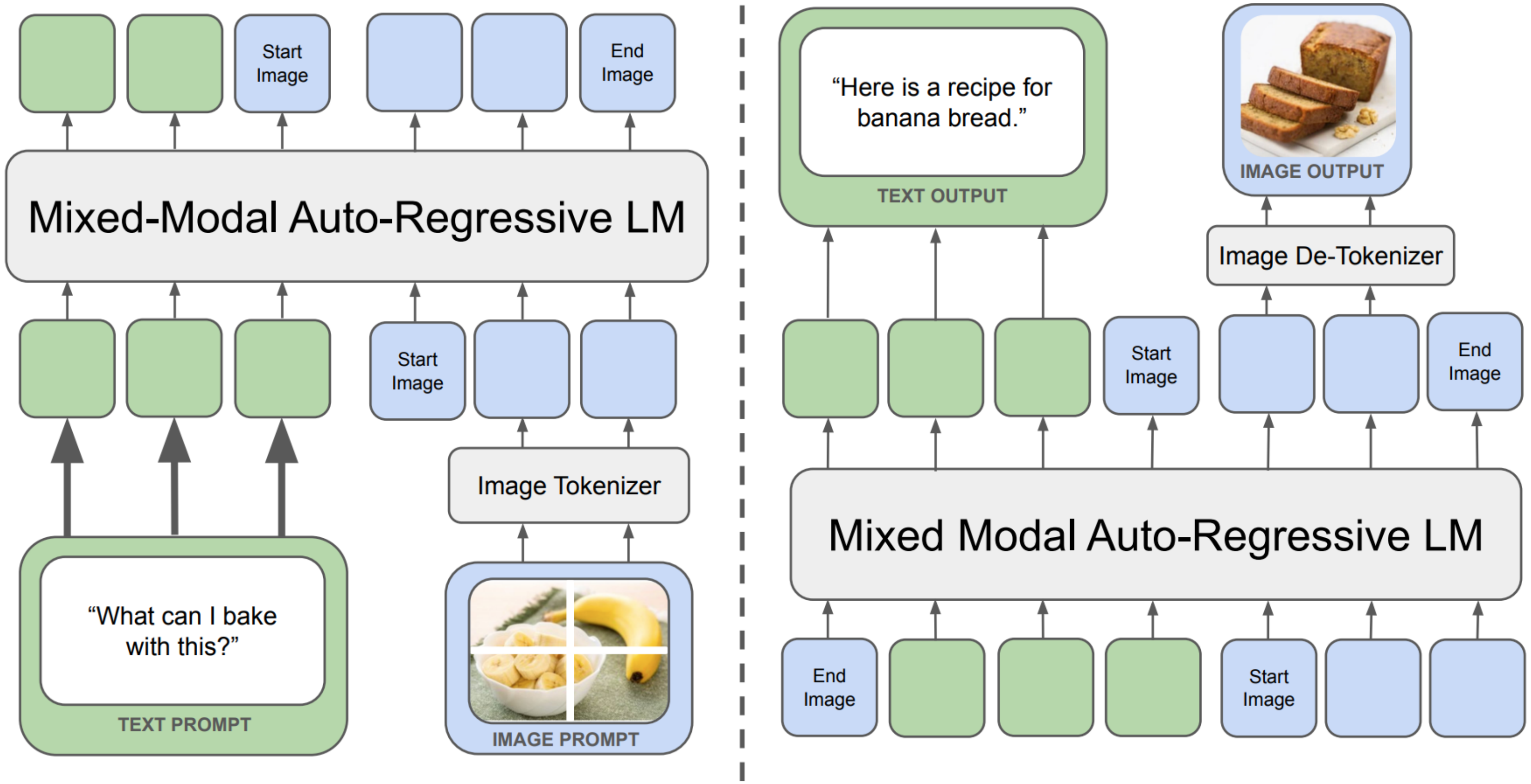
- MetaCLIP:
  - More transparent data curation with better models
  - “Released our training data distribution”



Multimodal Representation Techniques

- Multimodal Taxonomy in this Tutorial
  - Image Understanding: image & text in, text out
  - Image Generation: image & text in, image & text out
- Multimodal Understanding
  - Modeling: Llava, Flamingo, etc
  - Vision Encoders: CLIP, MetaCLIP
- Multimodal Generation
  - Autoregressive multimodal generation
  - Diffusion and Modeling Unification

Multimodal Generation – Autoregressive Generation



(a) Mixed-Modal Pre-Training

(b) Mixed-Modal Generation

Team, C. (2024). Chameleon: Mixed-modal early-fusion foundation models. *arXiv preprint arXiv:2405.09818*.

Image Tokenization: VQ-VAE

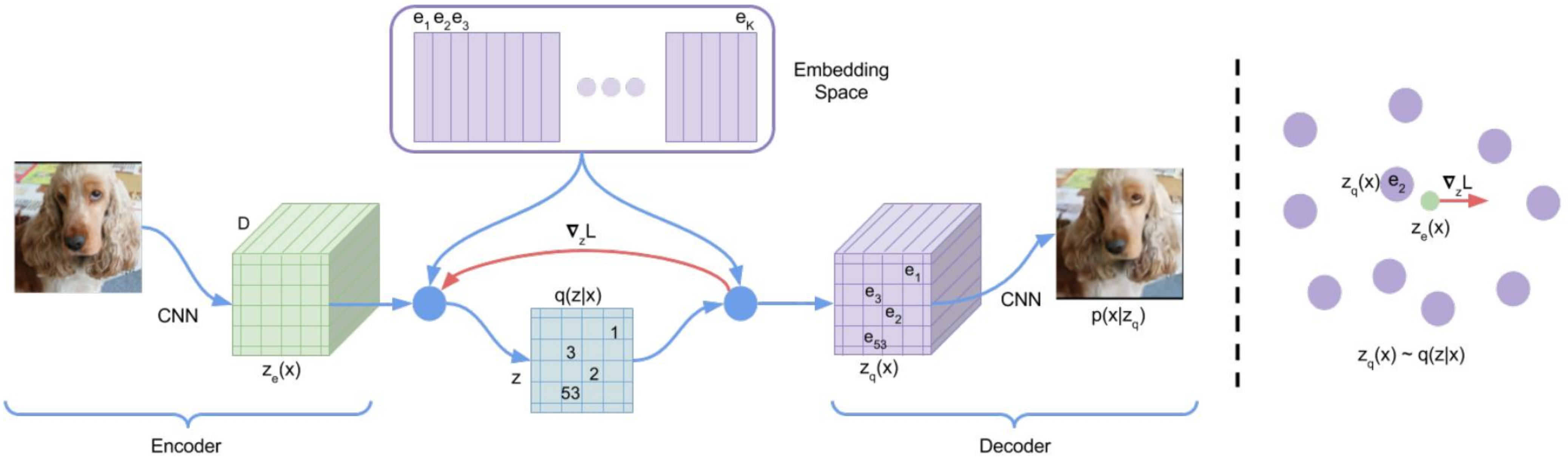
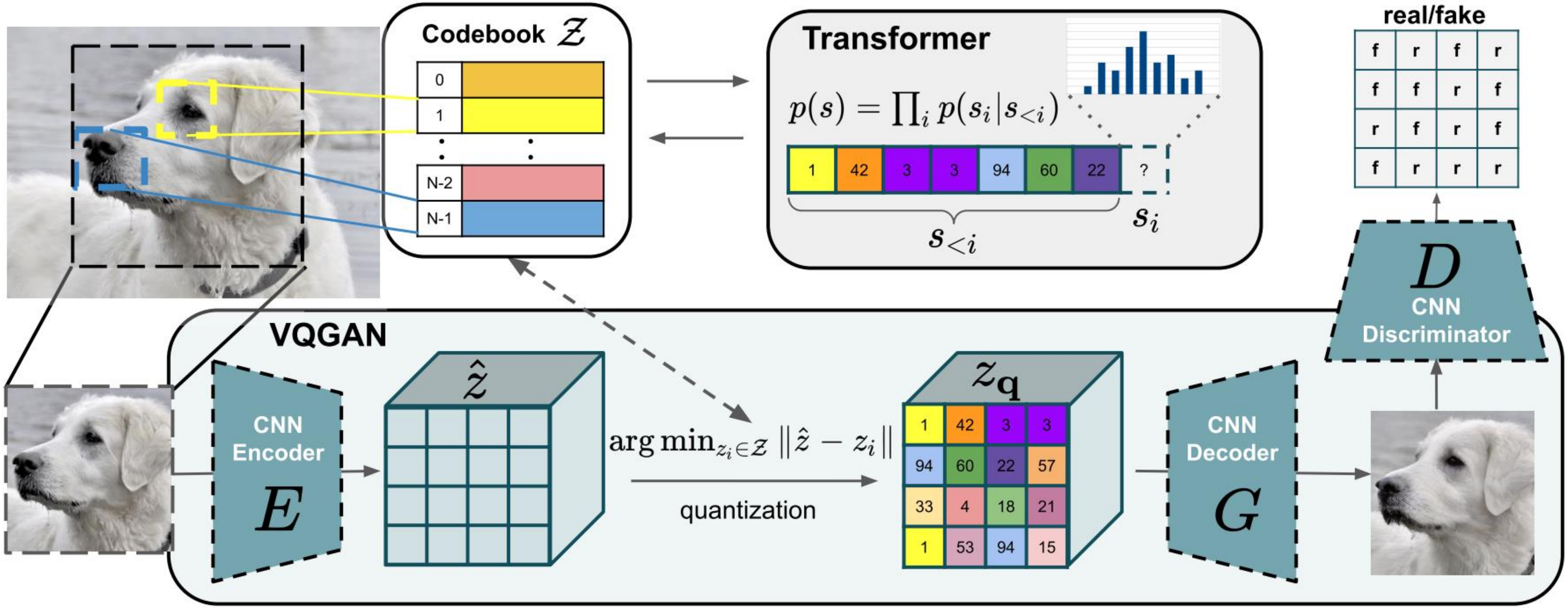


Figure 1: Left: A figure describing the VQ-VAE. Right: Visualisation of the embedding space. The output of the encoder $z(x)$ is mapped to the nearest point e_2. The gradient $\nabla_z L$ (in red) will push the encoder to change its output, which could alter the configuration in the next forward pass.

Van Den Oord, A., et al., (2017). Neural discrete representation learning. NeurIPS.

Image Tokenization: VQ-GAN



Esser, P., et al., (2021). Taming transformers for high-resolution image synthesis. CVPR.

Diffusion and Modeling Unification

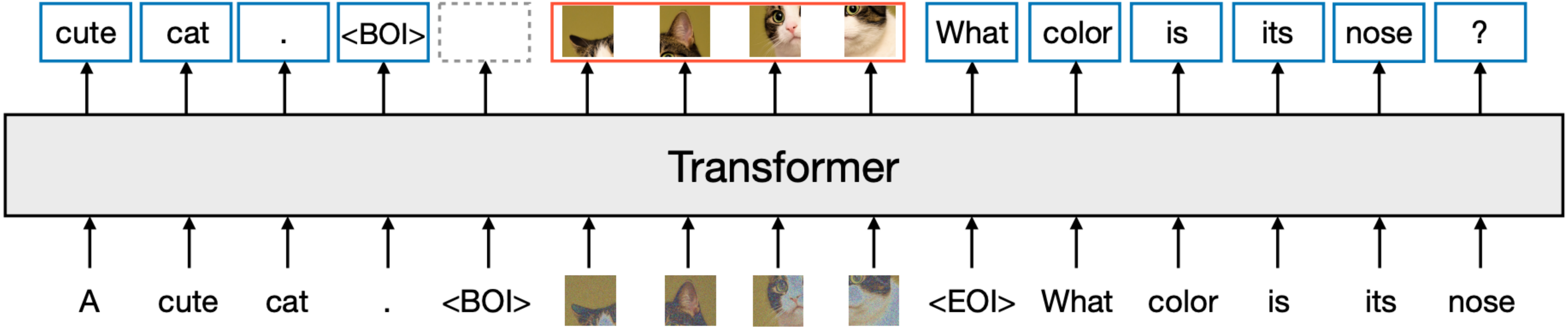


Figure 1: A high-level illustration of Transfusion. A single transformer perceives, processes, and produces data of every modality. Discrete (text) tokens are processed autoregressively and trained on the next token prediction objective. Continuous (image) vectors are processed together in parallel and trained on the diffusion objective. Marker BOI and EOI tokens separate the modalities.

Zhou, C., Yu, L., Babu, A., Tirumala, K., Yasunaga, M., Shamis, L., ... & Levy, O. (2024). Transfusion: Predict the next token and diffuse images with one multi-modal model. *arXiv preprint arXiv:2408.11039*.

Text Diffusion (and Multimodal Diffusion)

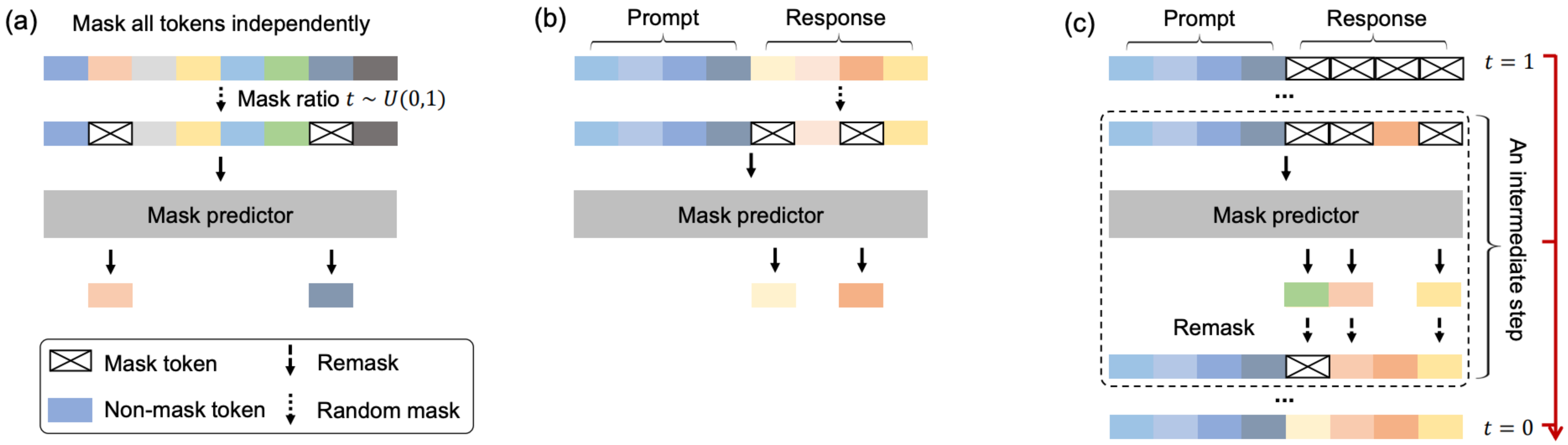


Figure 2. **A Conceptual Overview of LLaDA.** (a) Pre-training. LLaDA is trained on text with random masks applied independently to all tokens at the same ratio $t \sim U[0, 1]$. (b) SFT. Only response tokens are possibly masked. (c) Sampling. LLaDA simulates a diffusion process from $t = 1$ (fully masked) to $t = 0$ (unmasked), predicting all masks simultaneously at each step with flexible remask strategies.

Efficiency of Large Foundation Models

- Quantization
  - QAT, Post-training Quantization, QLoRA, FP8 training
- Low rank
  - LoRA
- Sparsity / pruning
  - Non-structured, structured, 2:4, MOE
- Parallelism
  - Parallel decoding: Speculative Decoding, Text Diffusion
  - Parallel Training: TP, PP, EP, CP, DP
- Linear-Time Sequence Modeling
  - Linear Transformer, xLSTM, Mamba

Efficiency of Large Foundation Models

- Quantization
  - QAT, Post-training Quantization, QLoRA, FP8 training
- Low rank
  - LoRA
- Sparsity / pruning
  - Non-structured, structured, 2:4, MOE
- Parallelism
  - Parallel decoding: Speculative Decoding, Text Diffusion
  - Parallel Training: TP, PP, EP, CP, DP
- Linear-Time Sequence Modeling
  - Linear Transformer, xLSTM, Mamba

Quantization for Efficiency – Taxonomy

Efficiency targeted phases

- Training efficiency: FP8 training
- Fine-tuning efficiency: QLoRA
- Inference efficiency:
  - Quantization-aware training
    - This is the go-to approach if accuracy is more important
    - Edge models are relatively small in practice, so the cost is acceptable
    - Straight-Through Estimator with grouping is a very strong baseline
  - Post-training Quantization
    - SpinQuant, SmoothQuant

Quantization – Basics

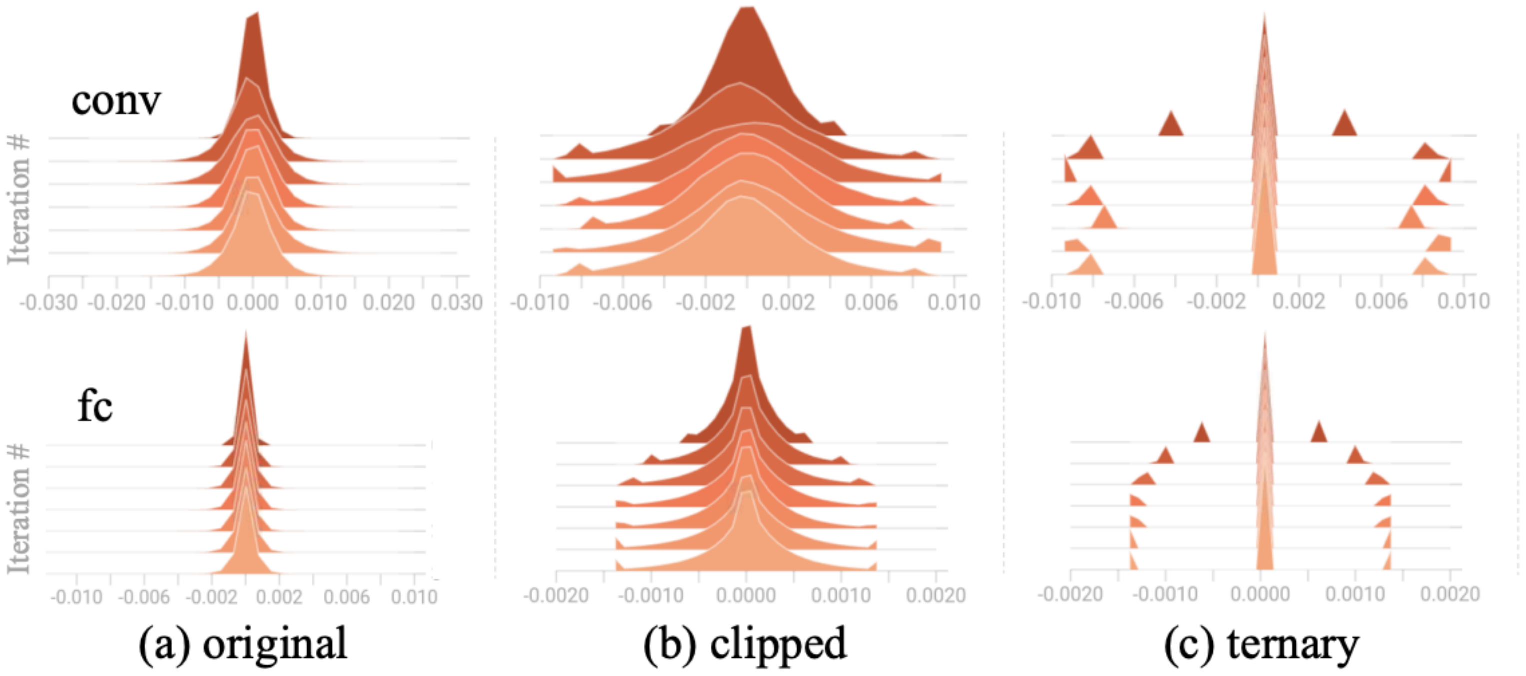
- Numerical bias
  - Deterministic rounding – bias in a quantization group, minimal/no bias in the final logit?
  - Stochastic rounding – no bias
- Numerical variance
  - Key problem!
  - Research focus: variance reduction
    - Constraining outlier scale
    - Grouping – if your group size is 1, quantization is floating-precision
      - A small group size (e.g. 32) can significantly reduce variance with minimal overhead

Quantization – Outlier Constraint

- Clipping
- Random rotation
- Rescaling
- ……

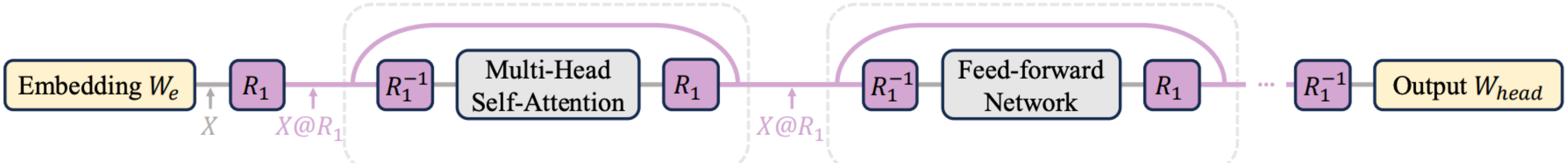
Quantization – Outlier Constraint: Clipping

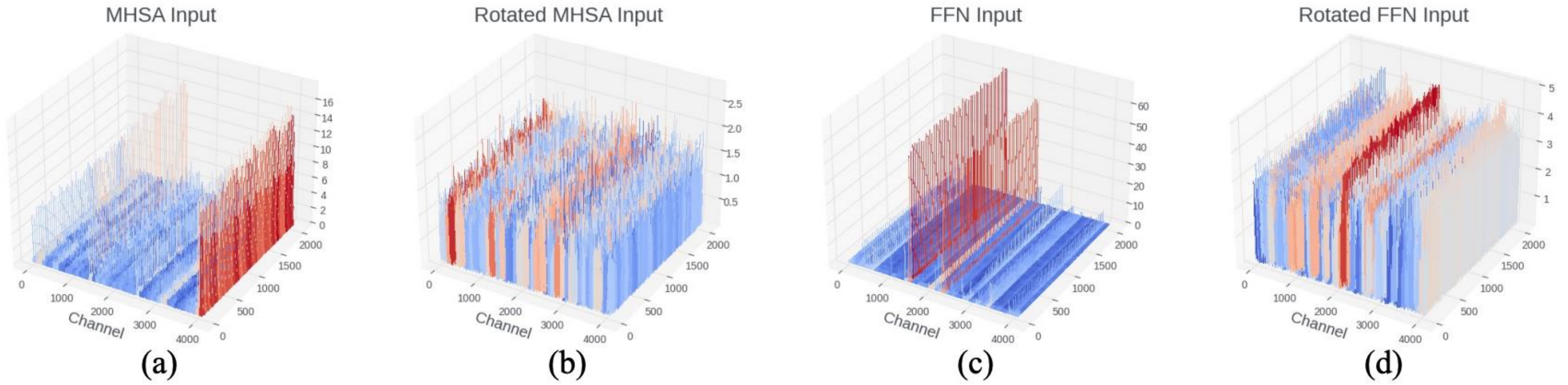
- **TernGrad: layer-wise clipping + grouping**



Wen, W., et al., (2017). Terngrad: Ternary gradients to reduce communication in distributed deep learning. *NeurIPS*.

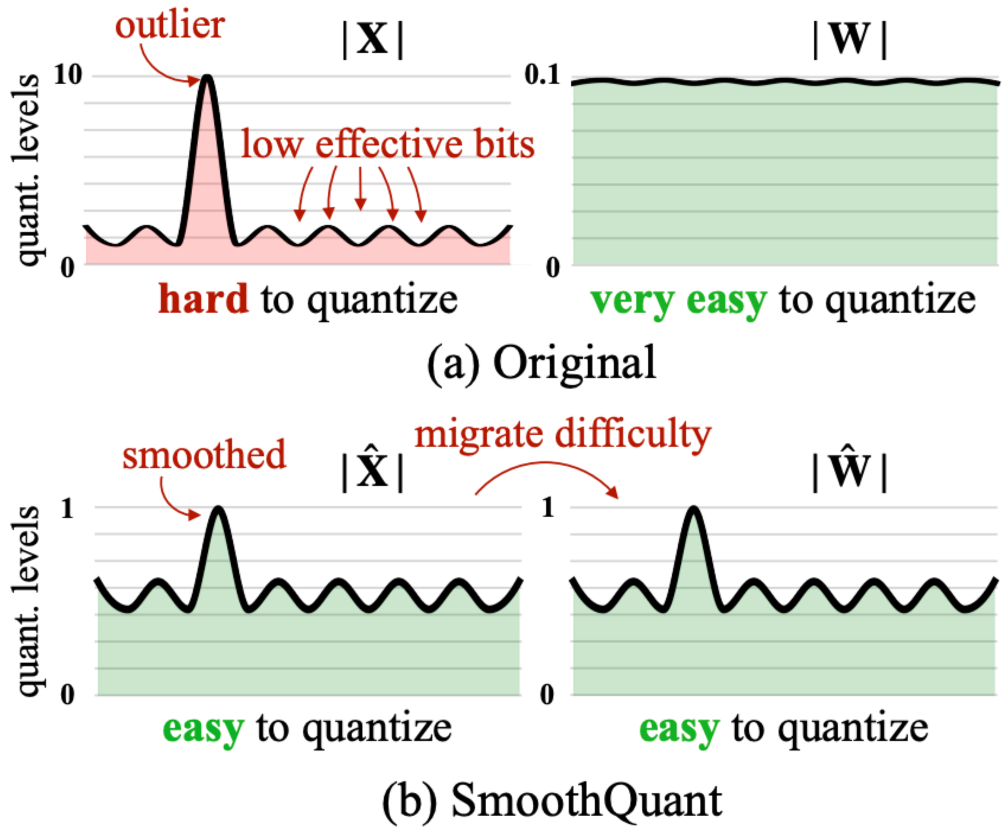
Quantization – Outlier Constraint: Rotation





Liu, Z., et al., (2024). Spinquant: LLM quantization with learned rotations. *arXiv:2405.16406*.

Quantization – Outlier Constraint: Rescaling



$$\mathbf{Y} = (\mathbf{X}\mathrm{diag}(\mathbf{s})^{-1}) \cdot (\mathrm{diag}(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}} \qquad (3)$$

Xiao, G.,et al., (2023). Smoothquant: Accurate and efficient post-training quantization for large language models. ICML.

