Efficient Inference for Large Language Models
Algorithm, Model, and System

Xuefei Ning[1] Guohao Dai[2,