Efficiency of Large Foundation Models

- Quantization
  - QAT, Post-training Quantization, QLoRA, FP8 training
- Low rank
  - LoRA
- Sparsity / pruning
  - Non-structured, structured, 2:4, MOE
- Parallelism
  - Parallel decoding: Speculative Decoding, Text Diffusion
  - Parallel Training: TP, PP, EP, CP, DP
- Linear-Time Sequence Modeling
  - Linear Transformer, xLSTM, Mamba

Low-rank + Quantization for Fine-tuning: QLoRA

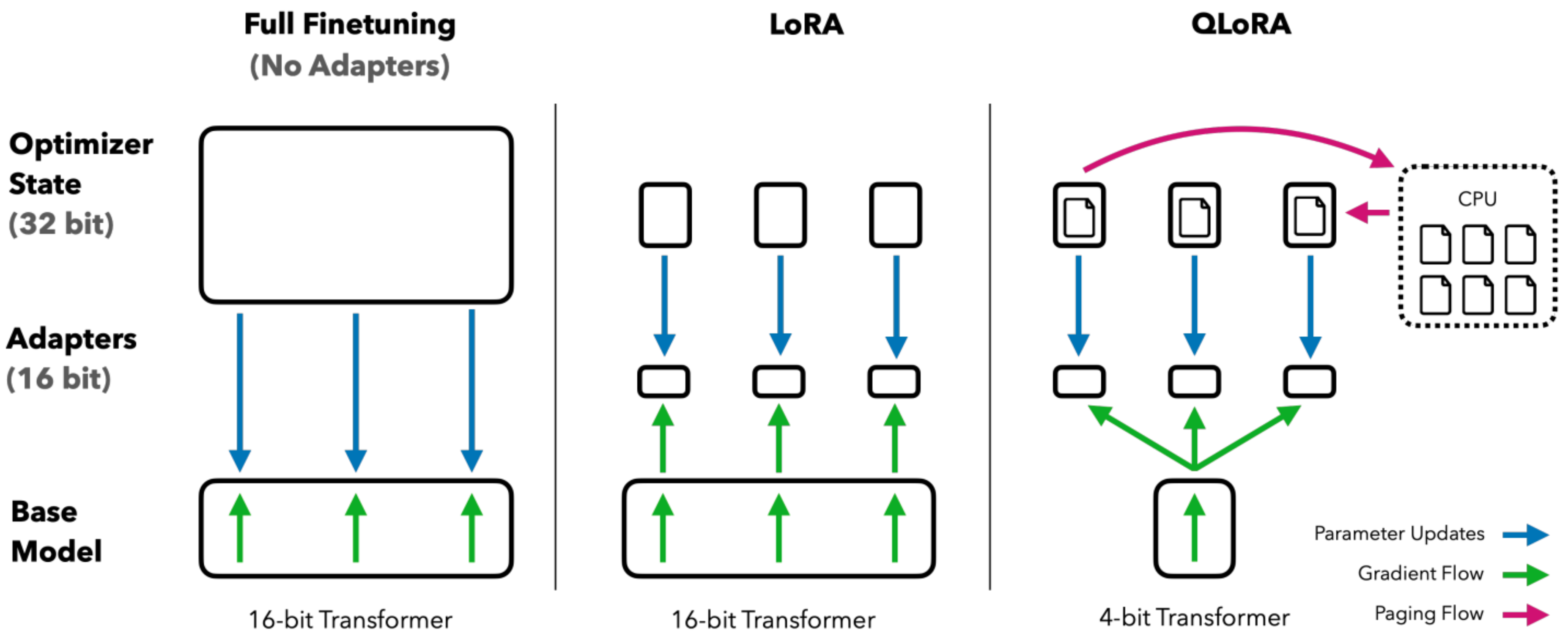


Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Dettmers, T., et al., (2023). Qlora: Efficient finetuning of quantized LLMs. *NeurIPS.*

Efficiency of Large Foundation Models

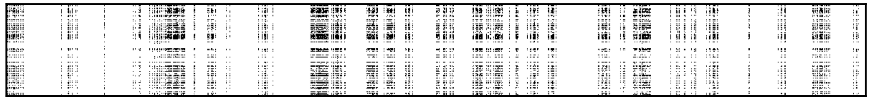
- Quantization
  - QAT, Post-training Quantization, QLoRA, FP8 training
- Low rank
  - LoRA
- Sparsity / pruning
  - Non-structured, structured, 2:4, MOE
- Parallelism
  - Parallel decoding: Speculative Decoding, Text Diffusion
  - Parallel Training: TP, PP, EP, CP, DP
- Linear-Time Sequence Modeling
  - Linear Transformer, xLSTM, Mamba

Sparsity / Pruning -- Patterns

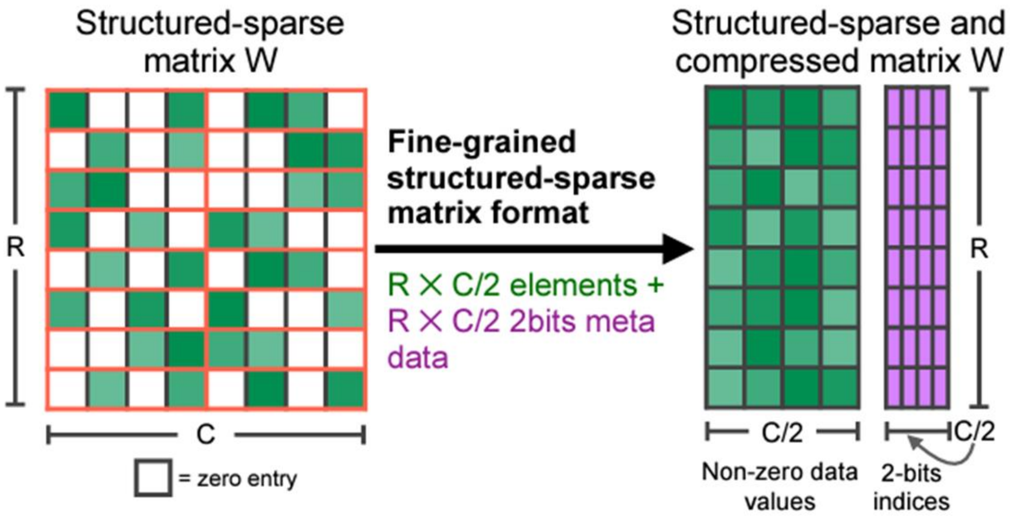
- Non-structured sparsity
  - Less popular because of computation inefficiency

Wen, W., Wu, C., Wang, Y., Chen, Y., & Li, H. (2016). Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, *29*.

- Structured sparsity
  - Remove weights group by group
  - Structured in a way with high compute efficiency
    - E.g. NVIDIA 2:4 sparsity



5.17X speedup

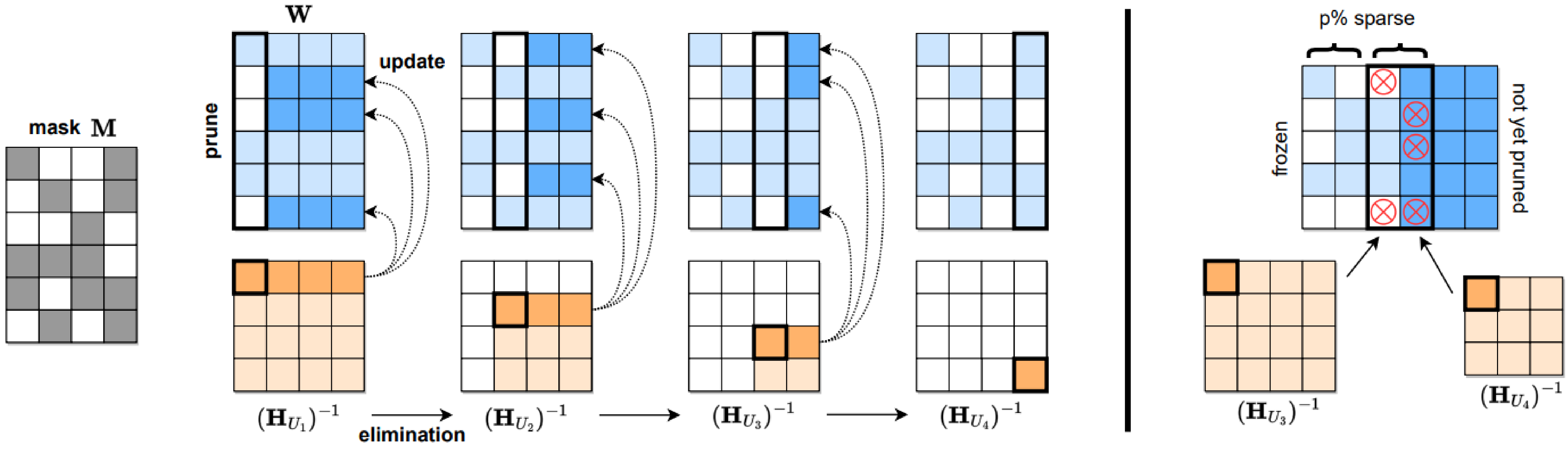


https://developer.nvidia.com/blog/structured-sparsity-in-the-nvidia-ampere-architecture-and-applications-in-search-engines/

Sparsity / Pruning -- Methods

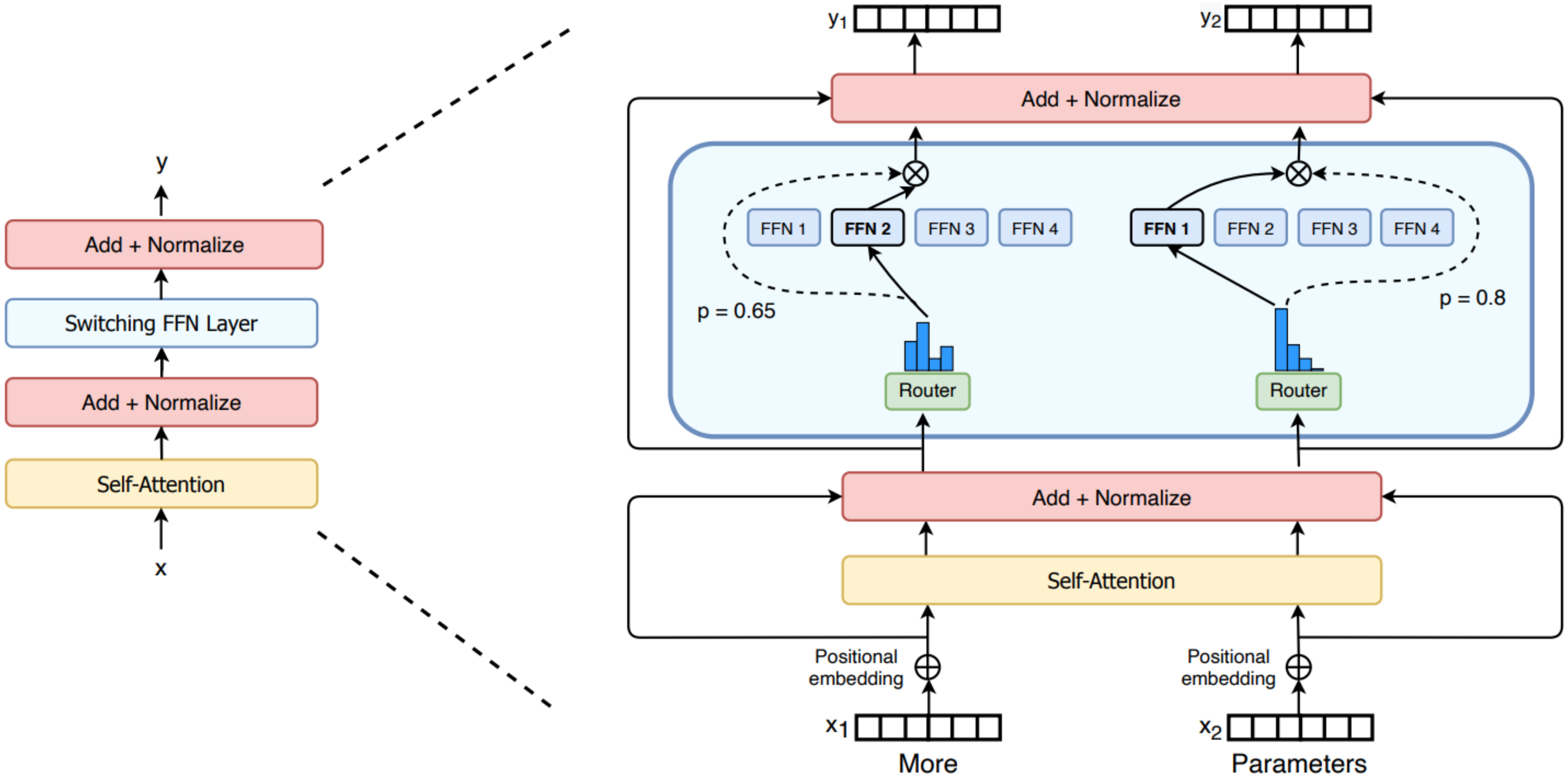
- Thresholding
- Regularization
- Optimizer

SparseGPT



Frantar, E., & Alistarh, D. (2023, July). Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning* (pp. 10323-10337). PMLR.

Natively Sparse Models: Mixture of Experts



Fedus, W., Zoph, B., & Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, *23*(120), 1-39.

Efficiency of Large Foundation Models

- Quantization
  - QAT, Post-training Quantization, QLoRA, FP8 training
- Low rank
  - LoRA
- Sparsity / pruning
  - Non-structured, structured, 2:4, MOE
- Parallelism
  - Parallel decoding: Speculative Decoding, Text Diffusion
  - Parallel Training: TP, PP, EP, CP, DP
- Linear-Time Sequence Modeling
  - Linear Transformer, xLSTM, Mamba

Parallelism

- Parallel decoding
  - Speculative Decoding
  - Text Diffusion
- Parallel Training
  - Data parallelism
    - Vanilla
    - ZeRO / FSDP sharding
  - Model parallelism
    - Tensor parallelism
    - Pipeline parallelism
    - Context parallelism
    - Expert parallelism

Speculative Decoding

https://research.google/blog/looking-back-at-speculative-decoding/

Algorithm 1 SpeculativeDecodingStep

Inputs: $M_p, M_q, prefix$.

$\triangleright$ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.

for $i = 1$ **to** γ **do**

  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$

  $x_i \sim q_i(x)$

end for

$\triangleright$ Run M_p in parallel.

$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$

  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$

$\triangleright$ Determine the number of accepted guesses n.

$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$

$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$

$\triangleright$ Adjust the distribution from M_p if needed.

$p'(x) \leftarrow p_{n+1}(x)$

if $n < \gamma$ **then**

  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$

end if

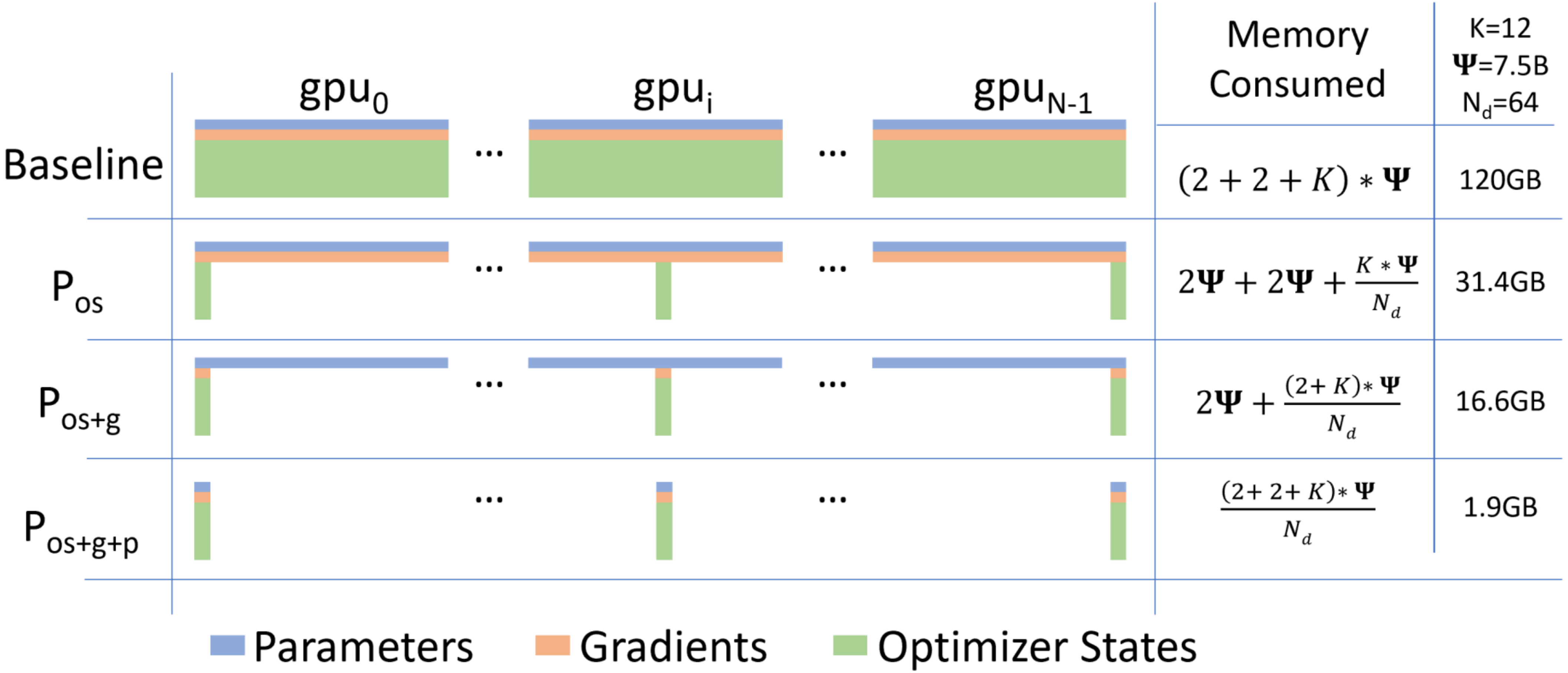
$\triangleright$ Return one token from M_p, and n tokens from M_q.

$t \sim p'(x)$

return $prefix + [x_1, \ldots, x_n, t]$

Follow-up works:
MEDUSA, EAGLE

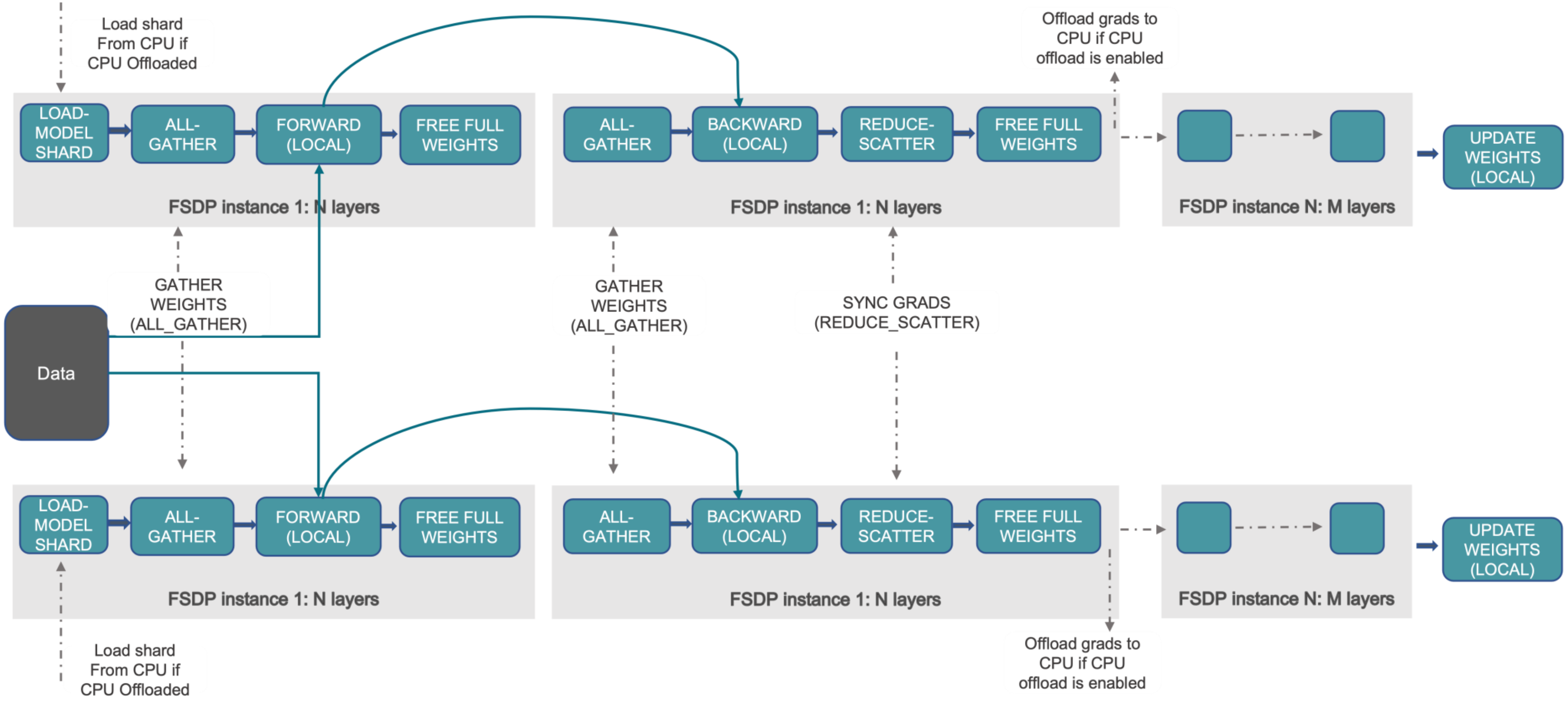
Data Parallelism – ZeRO (in DeepSpeed)



Rajbhandari, S., Rasley, J., Ruwase, O., & He, Y. (2020, November). Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 1-16). IEEE.

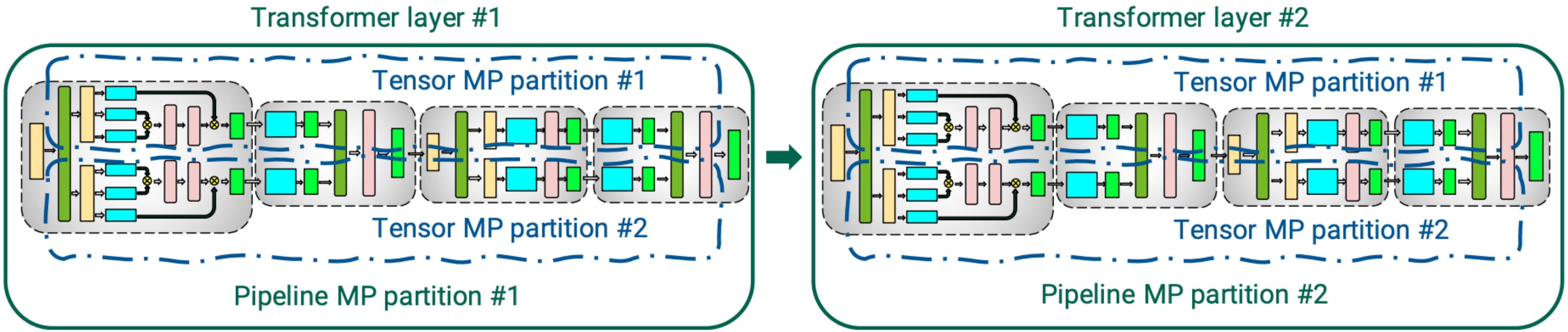
Data Parallelism – FSDP

- Fully Sharded Data Parallel (FSDP) -- A PyTorch implementation



https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html

Model Parallelism

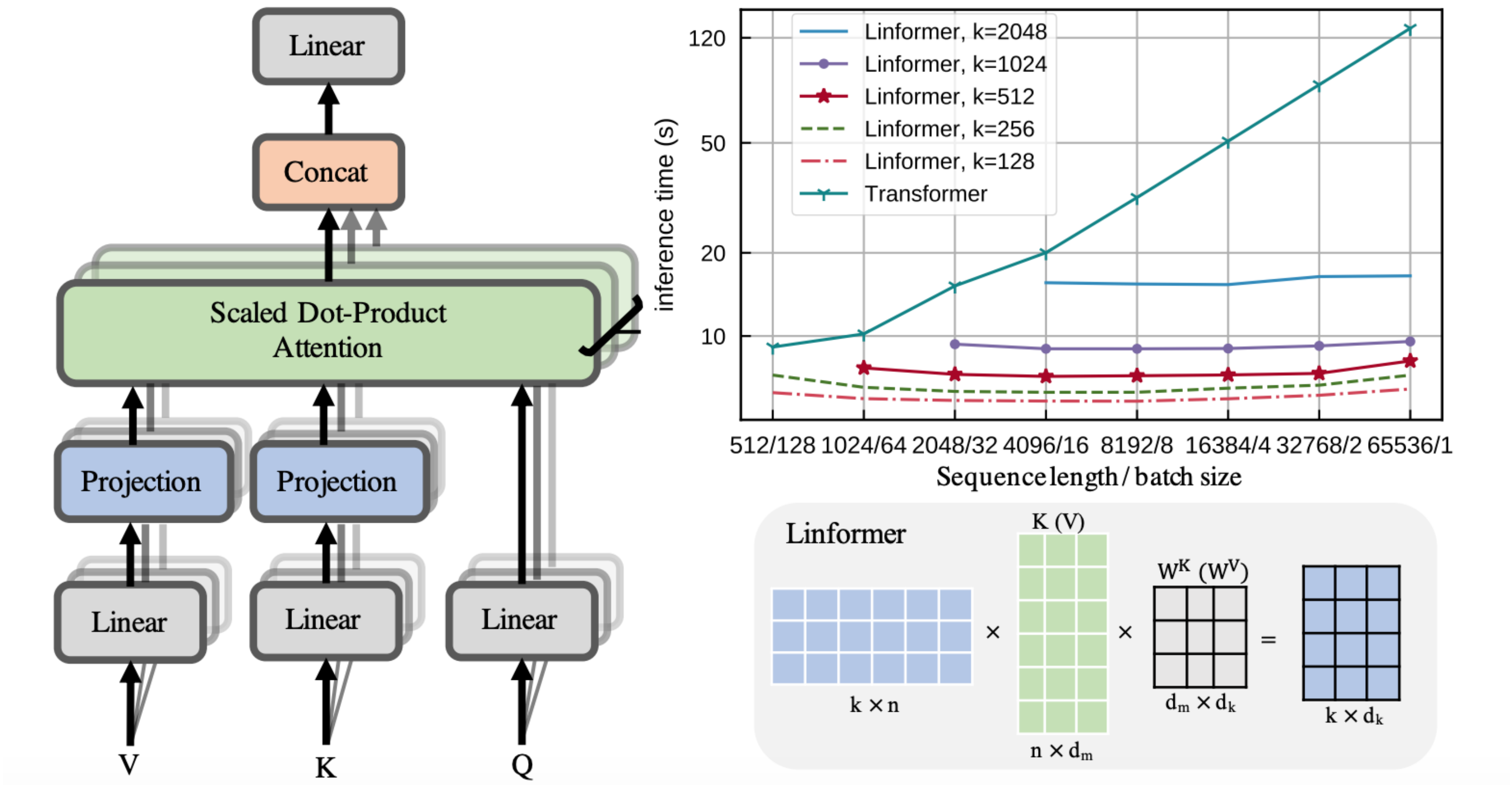


Narayanan, D., Shoeybi, M., Casper, J., LeGresley, P., Patwary, M., Korthikanti, V., ... & Zaharia, M. (2021, November). Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis* (pp. 1-15).

Efficiency of Large Foundation Models

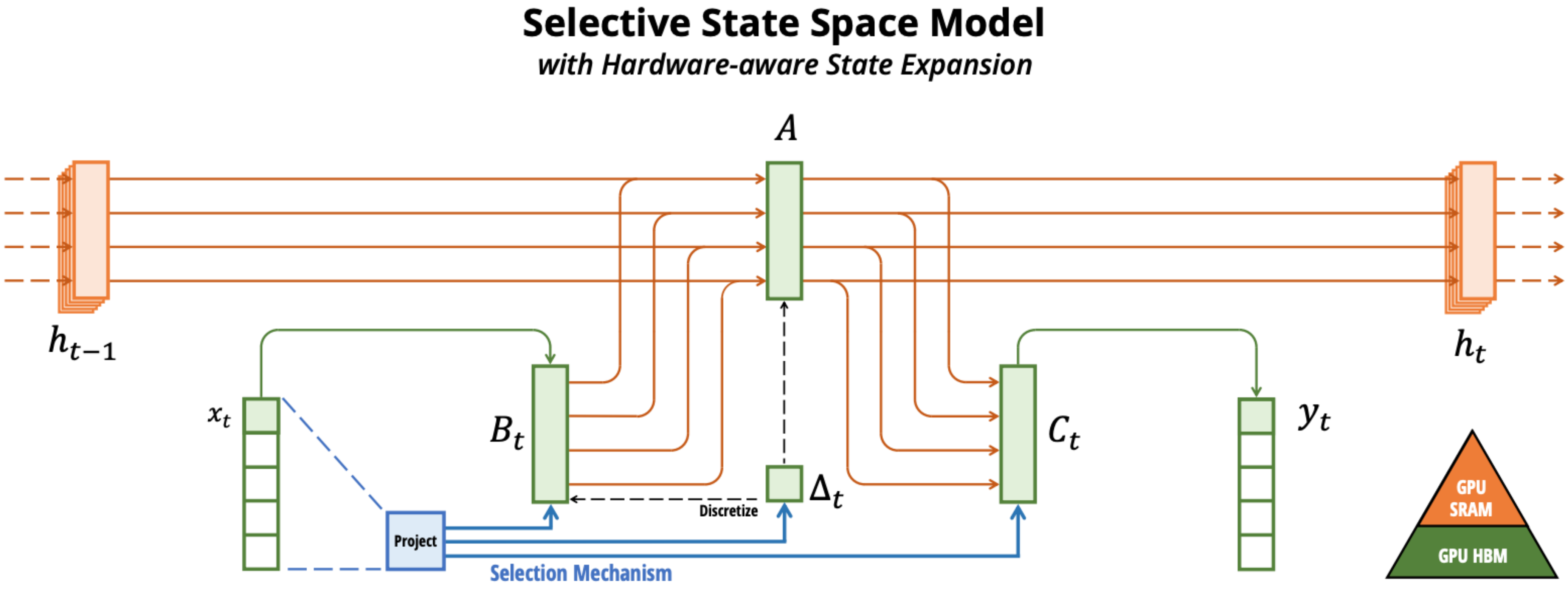
- Quantization
  - QAT, Post-training Quantization, QLoRA, FP8 training
- Low rank
  - LoRA
- Sparsity / pruning
  - Non-structured, structured, 2:4, MOE
- Parallelism
  - Parallel decoding: Speculative Decoding, Text Diffusion
  - Parallel Training: TP, PP, EP, CP, DP
- Linear-Time Sequence Modeling
  - Linear Transformer, xLSTM, Mamba

Linear-Time Sequence Modeling – Linear Transformer



Wang, S., Li, B. Z., Khabsa, M., Fang, H., & Ma, H. (2020). Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Linear-Time Sequence Modeling – Mamba



Gu, A., & Dao, T. (2023). Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*.

Linear-Time Sequence Modeling – xLSTM

- xLSTM: Extended Long Short-Term Memory

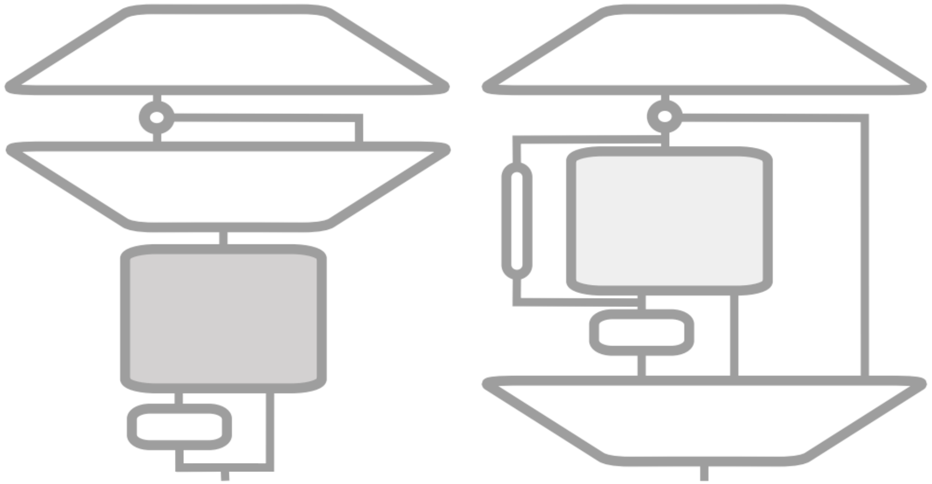


Figure 3: xLSTM blocks. **Left**: A residual sLSTM block with post up-projection (like Transformers): The input is fed into an sLSTM — with an optional convolution — followed by a gated MLP. **Right**: A residual mLSTM block with pre up-projection (like State Space models): mLSTM is wrapped inside two MLPs, via a convolution, a learnable skip connection, and an output gate that acts component-wise. See Figure 10 and Figure 11 in the appendix for details.

Break

THE CHIPS TO SYSTEMS CONFERENCE

SPONSORED BY CEDA sigda

Application of Foundation Models in EDA

Zhiyao Xie, Assistant Professor, HKUST

Duration: ~1 hour

THE CHIPS TO SYSTEMS CONFERENCE

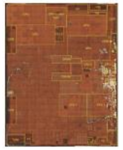
SPONSORED BY

Challenges in Delivering Better Chips

Increasing **IC design complexity**

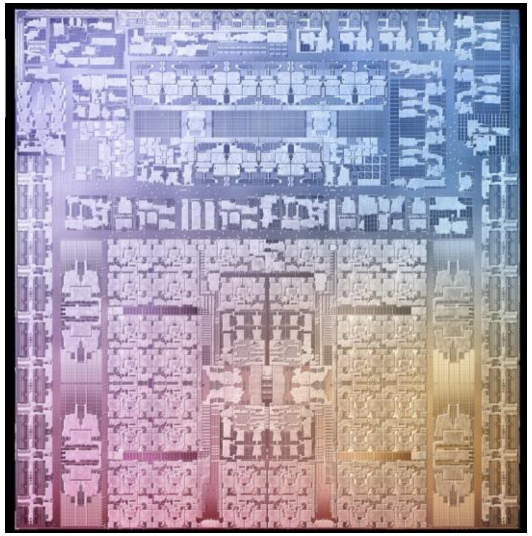
- Increasing **IC design cost**
- Increasing **time to market**

IC complexity



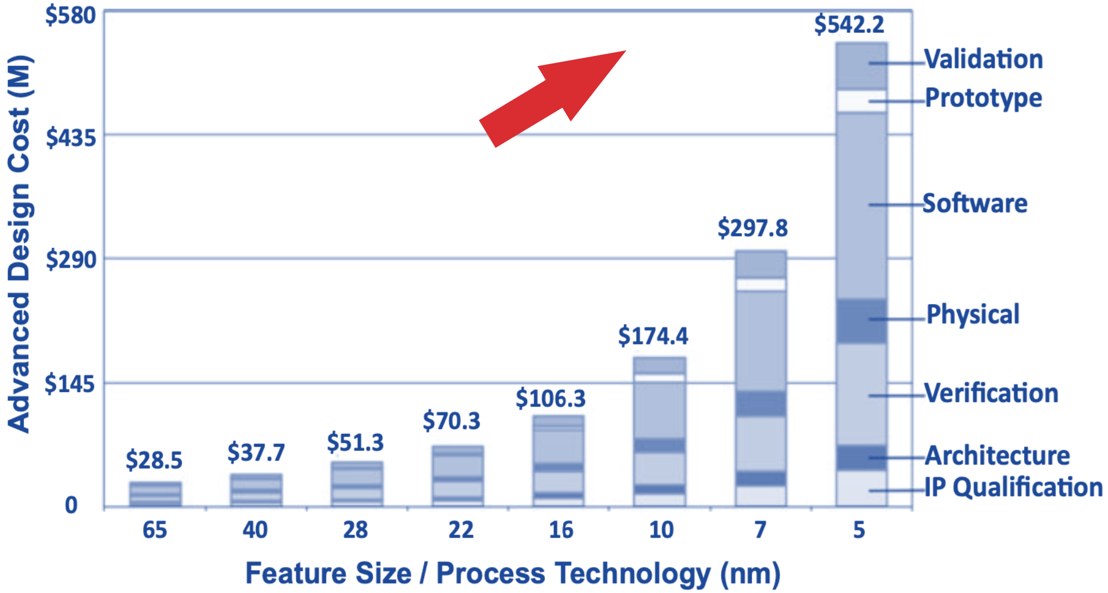
Apple A11
4B transistors

Apple A15
15B transistors



Apple M3 Max
92B transistors

IC Design Cost is Skyrocketing



(Not including manufacturing)

How AI Assists EDA - Our Taxonomy

Type I: **Supervised Predictive** AI Techniques for EDA

Type II: **Foundation** AI Techniques for EDA

(Circuit Foundation Model)

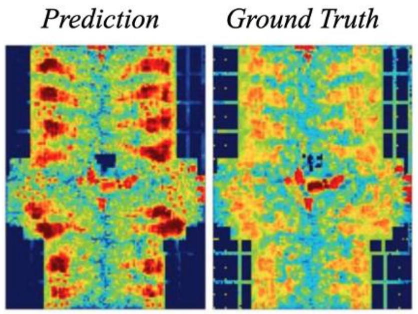
How AI Assists EDA - Our Taxonomy

Type I: **Supervised Predictive** AI Techniques for EDA

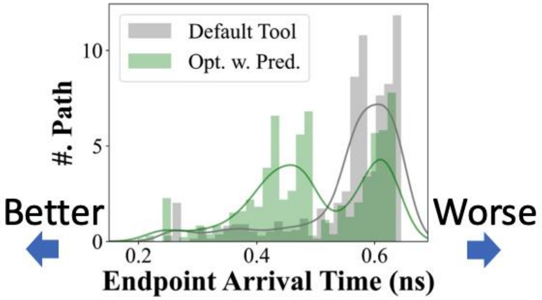
Type II: **Foundation** AI Techniques for EDA
(Circuit Foundation Model)

Explorations in Predictive AI Methods

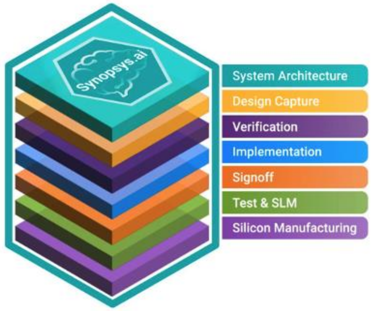
- Predictive AI supports many applications: both early evaluation & optimization



AI-Assisted IC Quality Prediction

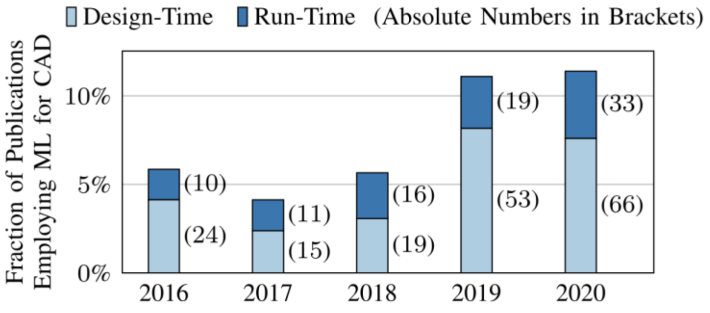


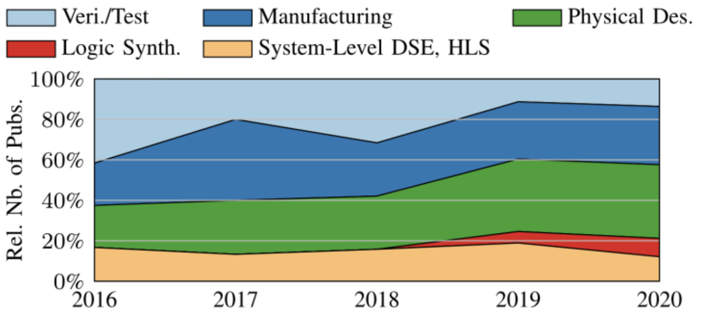
AI-Guided IC Optimization

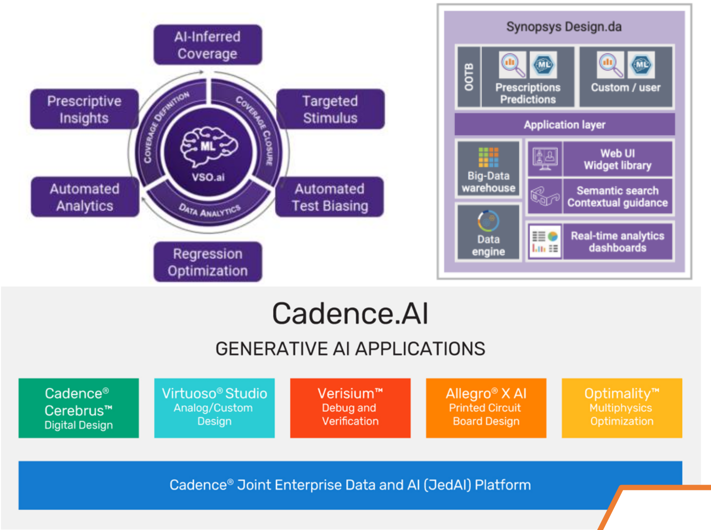


AI-Guided IC Design Space Exploration

- Explored in academia & industry, cover all stages

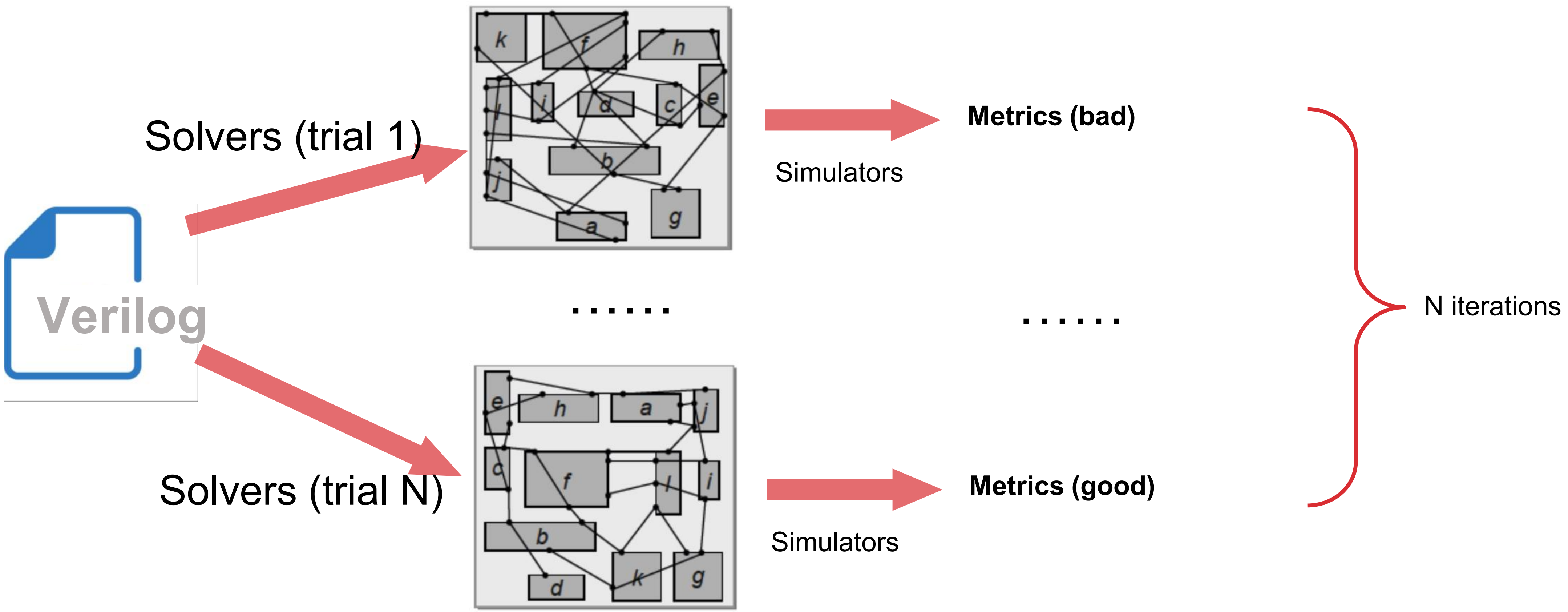






[1] Machine learning for electronic design automation: A survey. **ACM TODAES, 2021.**
[2] MLCAD: A survey of research in machine learning for CAD keynote paper. **IEEE TCAD, 2021.**

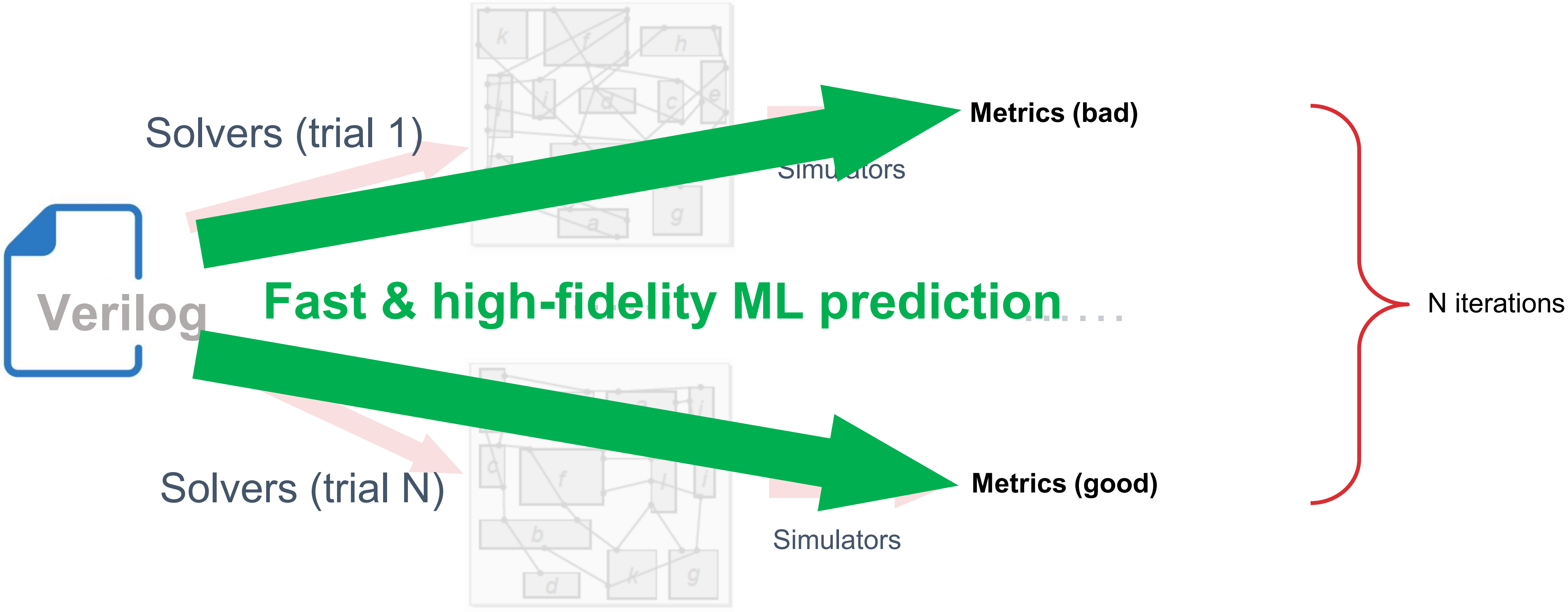
Predictive AI for EDA/Circuit Design



- Producing solutions **repeatedly from scratch**
- Why not learn from prior solutions? **More intelligence!**

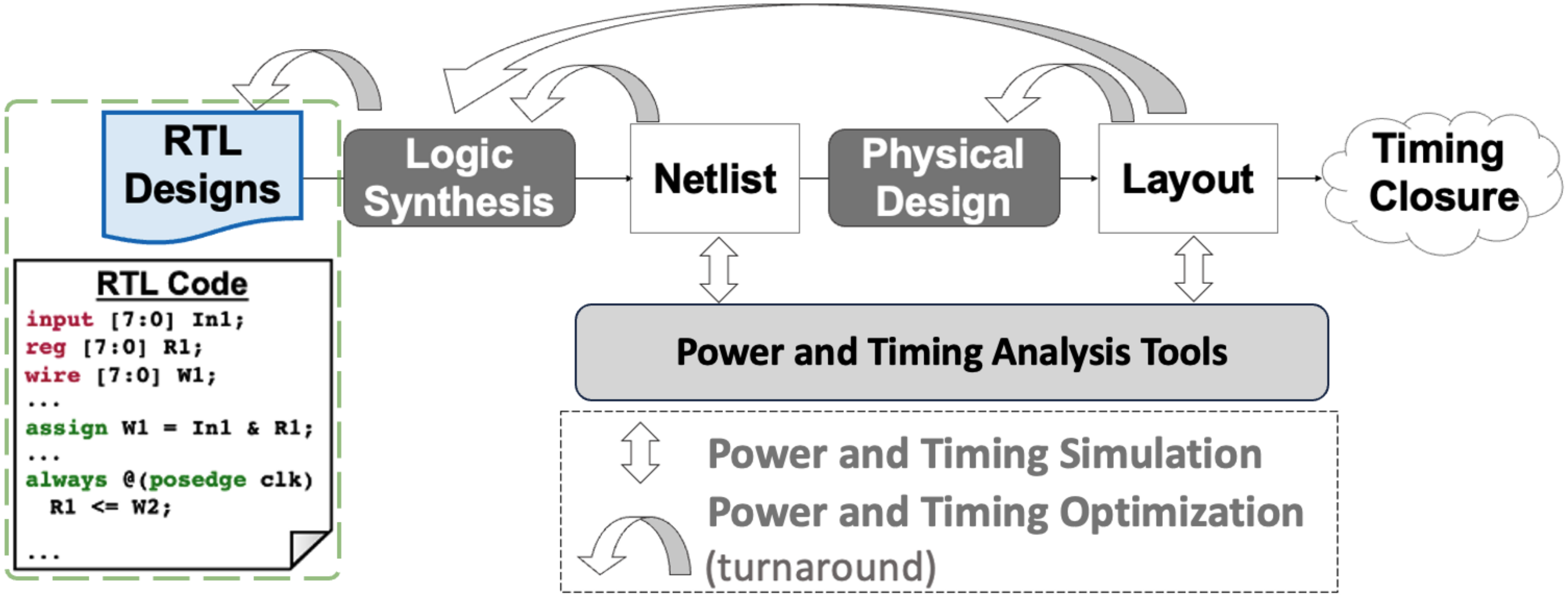
*Source: Kahng et al., VLSI physical design

Predictive AI for EDA/Circuit Design



- Why not learn from prior solutions? More intelligence!
- **ML in Electronic Design Automation: Early Timing and Power Modeling**

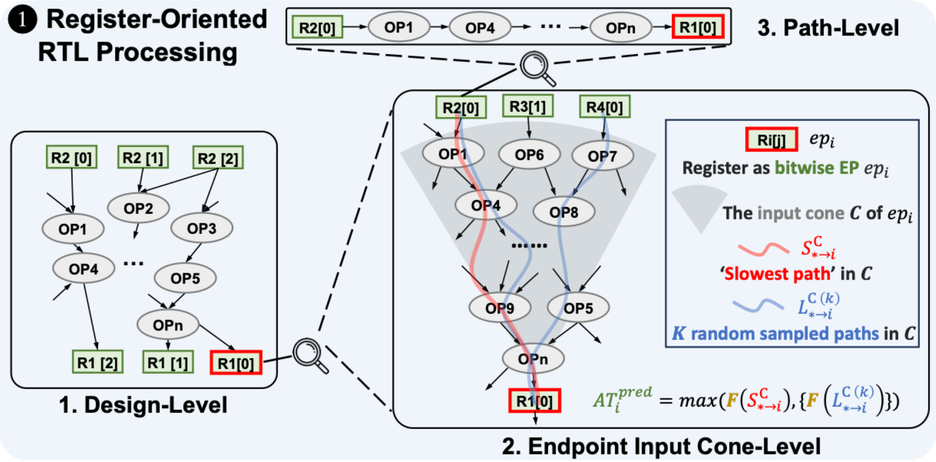
Example: Timing & Power Evaluation of RTL Code?



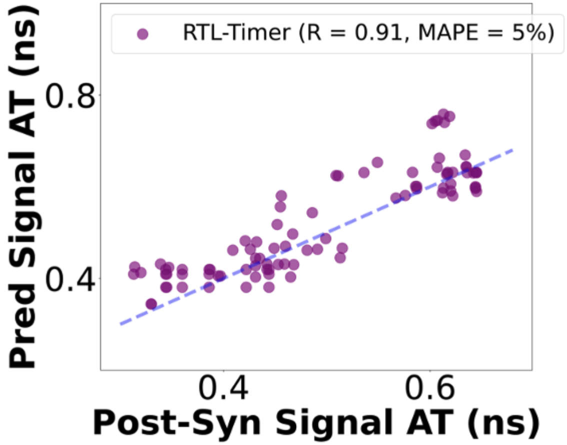
- Given an RTL, can we directly evaluate its **timing** and **power**?
  - Fine-grained **timing**: slack per register
  - Fine-grained **power**: per-cycle power

Case 1: Early Timing Prediction at RTL-Stage

- **Fine-grained timing** model at RTL
  - Evaluate slack of each register endpoint
  - Annotate slack directly on HDL
- **Guide optimization** during synthesis
  - Guide **retime** and **path_group**



High **correlation** in prediction

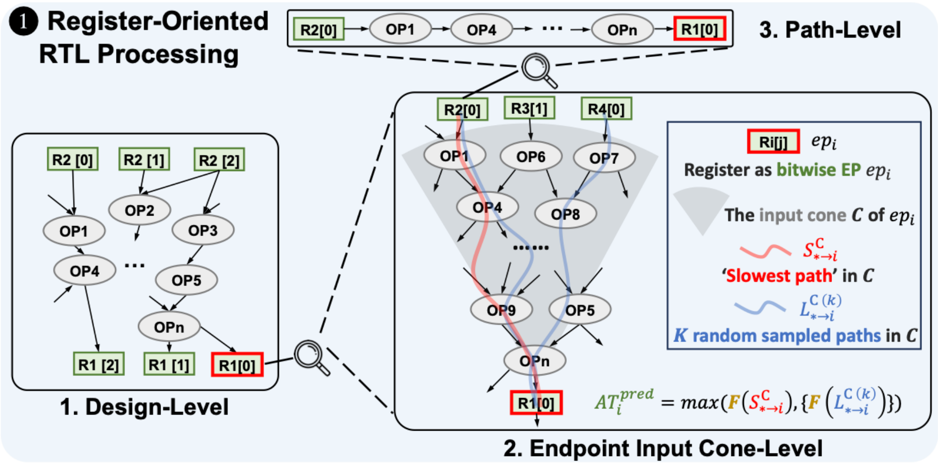


Better **post-opt timing distribution**

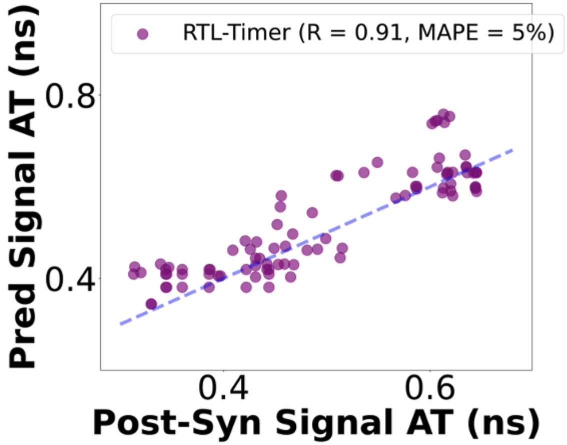
[1] Annotating Slack Directly on Your Verilog: Fine-Grained RTL Timing Evaluation for Early Optimization, **DAC 2024**
[2] Transferable Pre-Synthesis PPA Estimation for RTL Designs with Data Augmentation Techniques, **TCAD 2024**

Case 1: Early Timing Prediction at RTL-Stage

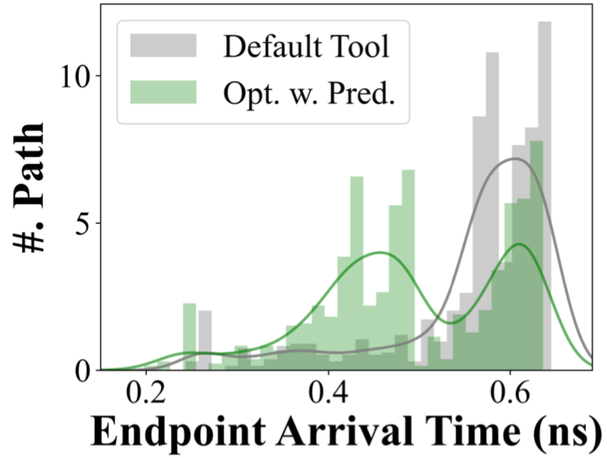
- **Fine-grained timing** model at RTL
  - Evaluate slack of each register endpoint
  - Annotate slack directly on HDL
- **Guide optimization** during synthesis
  - Guide **retime** and **path_group**



High **correlation** in prediction



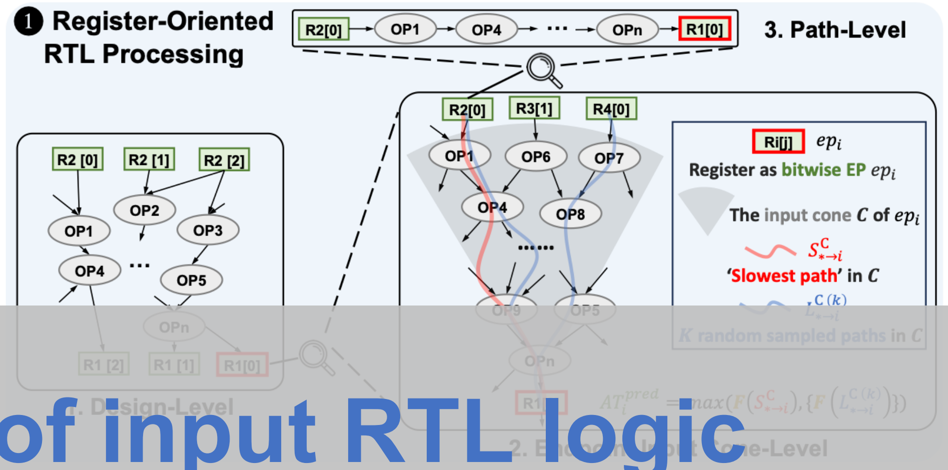
Better **post-opt timing distribution**



[1] Annotating Slack Directly on Your Verilog: Fine-Grained RTL Timing Evaluation for Early Optimization, **DAC 2024**
[2] Transferable Pre-Synthesis PPA Estimation for RTL Designs with Data Augmentation Techniques**, TCAD 2024**

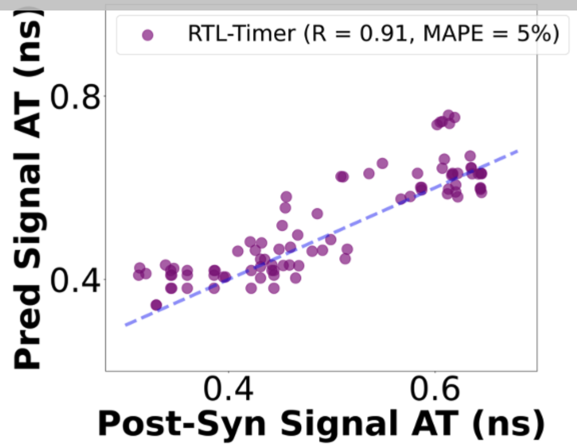
Case 1: Early Timing Prediction at RTL-Stage

- **Fine-grained timing** model at RTL
  - Evaluate slack of each register endpoint
  - Annotate slack directly on HDL
- **Guide optimization** during synthesis
  - Guide retime and path group

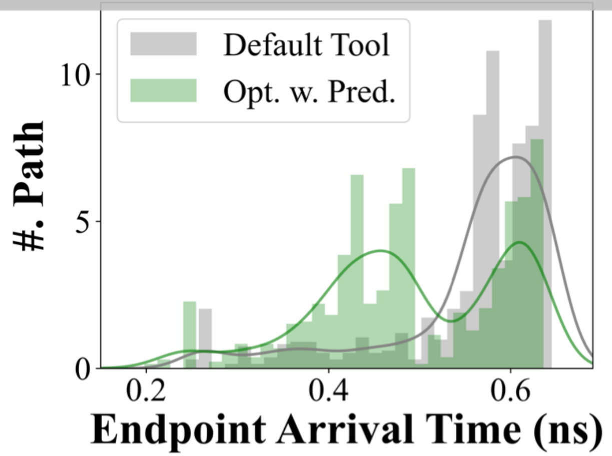


Key idea: learn the pattern of input RTL logic

High **correlation** in prediction



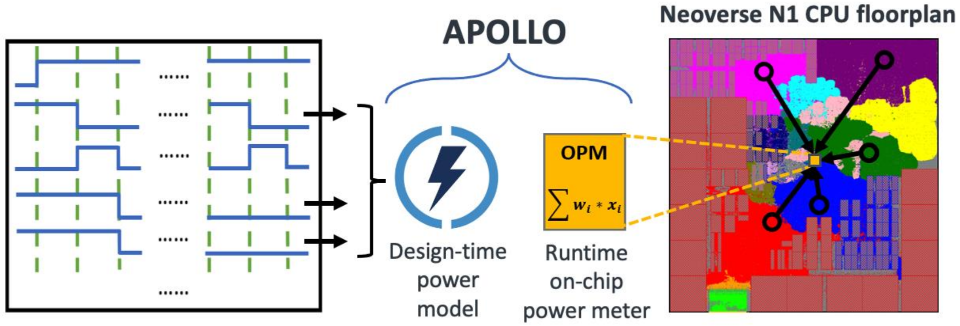
Better **post-opt timing distribution**



[1] Annotating Slack Directly on Your Verilog: Fine-Grained RTL Timing Evaluation for Early Optimization, **DAC 2024**
[2] Transferable Pre-Synthesis PPA Estimation for RTL Designs with Data Augmentation Techniques, **TCAD 2024**

Case 2: Efficient Power Model at RTL-Stage

- Per-cycle power model at RTL
  - Capture **key RTL signals** as inputs (proxies)
  - Fast & accurate **design-time simulation**
  - Low-cost & accurate **on-chip power model**

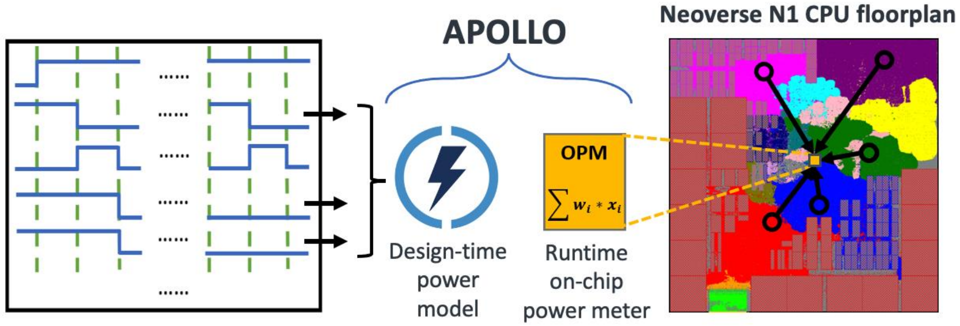


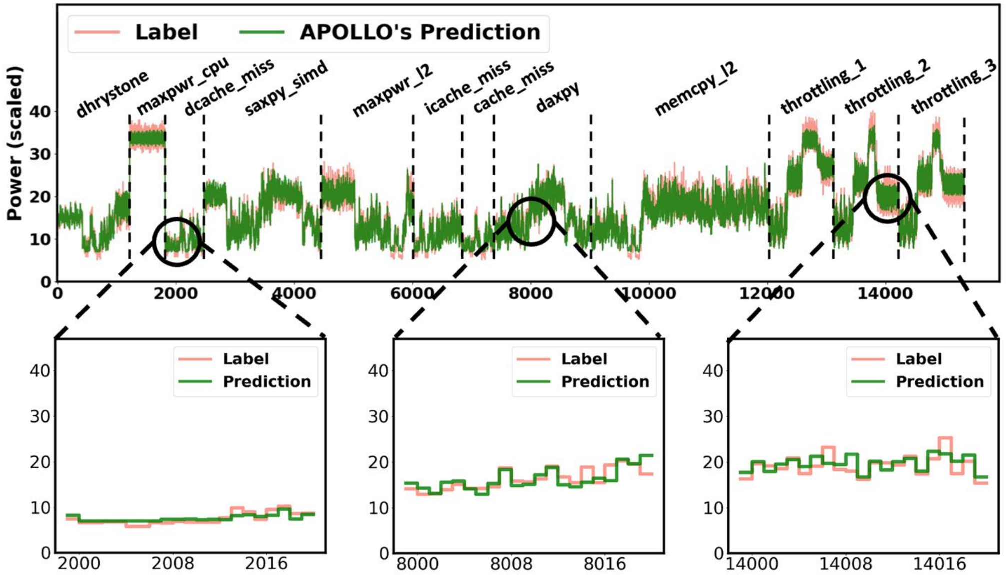
Small OPM in CPU layout (pink)

[1] APOLLO: An Automated Power Modeling Framework for … Microprocessors, **MICRO 2021 (Best Paper Award)**
[2] DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters," **ICCAD 2022**
[3] Unleashing Flexibility of ML-based Power Estimators Through Efficient Development Strategies, **ISLPED 2024 (Best Paper Nomination)**

Case 2: Efficient Power Model at RTL-Stage

- Per-cycle power model at RTL
  - Capture **key RTL signals** as inputs (proxies)
  - Fast & accurate **design-time simulation**
  - Low-cost & accurate **on-chip power model**



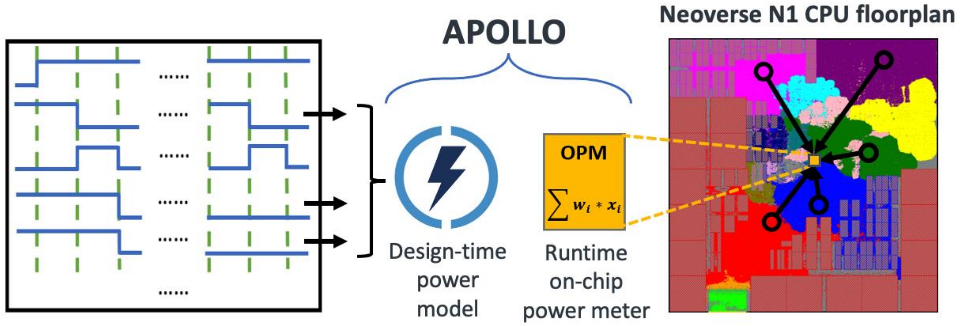


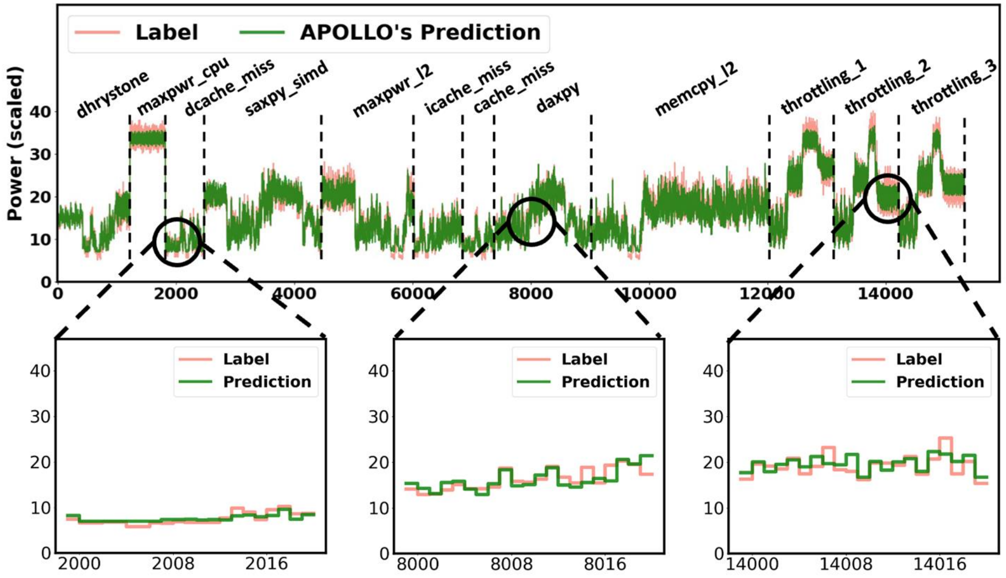
Small OPM in CPU layout (pink)

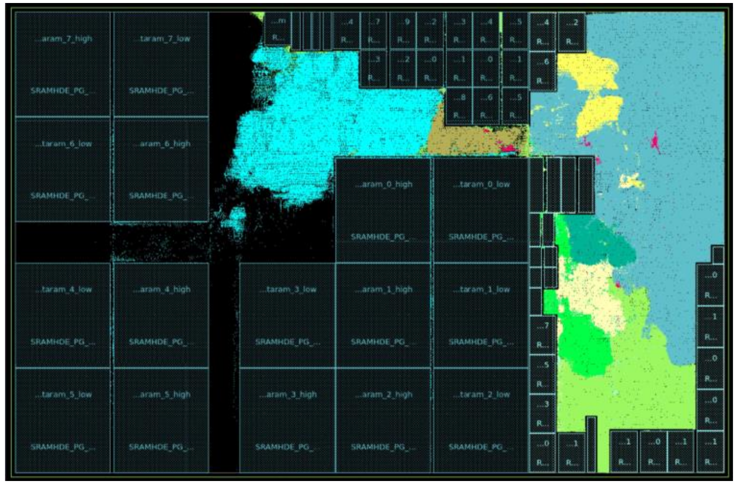
[1] APOLLO: An Automated Power Modeling Framework for … Microprocessors, **MICRO 2021 (Best Paper Award)**
[2] DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters," **ICCAD 2022**
[3] Unleashing Flexibility of ML-based Power Estimators Through Efficient Development Strategies, **ISLPED 2024 (Best Paper Nomination)**

Case 2: Efficient Power Model at RTL-Stage

- Per-cycle power model at RTL
  - Capture **key RTL signals** as inputs (proxies)
  - Fast & accurate **design-time simulation**
  - Low-cost & accurate **on-chip power model**





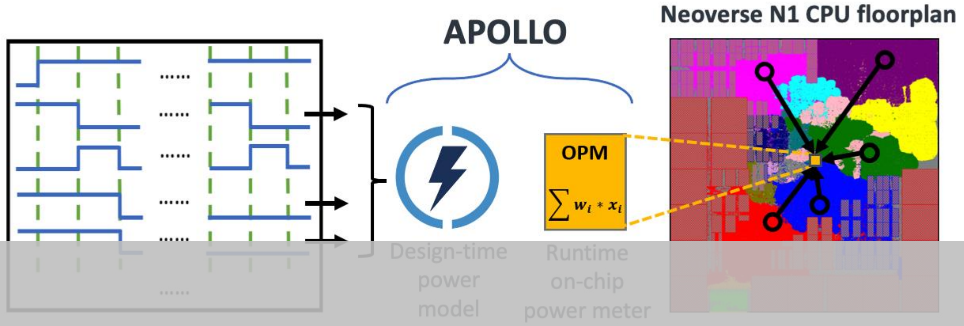


Small OPM in CPU layout (pink)

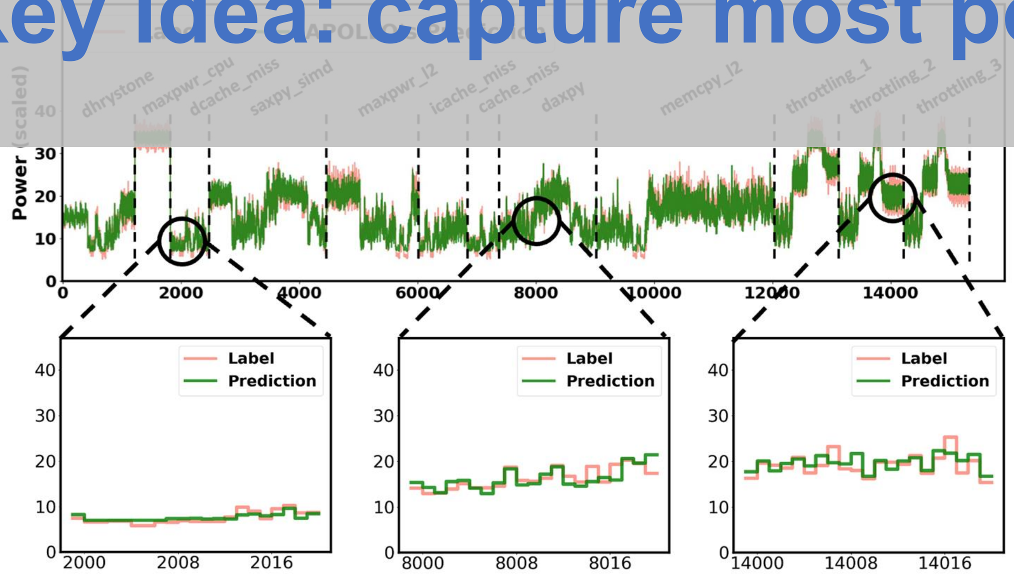
[1] APOLLO: An Automated Power Modeling Framework for … Microprocessors, **MICRO 2021 (Best Paper Award)**
[2] DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters," **ICCAD 2022**
[3] Unleashing Flexibility of ML-based Power Estimators Through Efficient Development Strategies, **ISLPED 2024 (Best Paper Nomination)**

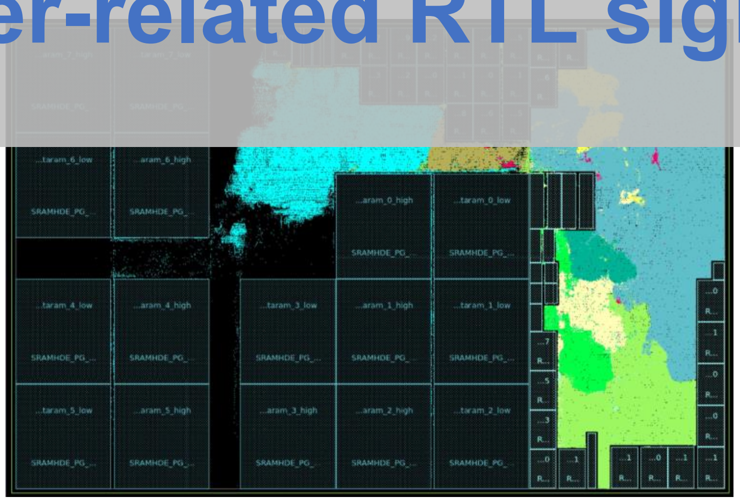
Case 2: Efficient Power Model at RTL-Stage

- Per-cycle power model at RTL
  - Capture **key RTL signals** as inputs (proxies)
  - Fast & accurate **design-time simulation**
  - Low-cost & accurate **on-chip power model**



Key idea: capture most power-related RTL signals





Small OPM in CPU layout (pink)

[1] APOLLO: An Automated Power Modeling Framework for … Microprocessors, **MICRO 2021 (Best Paper Award)**
[2] DEEP: Developing Extremely Efficient Runtime On-Chip Power Meters," **ICCAD 2022**
[3] Unleashing Flexibility of ML-based Power Estimators Through Efficient Development Strategies, **ISLPED 2024 (Best Paper Nomination)**

How AI Assists EDA - Our Taxonomy

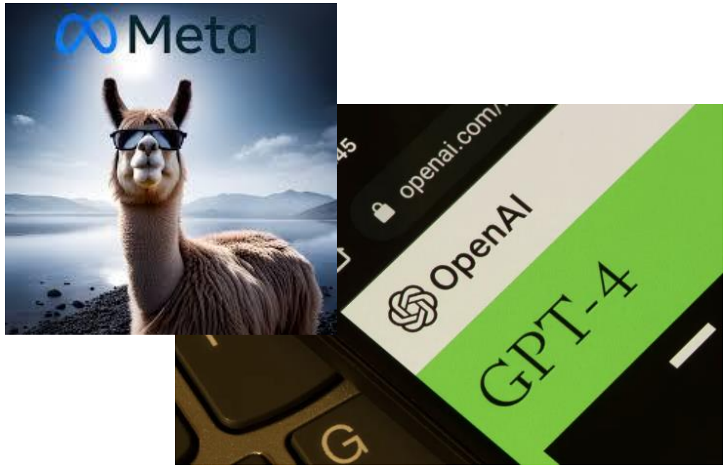
Type I: **Supervised Predictive** AI Techniques for EDA

- **Difficulty** in getting sufficient **labeled data**
- **Time-consuming** AI model **development** process
- **Lack of generalization** across tasks

Type II: **Foundation** AI Techniques for EDA

Opportunities from Foundation Models

- Emergence of **large foundation models** in many fields
  - Unprecedented ability to *understand*, *predict*, and *generate* content



Language model: GPT

Q: Image (A potato king)
A:

Image model: DALL-E

Q: Video (A family of monsters)
A:

Video model: Sora

Why **no counterpart** in **AI for chip design**?

	❶ Task-Specific Models	❷ Pre-Trained Models	❸ Models with Emerging Capability
		Foundation Models	
Language	• Sentiment • Translation	• BERT • GPT-2	• ChatGPT • LLaMA
		Language Foundation Models	
Vision & Multimodal	• Classification • Retrieval • Style Transfer	• MoCo • CLIP • DALLE	• Flamingo • PaLM-E
		Multimodal Foundation Models	
Circuits & EDA	• Prediction • Optimization	? ? ?	

Trend of AI in all fields:

- Task-specific → General
- Small data → Big data
- Supervised → Unsupervised
- Single-modality → Multimodal

Why **no counterpart** in **AI for chip design**?

	❶ Task-Specific Models	❷ Pre-Trained Models	❸ Models with Emerging Capability
			Foundation Models
Language	• Sentiment • Translation	• BERT • GPT-2	• ChatGPT • LLaMA
		Language Foundation Models	
Vision & Multimodal	• Classification • Retrieval • Style Transfer	• MoCo • CLIP • DALLE	• Flamingo • PaLM-E
		Multimodal Foundation Models	
Circuits & EDA	• Prediction • Optimization	? ? ?	
		Circuit Foundation Models (CFM)	

Trend of AI in all fields:

Task-specific → General

Small data → Big data

Supervised → Unsupervised

Single-modality → Multimodal

Circuit Foundation Model (CFM)

How AI Assists EDA- Our Taxonomy

Type I: **Supervised Predictive** AI Techniques for EDA

Type II: **Foundation** AI Techniques for EDA
(Circuit Foundation Model)

Paradigm 1: **Encoder**-based circuit foundation models

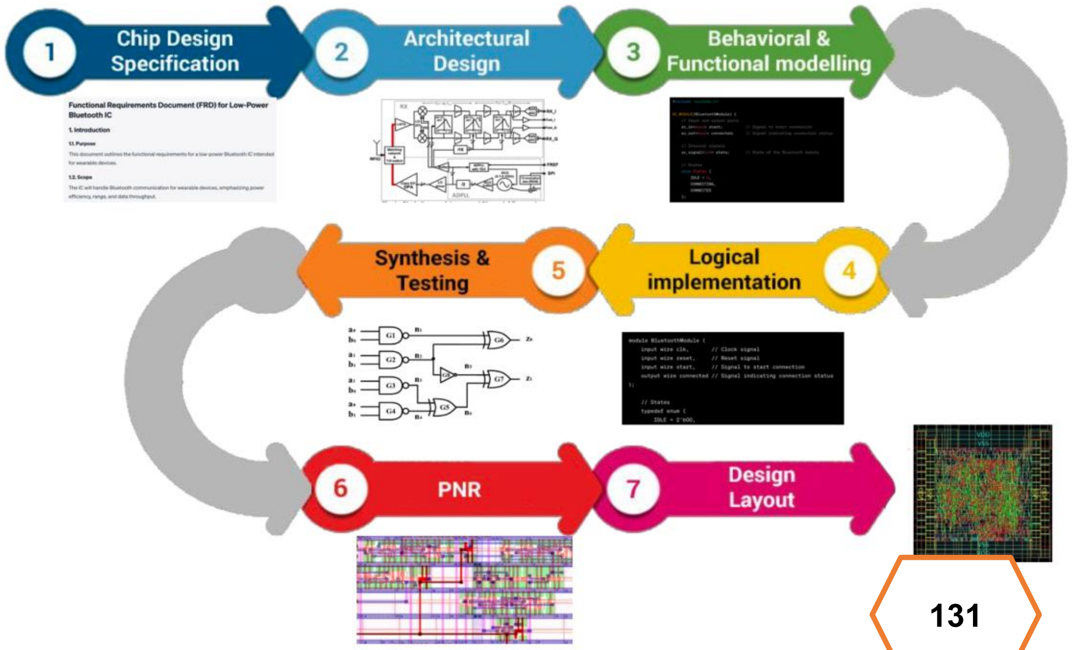
Paradigm 2: **Decoder**-based circuit foundation models

Rethink Circuits from Data Perspective

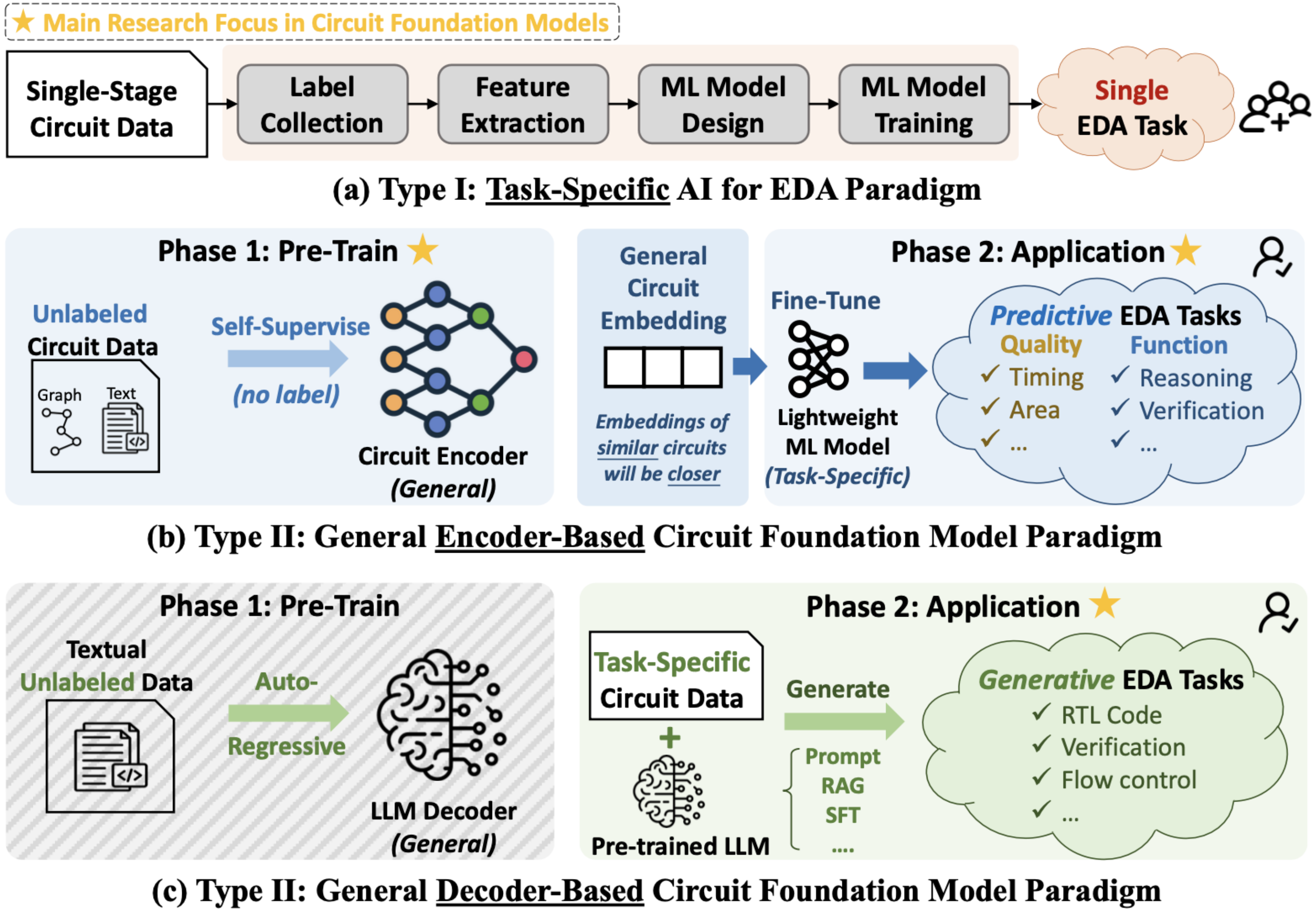
- Chip is a **delicate** *structured* implementation of *functionality*
  - Minor ***structure*** change (flipping a gate) drastically affect ***functionality***

- Chip is inherently **multi-stage** and **multi-modality**:
  - Different level of details across stages

- Lack of **chip data**:
  - The most important IP/asset
  - No companies share their chip design

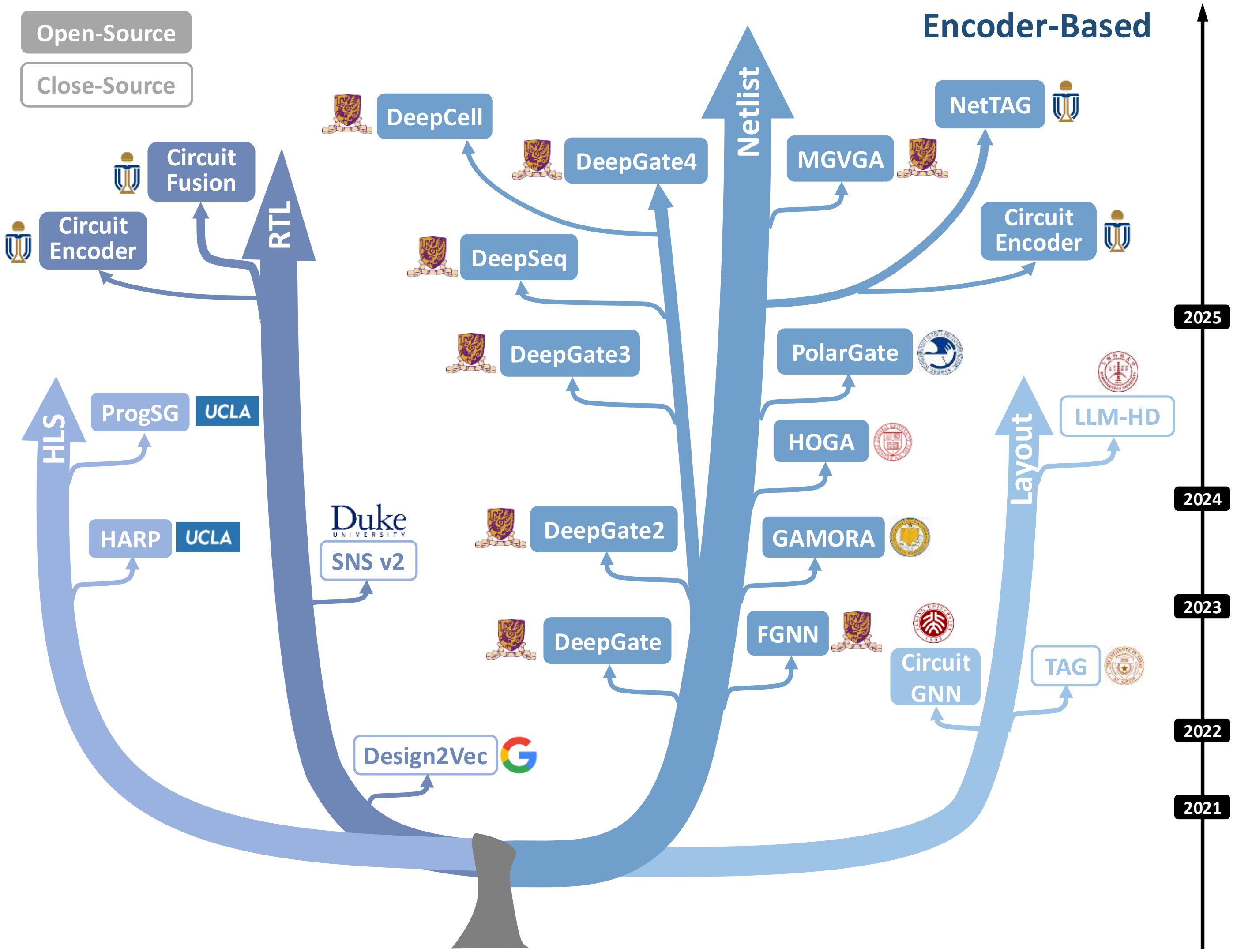


Paradigms of AI for EDA Techniques



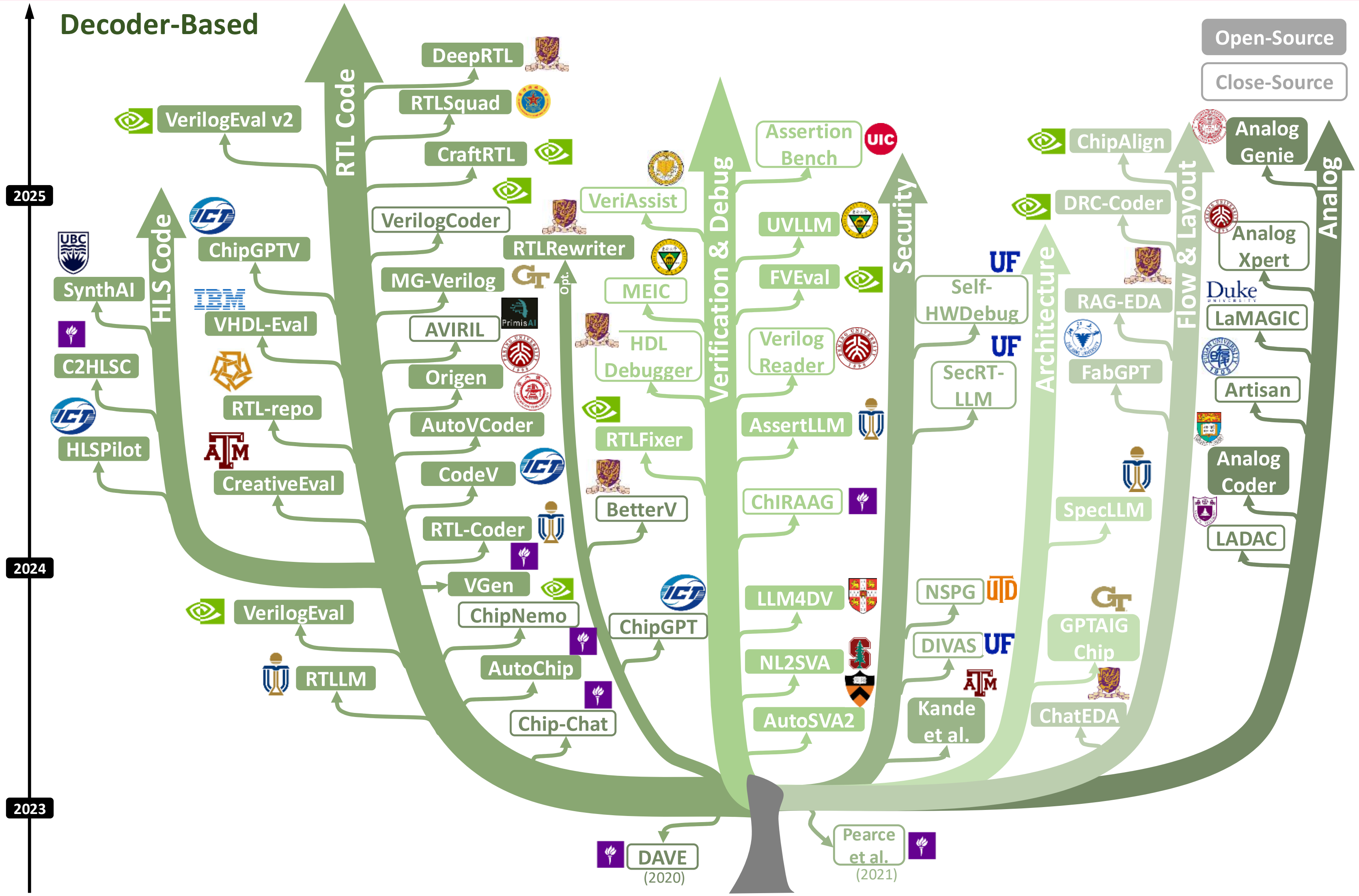
[1] A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA, arXiv:2504.03711.

Encoder-based circuit foundation model



[1] A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA, arXiv:2504.03711.

Decoder-based circuit foundation model



[1] A Survey of Circuit Foundation Model: Foundation AI Models for VLSI Circuit Design and EDA, arXiv:2504.03711.

How AI Assists EDA- Our Taxonomy

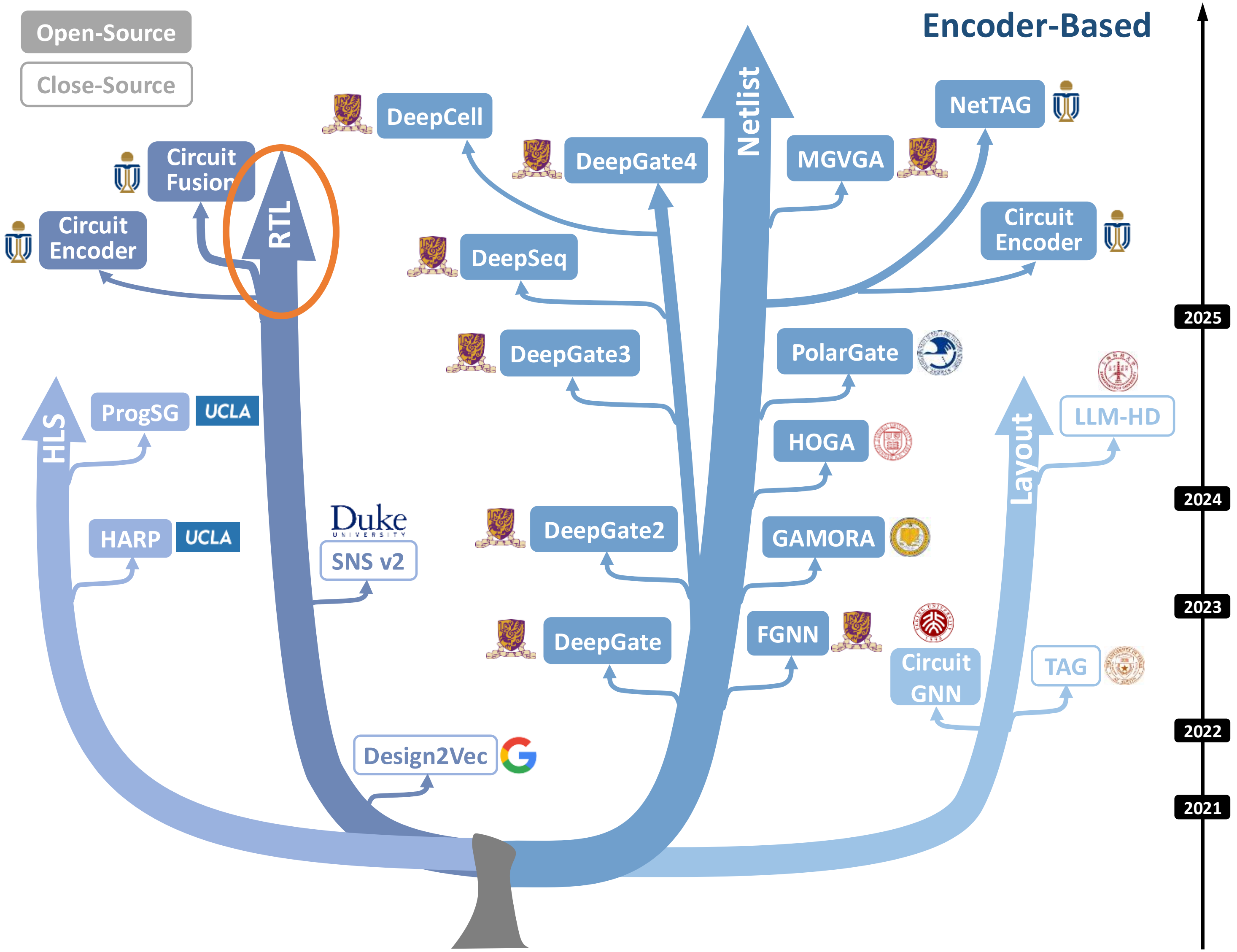
Type I: **Supervised Predictive** AI Techniques for EDA

Type II: **Foundation** AI Techniques for EDA
(Circuit Foundation Model)

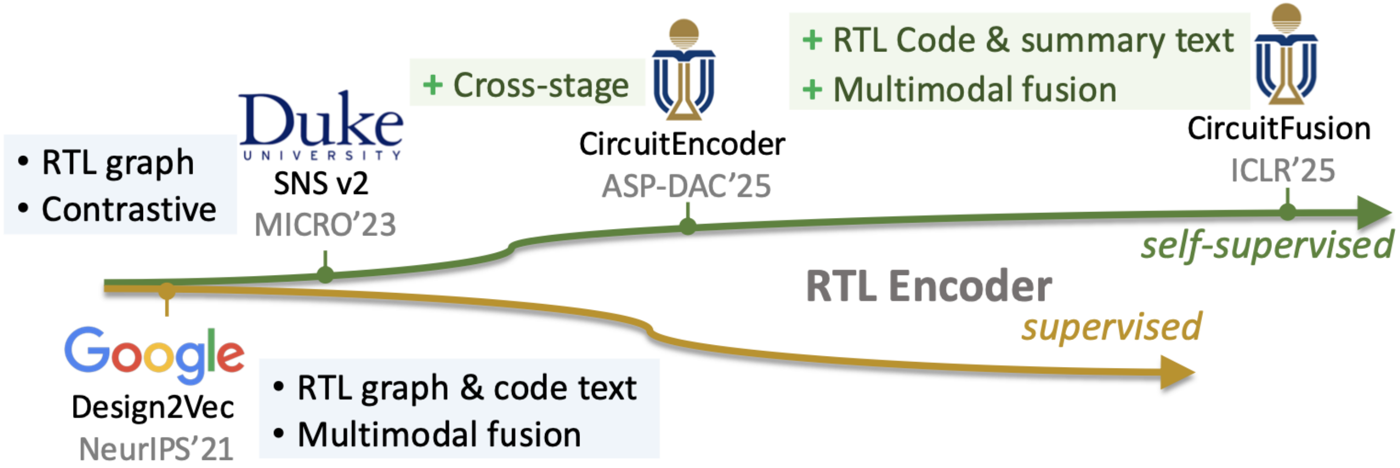
Paradigm 1: **Encoder**-based circuit foundation models

Paradigm 2: **Decoder**-based circuit foundation models

Encoder-based circuit foundation model



Encoder Model at RTL Stage



Target Stage	Method	Technique Pre-train objective	Downstream Task
RTL	SNS v2 [25]	Functional contrastive learning	Post-synthesis PPA prediction
	CircuitEncoder [102]	Intra-stage functional contrastive learning Cross-stage functional contrastive alignment	Post-synthesis PPA prediction
	CircuitFusion [97]	Masked gate reconstruction Functional contrastive for graph/ summary Modality fusion Cross-design-stage alignment	Post-synthesis PPA prediction

Multimodal Representation Learning, on RTL?

- **Encode & fuse information from diverse modalities**
  - Vision-language
  - Graph-language
  - Software-graph
  - ……

- **Can we fuse multiple circuit modalities to learn better circuit representation?**

Summary of Circuit Modalities

- **Multimodal** nature of RTL-stage circuits

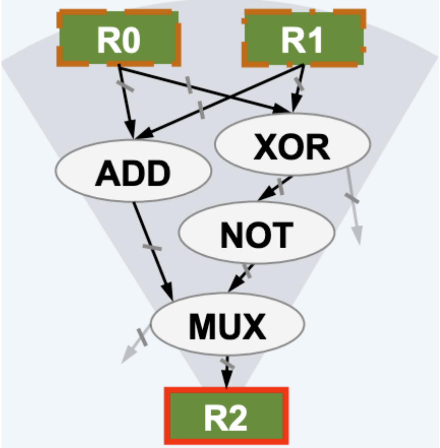
❖ Functionality Summary

❖ Implementation Details

Functionality Summary

```
reg [1:0] R0,R1;
reg [2:0] R2;
wire [2:0] W1,W2;
...
assign W1 = R0 + R1;
...
always @(posedge clk)
  R2 <= W2;
```

HDL Code



Structure Graph

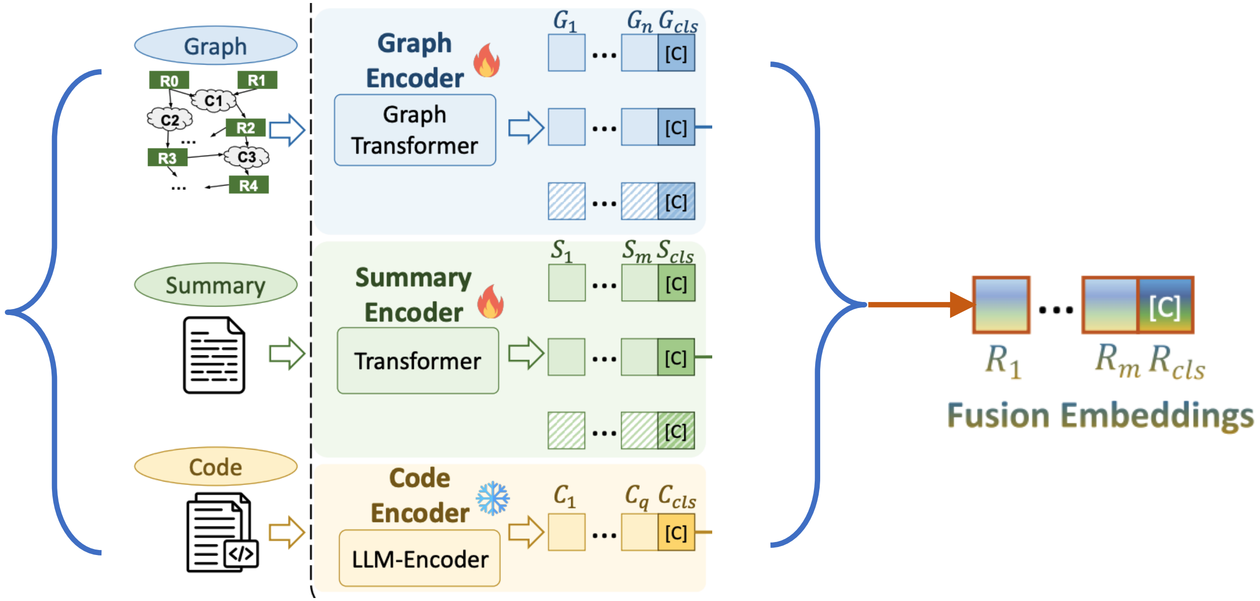
semantic ⟷ *structure*

RTL Circuit Encoder: CircuitFusion

- **Pre-Training**: Multimodal circuit **encoder** (unsupervised) training
  1. Learn to recognize **masked circuit elements**
  2. Learn to recognize **circuits with the same functionality**

Unsupervised contrastive learning

An RTL (sub-)circuit



A multimodal **circuit encoder**

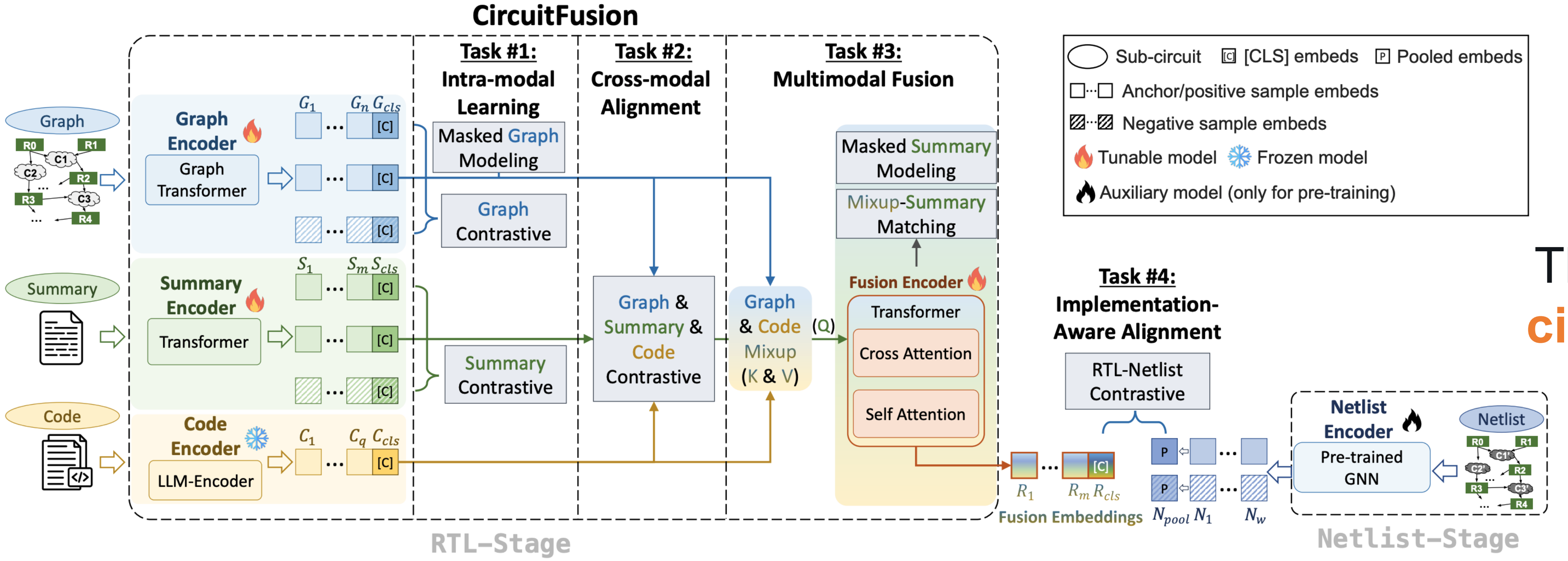
The **encoder** converts **RTL** into a general **embedding**

[1] CircuitFusion: Multimodal Circuit Representation Learning for Agile Chip Design, **ICLR'25.**

RTL Circuit Encoder: CircuitFusion

- **Pre-Training**: Multimodal circuit encoder (unsupervised) training
  1. Learn to recognize **masked circuit elements**
  2. Learn to recognize **circuits with the same functionality**

Unsupervised contrastive learning

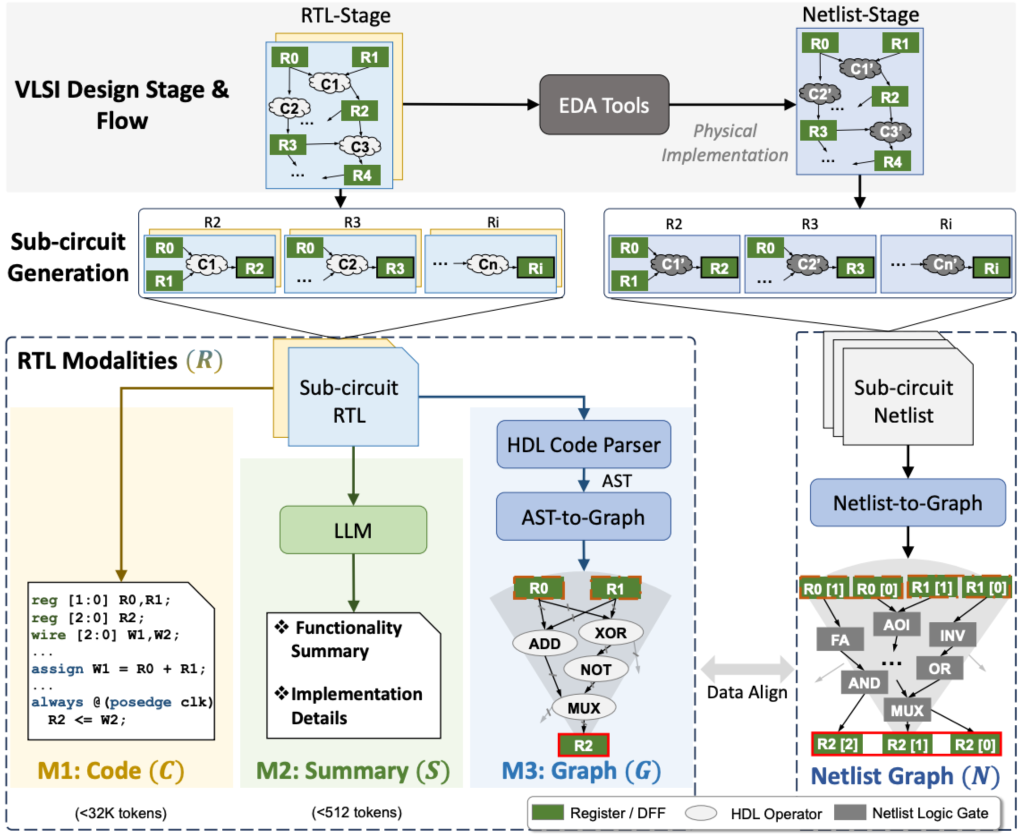


The multimodal circuit encoder

[1] CircuitFusion: Multimodal Circuit Representation Learning for Agile Chip Design, **ICLR’25.**

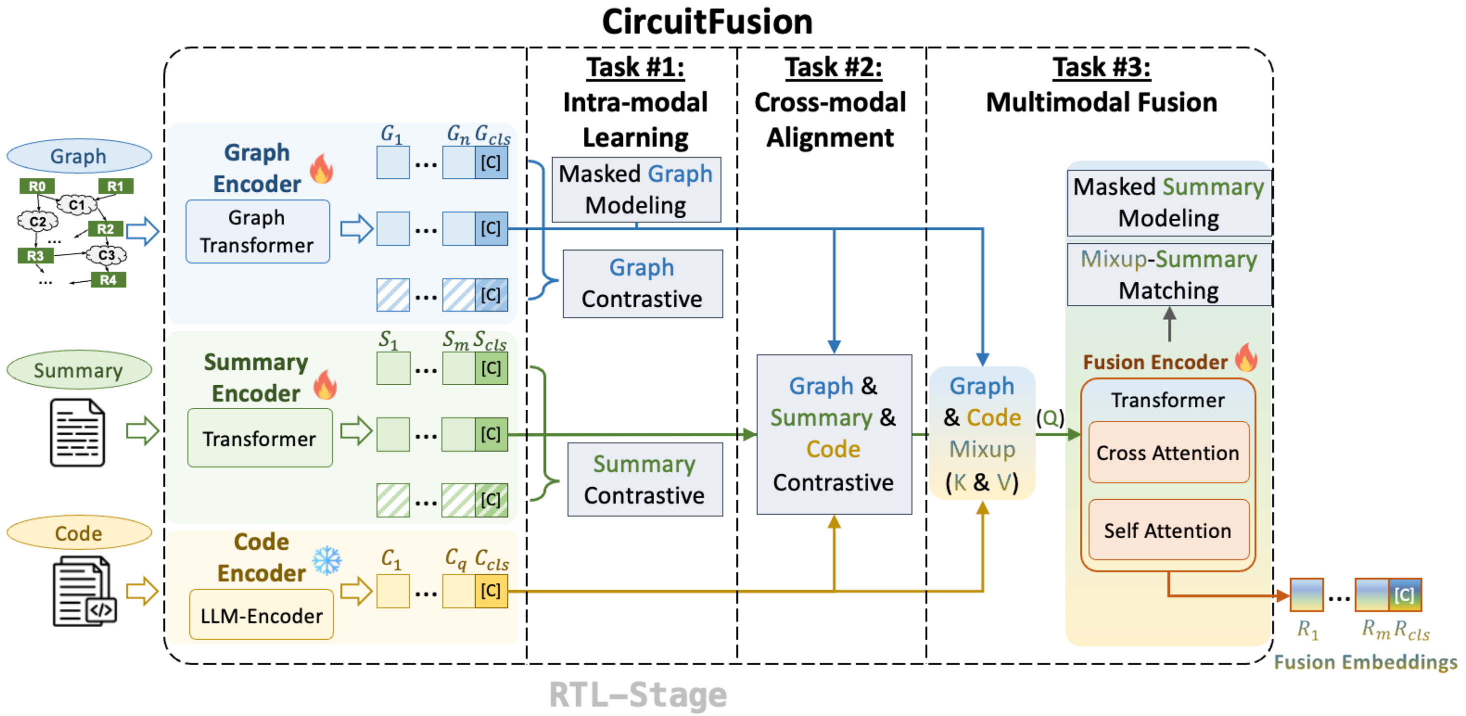
Preprocessing: Split Circuit to Sub-circuits

- **Circuit Property 1: parallel execution**
  - Combinational logic calculates simultaneously
  - Sequential registers are updated only at each clock cycle
- **Strategy 1: sub-circuit generation**
  - Split based on register cones
    - Backtrace all combinational input logic
  - Advantages
    - Consistency in Modality & stage
    - Complete state transition of 1 cycle
    - Intermediate granularity



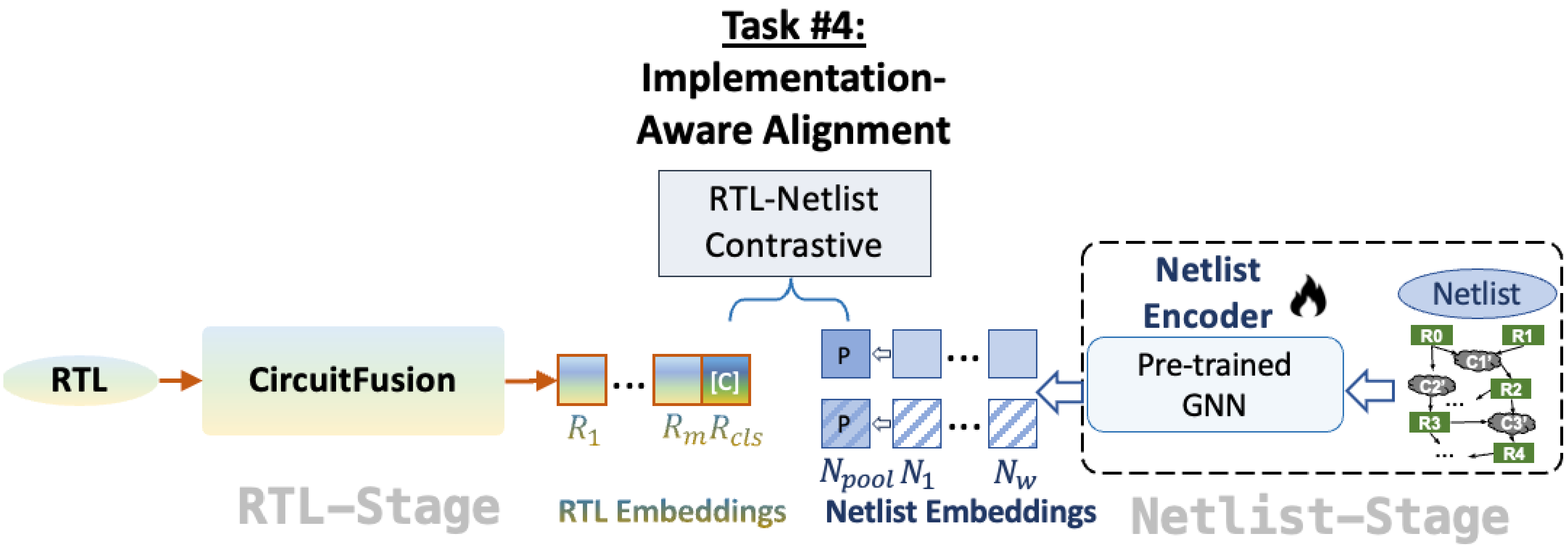
CircuitFusion Pre-Training (1/2)

- **Circuit Property 2: functional equivalent transformation**
  - Circuit w. similar function may have different structures
- **Strategy 2: semantic-structure pre-training**
  - Self-supervised Task #1-3 for each modality and multimodal fusion



CircuitFusion Pre-Training (2/2)

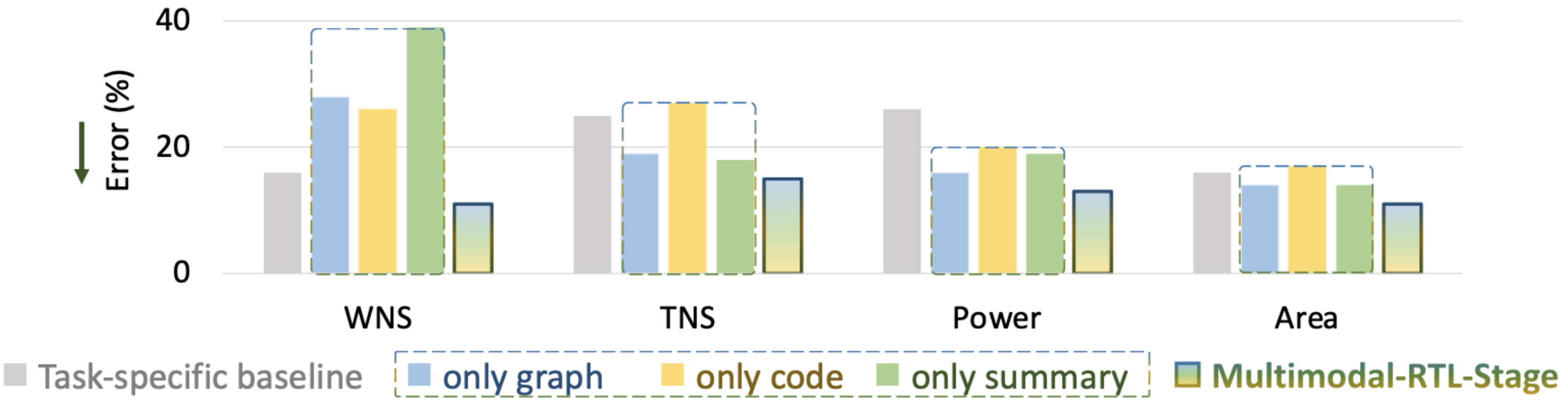
- **Circuit Property 3: multiple design stages**
  - RTL (high-level semantics) → netlist (low-level details)
- **Strategy 3: implementation-aware alignment**
  - Pre-training with netlist encoder across design stage (Task #4)

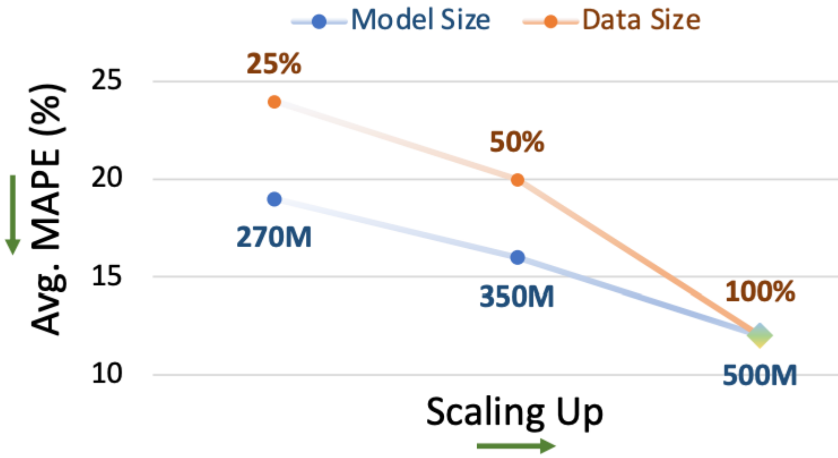


Design Quality Prediction Tasks at RTL

- **High performance on RTL-stage PPA prediction:**

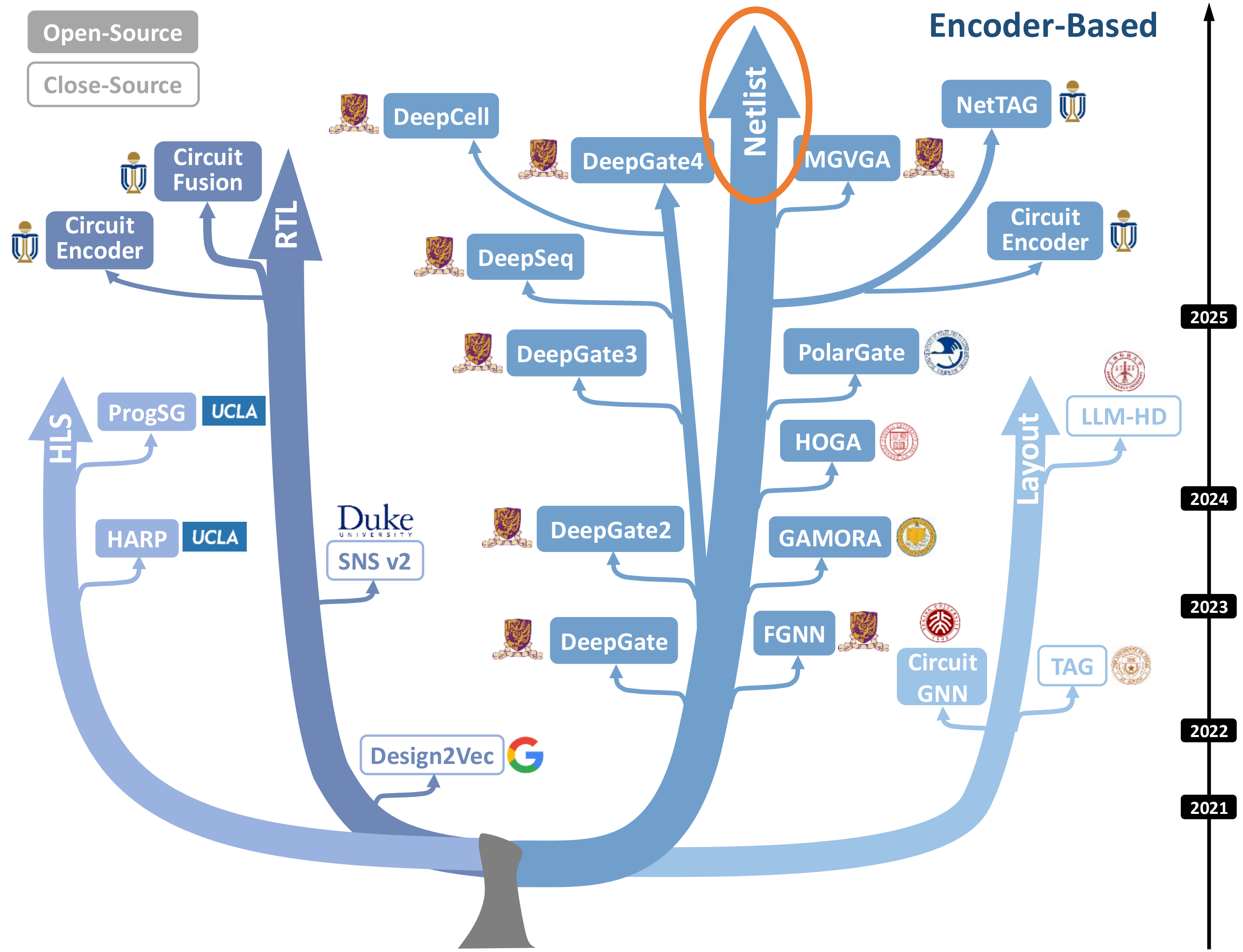
Type	Method	Slack		WNS		TNS		Power		Area	
		R	MAPE	R	MAPE	R	MAPE	R	MAPE	R	MAPE
Hardware Solution	RTL-Timer	0.85	17%	0.9	16%	0.96	25%	N/A		N/A	
Hardware Solution	MasterRTL	N/A		0.89	18%	0.94	28%	0.89	26%	0.98	16%
Hardware Solution	SNS v2	N/A		0.82	22%	N/A		0.76	28%	0.93	25%
Text Encoder	NV-Embed-v1	N/A		0.49	17%	0.97	55%	0.85	44%	0.86	24%
Software Code Encoder	UnixCoder	N/A		0.46	21%	0.95	44%	0.83	29%	0.85	26%
Software Code Encoder	CodeT5+ Encoder	N/A		0.55	21%	0.63	43%	0.49	46%	0.45	39%
Software Code Encoder	CodeSage	N/A		0.23	25%	0.86	45%	0.8	38%	0.77	41%
Ours	**CircuitFusion**	**0.87**	**12%**	**0.91**	**11%**	**0.99**	**15%**	**0.99**	**13%**	**0.99**	**11%**



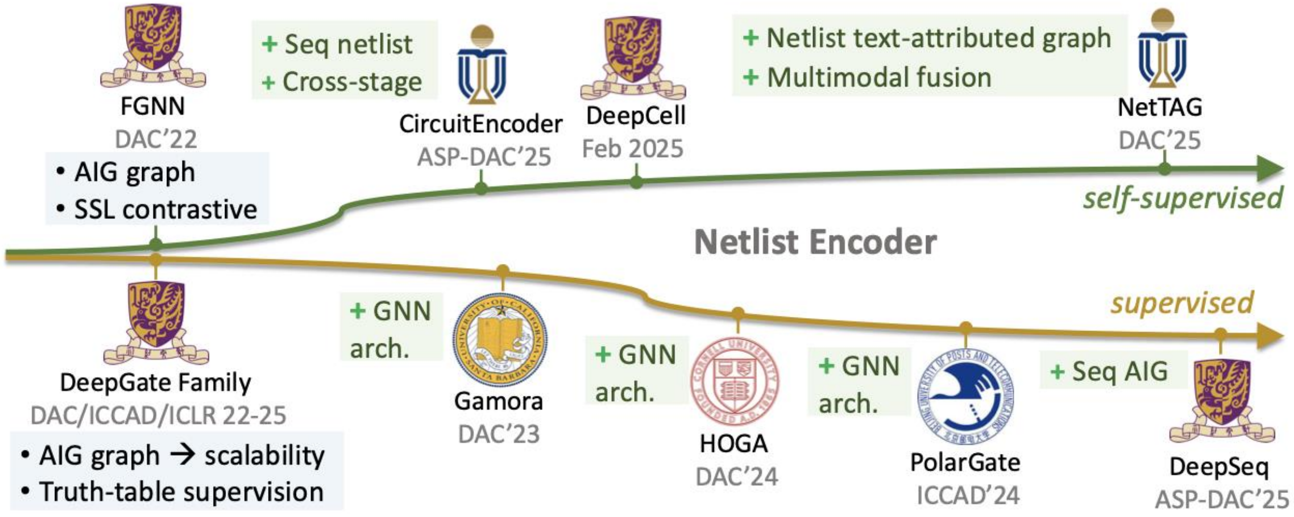


[1] CircuitFusion: Multimodal Circuit Representation Learning for Agile Chip Design, **ICLR'25.**

Encoder-based circuit foundation model



Encoder Model at Netlist Stage



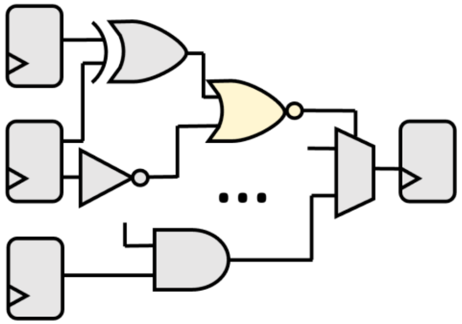
Target Stage	Method	Modality		Pre-Training		Downstream Task	
		Graph	Text	Self-Supervised	Supervised	Design Quality	Functionality
Netlist	DeepGate [103]	✓			✓		✓
	DeepGate2 [104]	✓			✓		✓
	DeepGate3/4 [98, 105]	✓			✓		✓
	GAMORA [54]	✓			✓		✓
	HOGA [106]	✓			✓	✓	✓
	PolarGate [107]	✓			✓		✓
	DeepSeq [108, 109]	✓			✓	✓	
	FGNN [110, 111]	✓		✓			✓
	CircuitEncoder [102]	✓		✓			✓
	MGVGA [112]	✓	✓	✓		✓	✓
	NetTAG [113]	✓	✓	✓		✓	✓
	DeepCell [114]	✓		✓			✓

Multimodal Circuit Learning: RTL vs Netlist

- **Multimodal learning: fuse information from diverse modalities**
  - Vision-language
  - Software-graph
  - ……

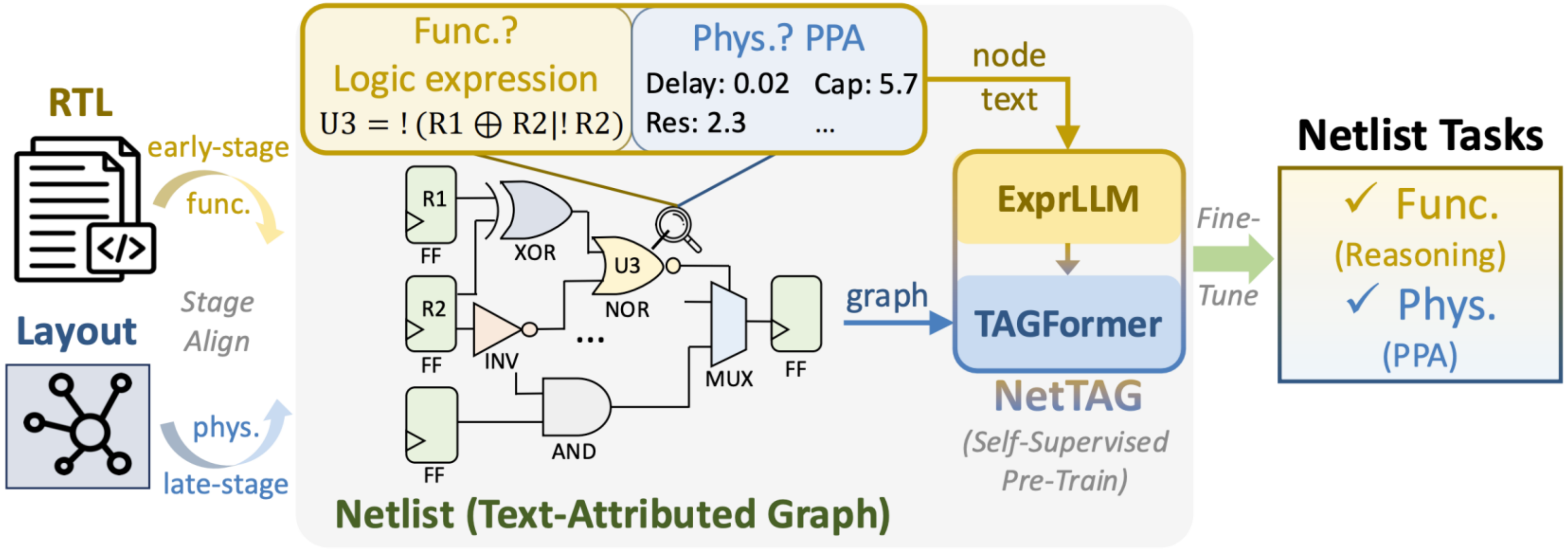
- **Multimodal learning on RTL**
  - ***Register-transfer level*** (RTL)
  - *Earlier* stage → more ***semantic***
  - Fuse 3 RTL modalities at ***register level***

- **Multimodal learning on netlist?**
  - ***Gate-level*** netlists
  - *Later* stage → more ***structure***
  - Should fuse at ***gate level***



NetTAG: A Multimodal Netlist Encoder

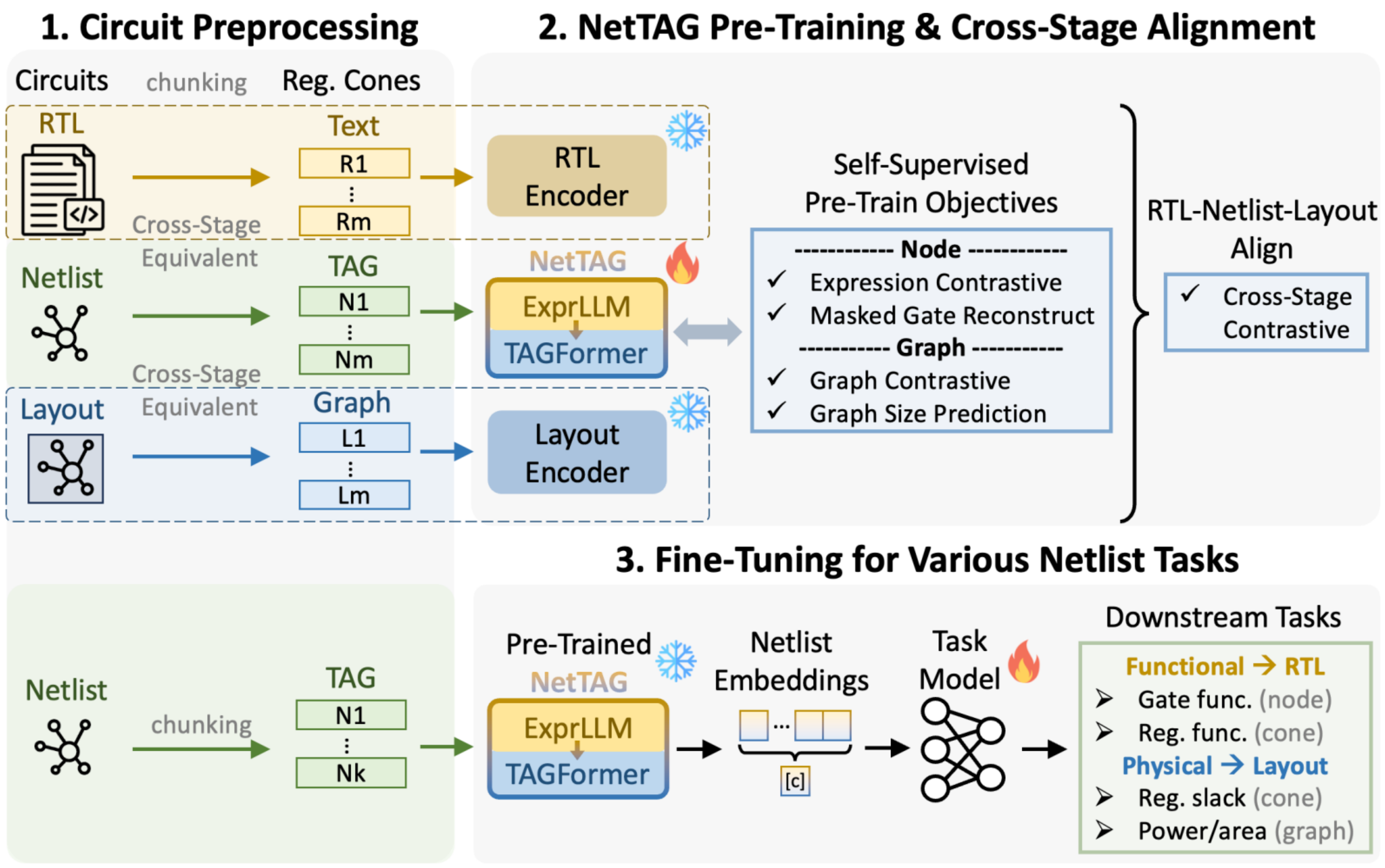
- **Netlist *functional* and *physical* properties in text-attributed graph**
  - **Multimodal preprocess:** formulate netlist as **text-attributed graph**
  - **Multimodal model:** fuse ***gate text*** (LLM) with ***circuit graph*** (GNN)
  - **Multimodal pre-train:** **self-supervised** and **cross-stage-aware**



[1] NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph, **DAC'25.**

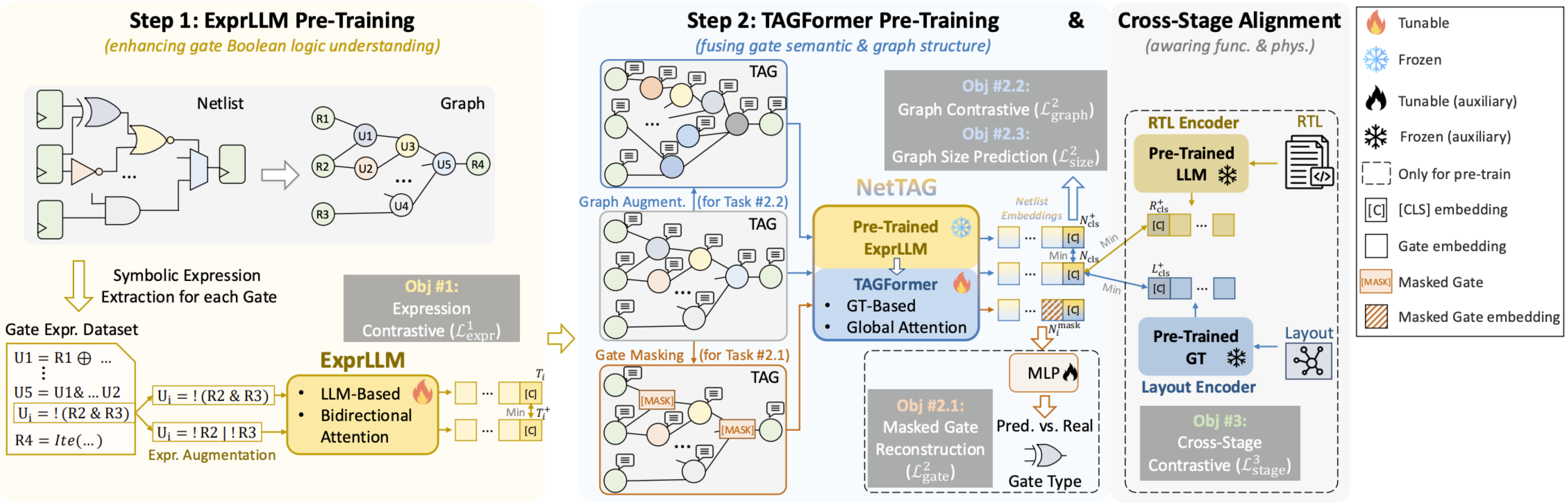
NetTAG Framework Overview

1. Preprocessing → 2. Pre-Training → 3. Fine-Tuning



2. Self-Supervised Pre-Training

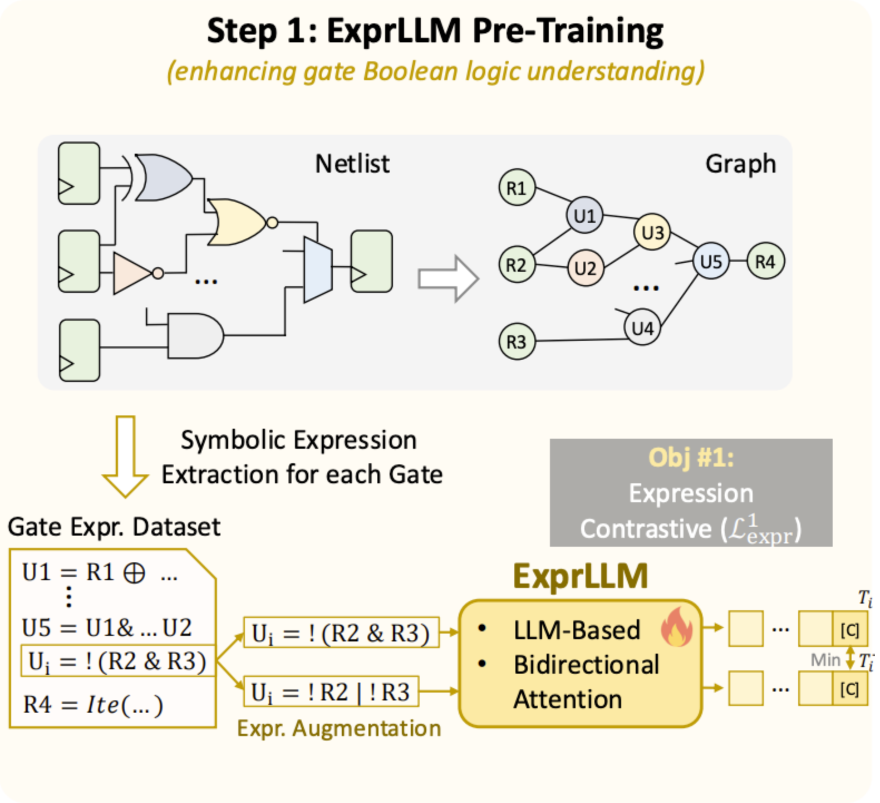
- **Two-phase encoding → Two-step pre-training**
  - Capture netlist ***functional*** and ***physical*** properties



[1] NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph, **DAC'25.**

2. Self-Supervised Pre-Training (1/2)

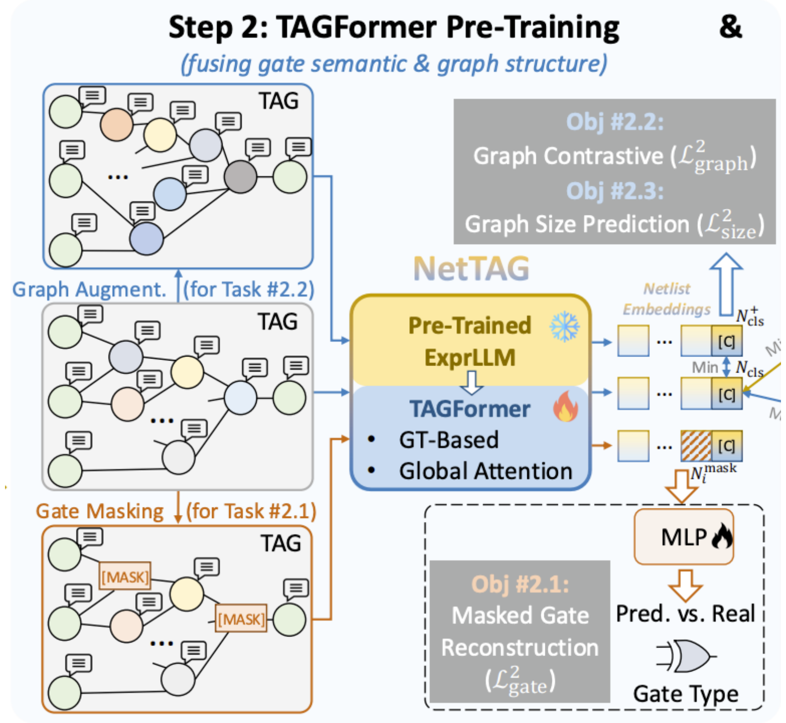
- **Step 1: Enhancing logic understanding in ExprLLM**
  - **Goal 1: Differentiate gate expression functionality**
  - **Objective # 1:** Symbolic expression contrastive learning



- Build **gate expression dataset**
  - 2-hop symbolic expressions
  - Boolean equivalence transformation rules

2. Self-Supervised Pre-Training (2/2)

- **Step 2: Fusion in TAGFormer & Cross-Stage Align**
  - **Goal 2: Training within TAGFormer for semantic and structure fusion**
  - **Objective # 2.1:** Masked gate reconstruction
    - *Gate-level*
    - Predict masked gate type to capture **gate structure**
  - **Objective # 2.2:** Netlist graph contrastive learning
    - *Circuit-level*
    - Differentiate **graph functionality**
  - **Objective # 2.3:** Netlist graph size prediction
    - *Circuit-level*
    - Predict gate count to capture **graph structure**

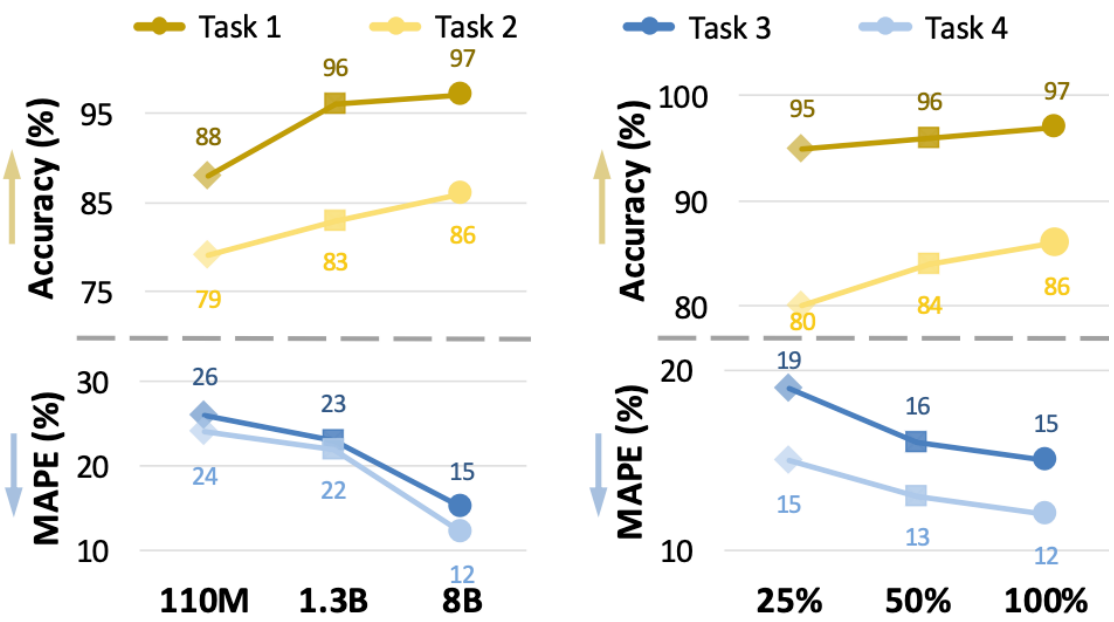


Applications of NetTAG: 4 tasks

- **Task 1: Combinational gate function identification**
  - Identify **functional type** (e.g., adder, multiplier) of each gate
- **Task 2: Sequential state/data register identification**
  - Differentiate **state registers** and **data path registers** for each register
- **Task 3: Endpoint register slack prediction**
  - Predict **layout timing slack** of each register
- **Task 4: Overall circuit power/area prediction**
  - Predict **layout power and area** of the whole design

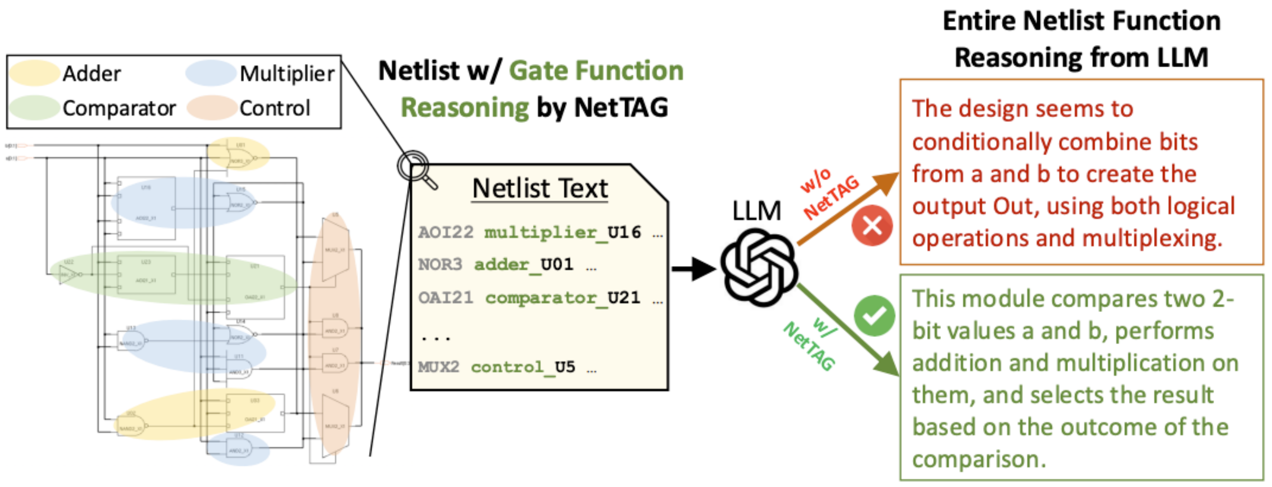
NetTAG Results & Discussion

- **Scalability**
  - Performance per task all scale up with **model** and **data**



(a) Increasing model size (b) Increasing data size

- **Demo**
  - Reasoning the netlist arithmetic function
  - **Next step:** **NetTAG-LLM alignment**[1] for generative reasoning



[1] NetTAG: A Multimodal RTL-and-Layout-Aligned Netlist Foundation Model via Text-Attributed Graph, **DAC'25.**

How AI Assists EDA- Our Taxonomy

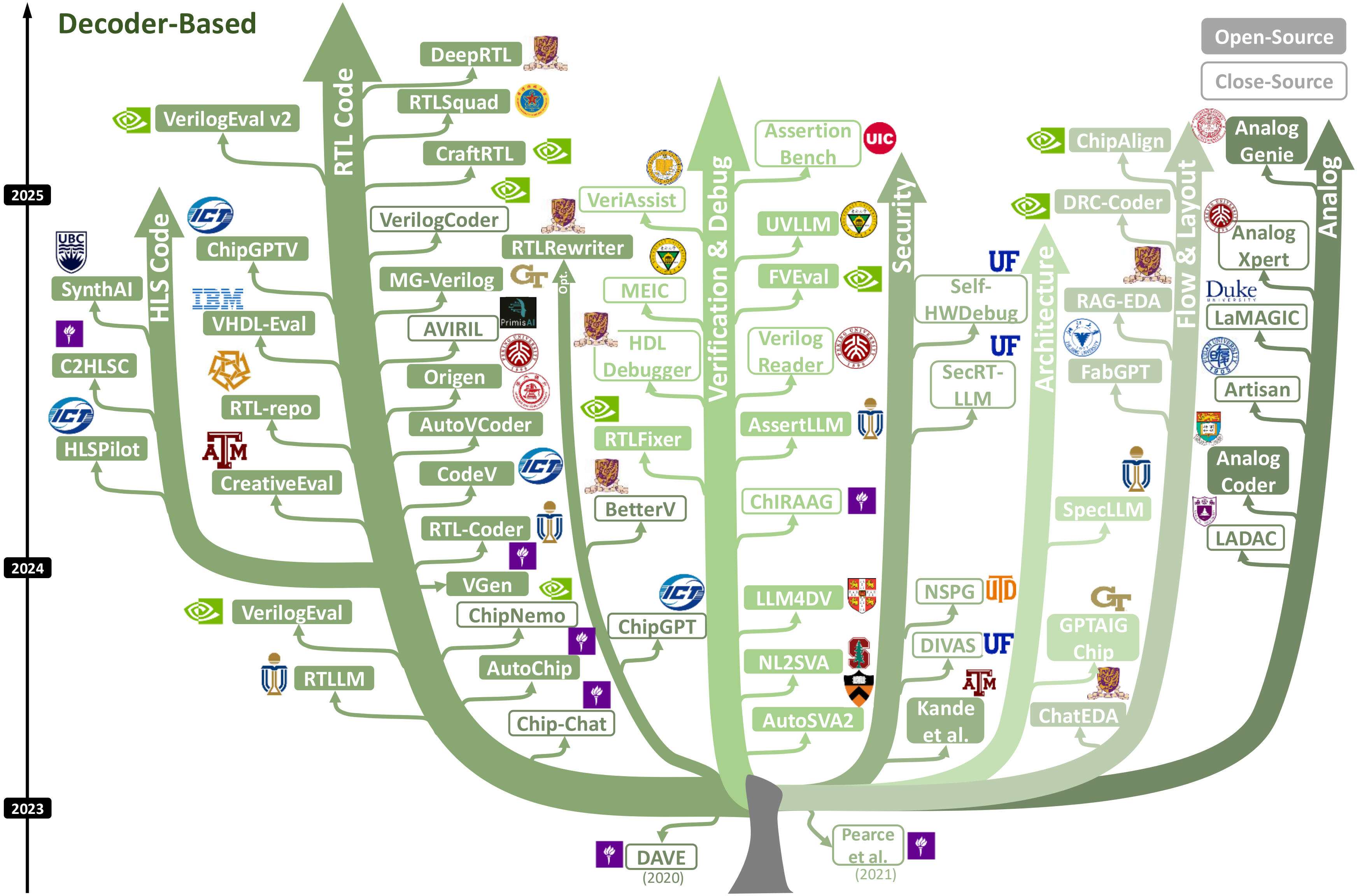
Type I: **Supervised Predictive** AI Techniques for EDA

Type II: **Foundation** AI Techniques for EDA
(Circuit Foundation Model)

Paradigm 1: **Encoder**-based circuit foundation models

Paradigm 2: **Decoder**-based circuit foundation models

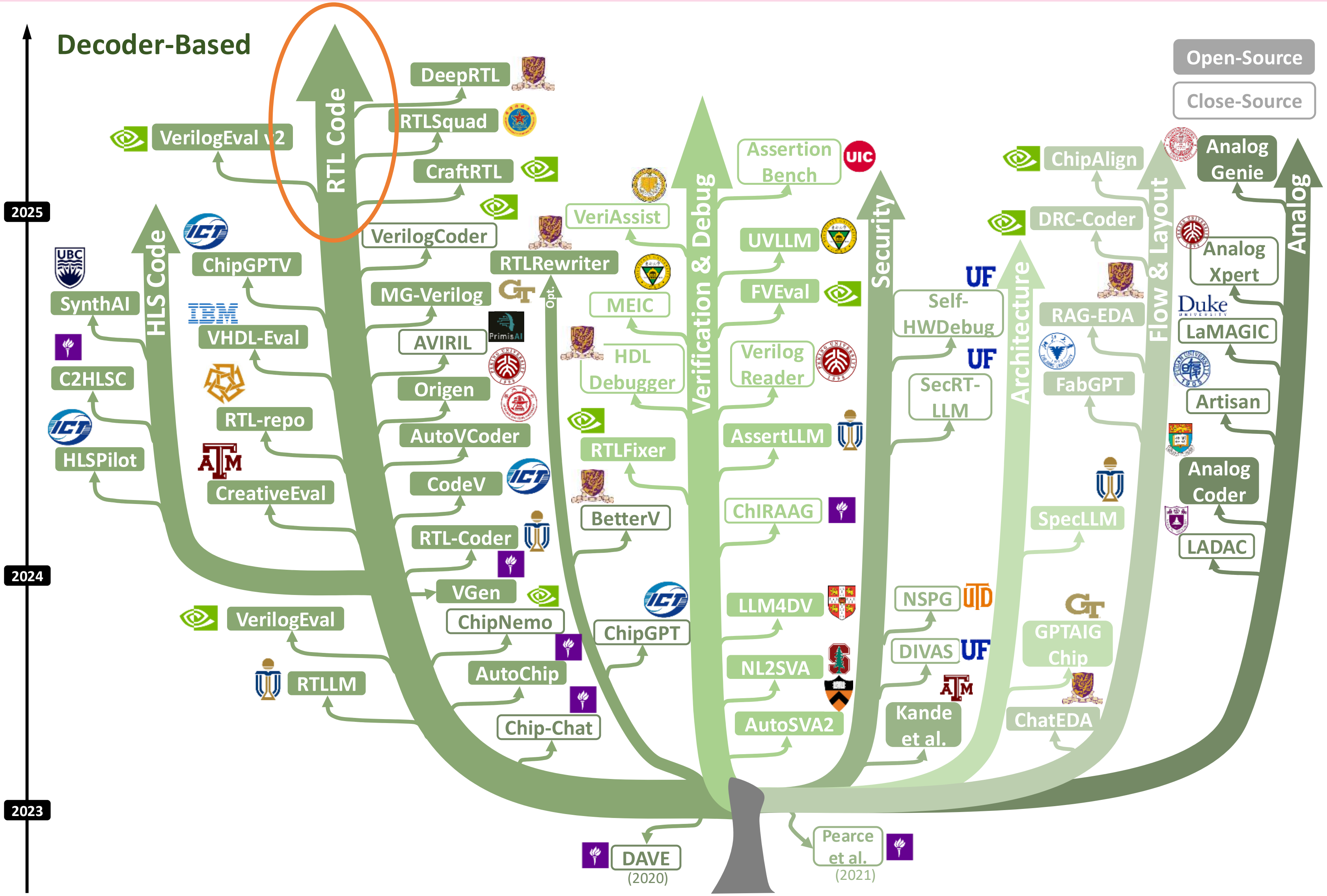
Decoder-based circuit foundation model



LLMs Enable Many Generative Applications

- **RTL code generation (Section 5.1)** — DAVE [132], ChipGPT [133], VerilogEval [14], GPT4AIGChip [134], Chip-Chat [135], AutoChip [136], RTLLM [12], VeriGen [137], RapidGPT [138], CodeV [139], AutoVCoder [140], BetterV [141], ChipNemo [13], Chang et al. [142], OriGen [143], VerilogCoder [144], Thakur et al. [15], RTLCoder [145], MG-Verilog [146], CreativeEval [147], VHDL-Eval [148], Chang et al. [149], OPL4GPT [150], RTLSquad [151], MAGE [152], RTL-repo [153], OpenLLM-RTL [154], Sun et al. [155], DeepRTL [156], CraftRTL [157], [158–166]
- **HLS code generation (Section 5.2)** — HLSPilot [167], C2HLSC [168], SynthAI [169], Liao et al. [170], Gai et al. [171]
- **Design optimization (Section 5.3)** — BetterV [141], ChipGPT [133], RTLRewriter [172], Martine et al. [173], Xu et al. [174], Thorat et al. [175], Sandal et al. [161], DeLorenzo et al. [162]
- **Hardware verification (Section 5.4)** — ChipNeMo [13], RTLFixer [176], AutoSVA [177], NSPG [178], DIVAS [179], SimEval [180], AssertLLM [181], ChIRAAG [182], UVLLM [183], LLM4DV [184], VerilogReader [185], FVEval [186], OpenLLM-RTL [154], AssertionBench [187], [188–198]
- **Circuit code debugging (Section 5.5)** — MEIC [199], RTLFixer[176], VeriAssist [191], HDLdebugger [200], Chrysalis [201], Llm4sechw [202], [179, 203, 204]
- **Hardware security (Section 5.6)** — DIVAS [179], Saha et al. [205], Self-HWDebug [206], Ahmad [207], AutoSVA2 [208], ChIAAG [182], Latibari et al. [209], Netlist Whisperer [210], SCAR [211], Kande et al. [188], Pearce et al. [212], [70–73, 178, 193, 213–215]
- **Design flow & Layout (Section 5.7)** — ChatEDA [216], SmartonAI [217], LLSM [218], MetRex [219], DRC-Coder [220], ChipAlign [221], FabGPT [222], Chen et al. [223], Ho et al. [224]
- **Architecture design (Section 5.8)** — AIGChip [134], ChatEDA [216], SpecLLM [225], [217, 226–228]
- **Analog design (Section 5.9)** — LADAC [229], AnalogCoder [230], FLAG [231], ADO-LLM [232], LaMAGIC [233], Artisan [234], LEDRO [235], AnalogXpert [236], AnalogGenie [237]

Decoder-based circuit foundation model

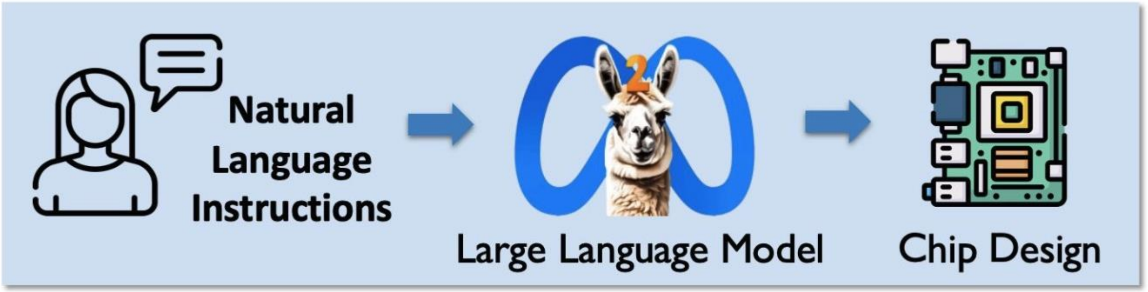


Generative AI: LLM for RTL Generation

Task: LLM-based RTL Generation

- Input: **natural language description**
  - Target design functionality.
  - Module names, I/O names.
- Output: **design in RTL code**

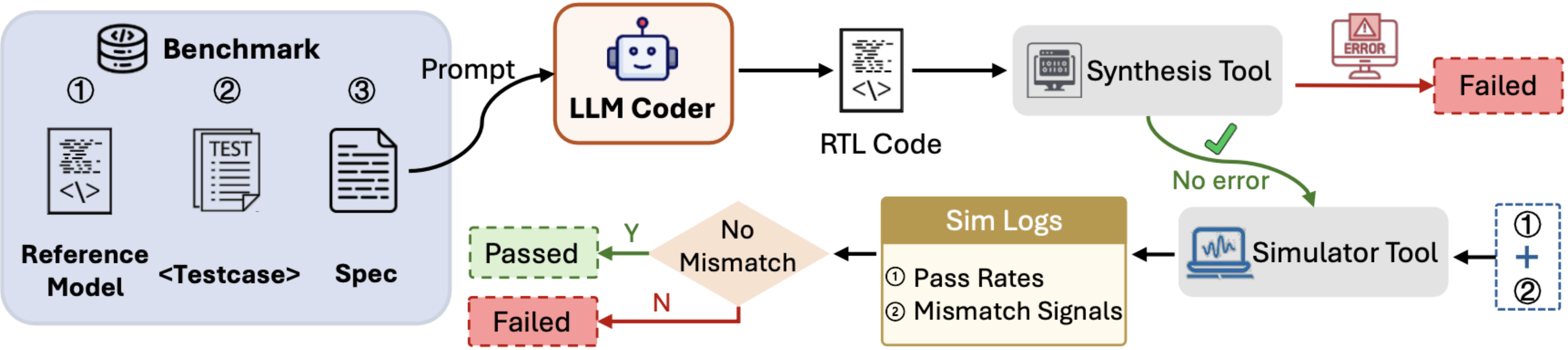
Image from Yongan Zhang, et al., MG-Verilog [LAD'24]



In addition to hardware code generation:

- LLM for hardware code optimization, debugging, verification, …

Benchmarking LLM for RTL Generation



Benchmarks for RTL Code Generation			
Benchmarks	Open-sourced	link	Date
RTLLM [12, 154]	✓	https://github.com/hkust-zhiyao/rtllm	2023-10
VerilogEval [14]	✓	https://github.com/NVlabs/verilog-eval	2023-12
VerilogEval v2[165]			2024-08
CreativeEval [147]	✓	https://github.com/matthewdelorenzo/creativeval	2024-04
RTL-repo [153]	✓	https://github.com/AUCOHL/RTL-Repo	2024-05
VHDL-Eval [148]			2024-06
ChatGPTV [149]	✓	https://github.com/aichipdesign/chipgptv	2024-11

Example: RTLLM2.0

50 design problems

Four categories

1. Arithmetic Modules
2. Memory Modules
3. Control Modules
4. Miscellaneous Modules

Arithmetic Modules

Design	Description
adder_8bit	An 8-bit adder
adder_16bit	A 16-bit adder implemented with full adders
adder_32bit	A 32-bit carry-lookahead adder
adder_pipe_64bit	A 64-bit ripple carry adder based on 4-stage pipeline
adder_bcd	A BCD adder for decimal arithmetic operations
sub_64bit	A 64-bit subtractor for high-precision arithmetic
multi_8bit	An 8-bit multiplier based on shifting and adding operation
multi_16bit	A 16-bit multiplier based on shifting and adding operation
multi_booth_8bit	An 8-bit booth-4 multiplier
multi_pipie_4bit	A 4-bit unsigned number pipeline multiplier
multi_pipie_8bit	An 8-bit unsigned number pipeline multiplier
div_16bit	A 16-bit divider based on subtraction operation
radix2_div	An 8-bit radix-2 divider
comparator_3bit	A 3-bit comparator for comparing binary numbers
comparator_4bit	A 4-bit comparator for comparing binary numbers
accu	Accumulates 8-bit data and output after 4 inputs
fixed_point_adder	A fixed-point adder for arithmetic operations with fixed precision
fixed_point_substractor	A fixed-point subtractor for precise fixed-point arithmetic
float_multi	A floating-point multiplier for high-precision calculations

Control Modules

Design	Description
fsm	FSM detection circuit for specific input
sequence_detector	Detect specific sequences in binary input
counter_12	Counter module counts from 0 to 12
JC_counter	A 4-bit Johnson counter with specific cyclic state sequence
ring_counter	An 8-bit ring counter for cyclic state sequences
up_down_counter	A 16-bit counter that can increment or decrement based on control signals

Memory Modules

Design	Description
asyn_fifo	An asynchronous FIFO 16×8 bits
LIFObuffer	A Last-In-First-Out buffer for temporary data storage
right_shifter	Right shifter with 8-bit delay
LFSR	A Linear Feedback Shift Register for generating pseudo-random sequences
barrel_shifter	A barrel shifter for rotating bits efficiently
RAM	8x4 bits true dual-port RAM
ROM	A Read-Only Memory module for storing fixed data

Miscellaneous Modules

Design	Description
clkgenerator	A clock generator for providing timing signals
instr_reg	An instruction register module for holding and processing CPU instructions
signal_generator	Generate various signal patterns
square_wave	A generator for producing square wave signals
alu	An ALU for 32bit MIPS-ISA CPU
pe	A Multiplying Accumulator for 32bit integer
freq_div	Frequency divider for 100M input clock, outputs 50MHz, 10MHz, 1MHz
freq_divbyeven	Frequency divider that divides input frequency by even numbers
freq_divbyodd	Frequency divider that divides input frequency by odd numbers
freq_divbyfrac	Frequency divider that divides input frequency by fractional values
calendar	Perpetual calendar with seconds, minutes, and hours
traffic_light	Traffic light system with three colors and pedestrian button
width_8to16	First 8-bit data placed in higher 8-bits of the 16-bit output
synchronizer	Multi-bit mux synchronizer
edge_detect	Detect rising and falling edges of changing 1-bit signal
pulse_detect	Extract pulse signal from the fast clock and create a new one in the slow clock
parallel2serial	Convert 4 input bits to 1 output bit
serial2parallel	1-bit serial input and output data after receiving 6 inputs

LLM for RTL Generation **Methodologies**

Using **commercial** LLMs → circuit **privacy concerns**

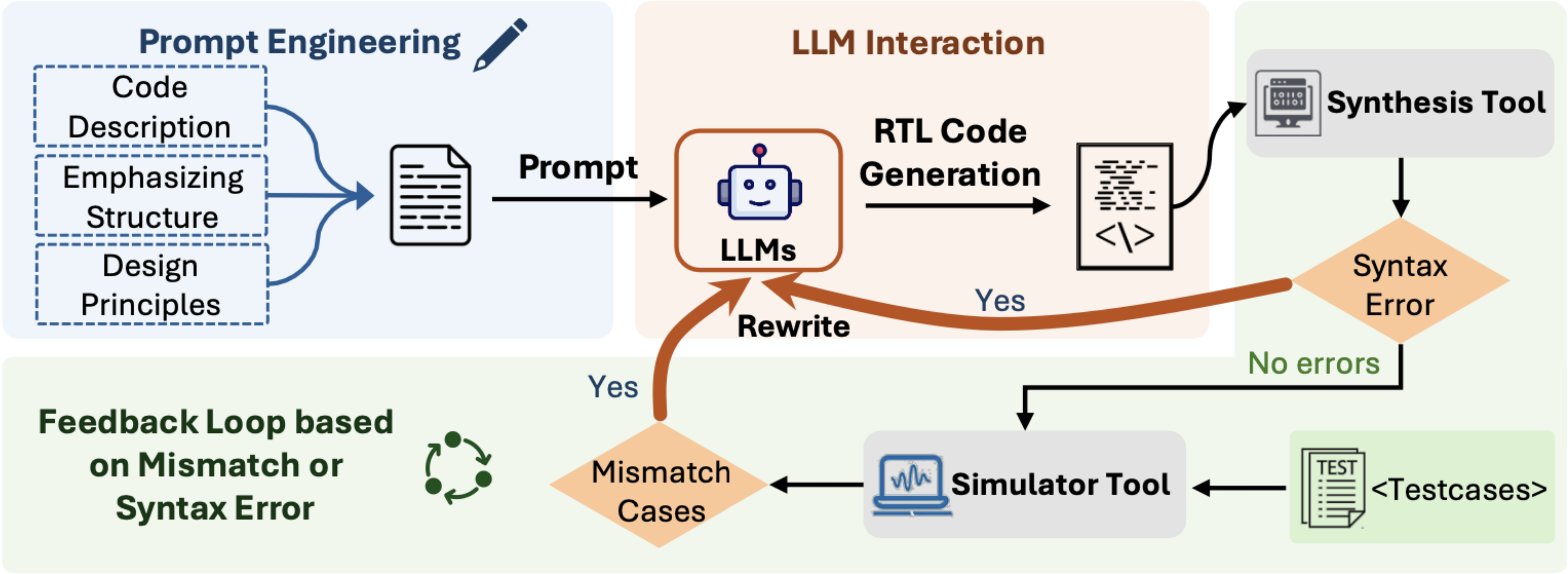
1. Prompt engineering on commercial LLMs.

Using **open-source** LLMs → allows **local deployment**

2. LLMs fine-tuned on <u>private</u> datasets with instruction-code pairs
3. LLMs fine-tuned on <u>open</u> datasets with code only
4. LLMs fine-tuned on <u>open</u> datasets with instruction-code pairs

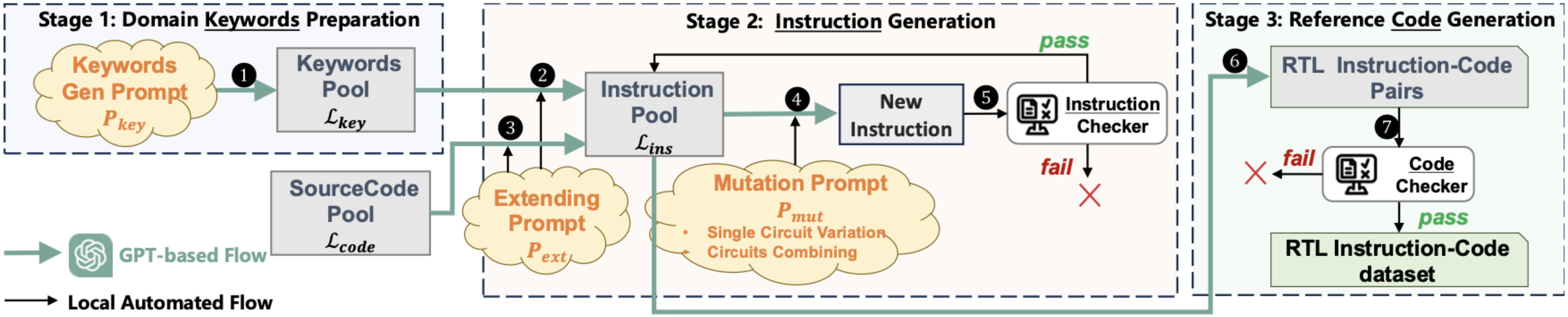
Challenge: How to get the **dataset**?

Basic flow using prompt engineering



- Input specification + structure analysis and design principles (in prompt)
- Feed prompt into LLMs → RTL code
- Incorporate the feedback from EDA tools into the flow for **rewriting**

Generation of RTL Code Dataset

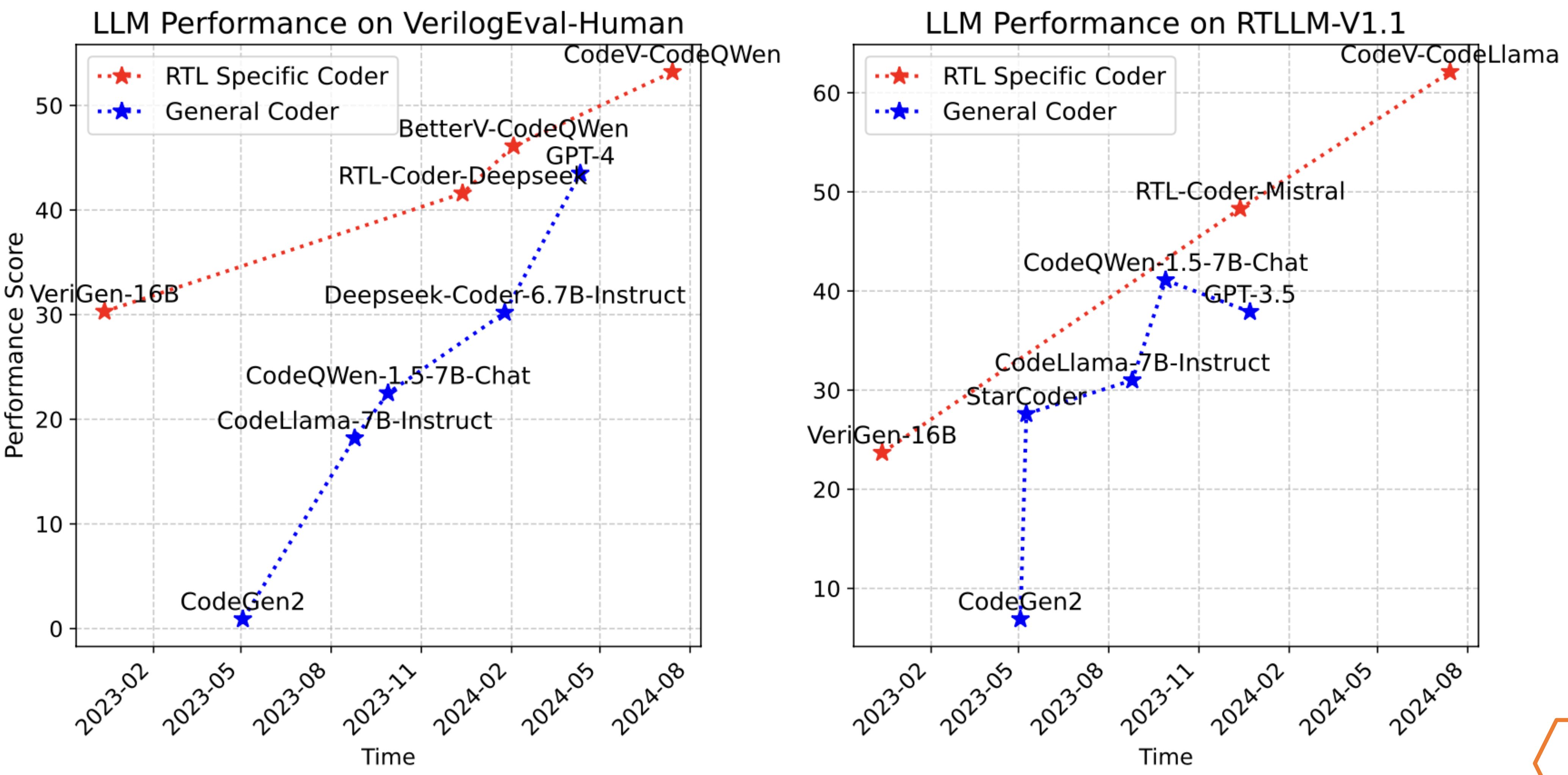


- Data generation flow of RTLCoder, as an example
- Other works adopt similar methodologies for dataset generation

1. Generate diverse **instructions** (design specifications)
2. Generate high-quality reference **code**
3. Collect the **instruction-code pairs** for (supervised) fine-tuning

[1] RTLCoder: Fully Open-Source and Efficient LLM-Assisted RTL Code Generation Technique, **TCAD'25.**

Performance in RTL Generation



Other Directions Besides Code Generation

In addition to LLMs for Hardware (RTL or HLS) Generation:

- LLMs for Hardware (Code) Optimizations
- LLM for Hardware (Code) Verification
- LLMs for Hardware (Code) Security
- …
- LLM for Design Flow Automation
- LLM for Layout Design
- LLM for Analog Design

Challenges & Room for Improvement

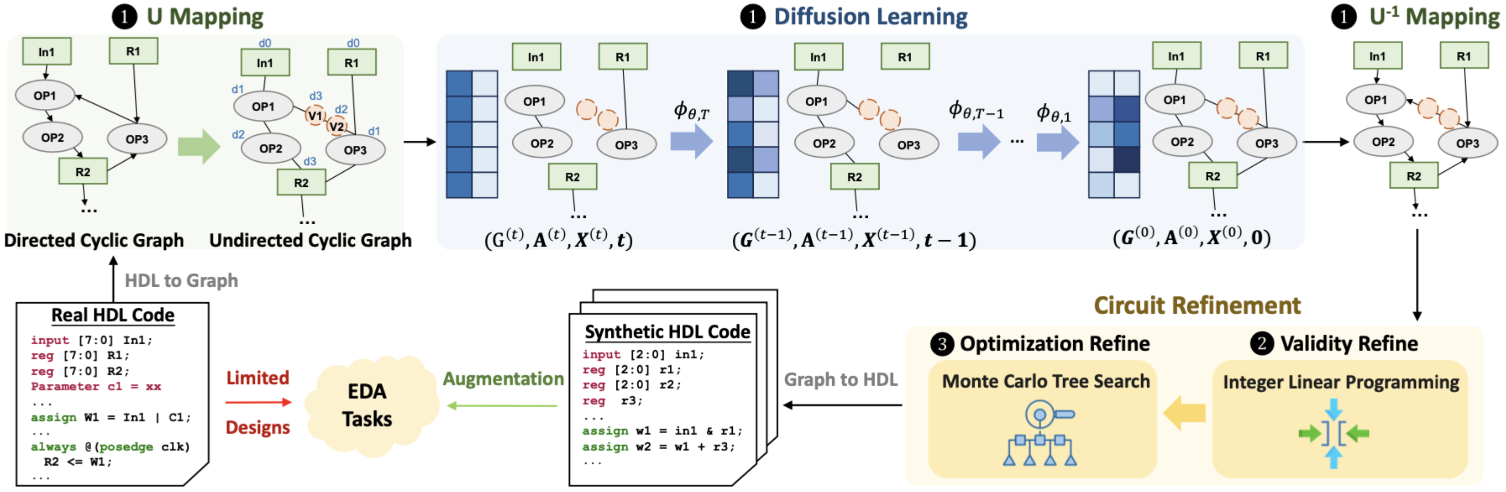
1. Circuit Foundation Model Generalization and Scalability

2. Circuit Data Availability

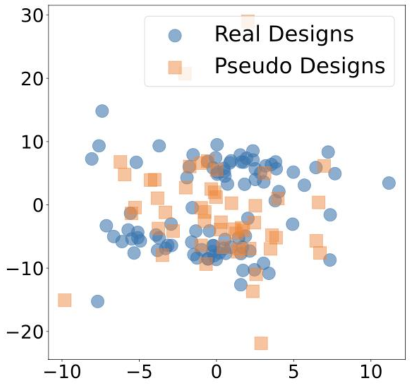
3. Bridging the Gap Between Circuit Encoder and Decoder

Lack of Circuit? Generate Synthetic Circuits

- **Solution**: Generate **synthetic pseudo-circuits** for foundation model training



- **Real circuit** designs are **private**
- **Synthetic circuit generation** based on graph generation models
- **Synthetic circuits** enable "big data"



Synthetic designs are similar to ***real designs***

Timing Path Group 'clk' (max_delay)

Level of Logic:	43
Critical Path Length:	3.264
Critical Path Slack:	-2.732
No. of Violating Paths:	3867
……	……

Cell & Pin Count

Pin Count:	640145
……	……
Leaf Cell Count:	173466

Synthetic designs reach >100K cells

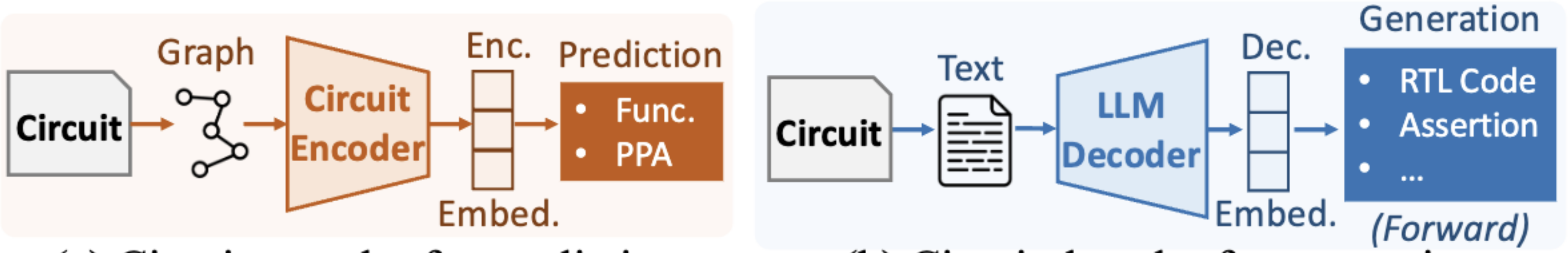
	Target	R	MAPE	RRSE
No Pseudo-Circuits	WNS	NA	52 %	2.1
GraphRNN [27]		0.71	42 %	1.7
DVAE [28]		0.75	77%	2.6
CircuitGen		0.88	36 %	1.3

Pseudo-designs can boost AI accuracy in IC prediction

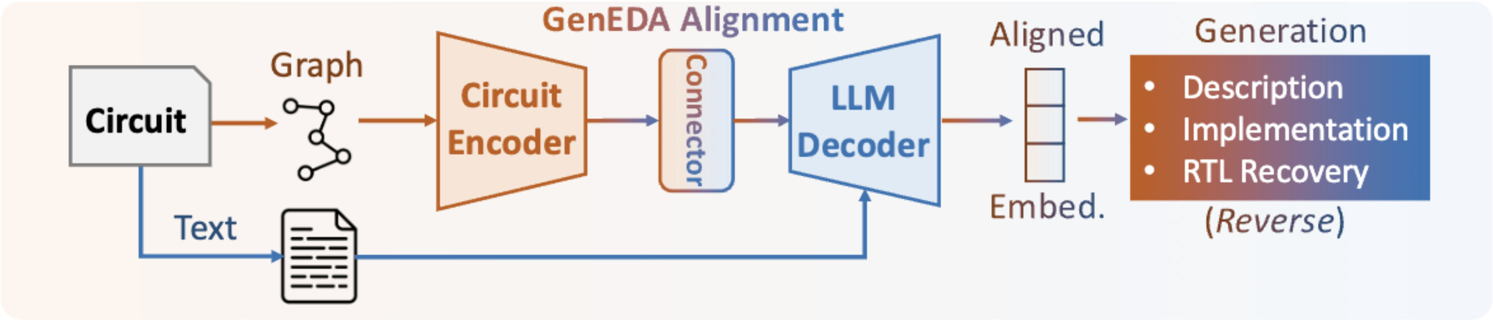
[1] Towards Big Data in AI for EDA Research: Generation of New Pseudo Circuits at RTL Stage, **ASPDAC'25**
[2] SynCircuit: Automated Generation of New Synthetic RTL Circuits Can Enable Big Data in Circuits, **DAC'25**

Bridging the Gap Between Circuit Encoder and Decoder

- An Encoder-Decoder framework with connectors

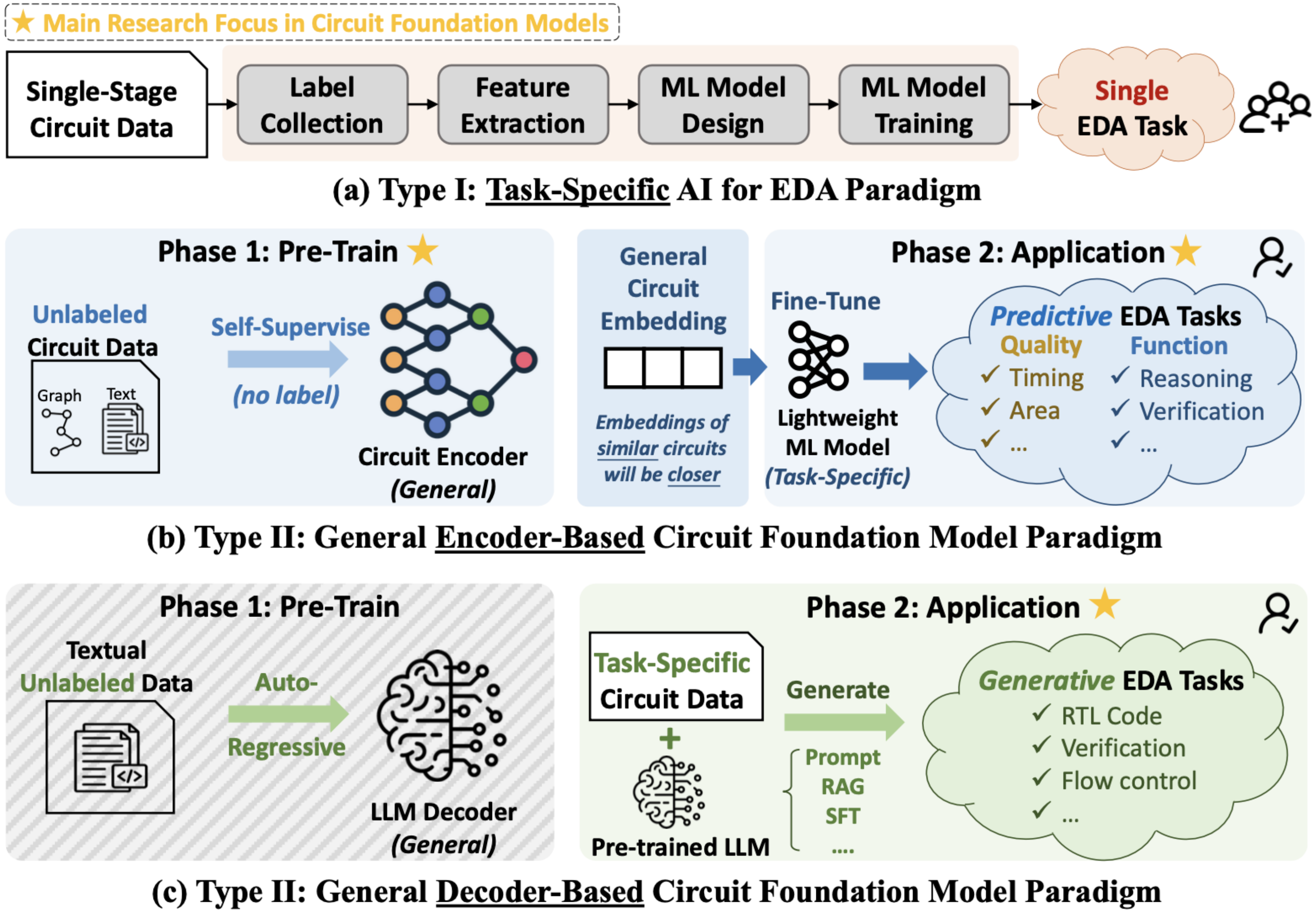


(a) Circuit encoder for prediction (b) Circuit decoder for generation



(c) Our proposed circuit encoder-decoder framework

Takeaway: Paradigms of AI for EDA



62

THE CHIPS
TO SYSTEMS
CONFERENCE

SPONSORED BY CEDA IEEE Council on Electronic Design Automation

sigda acm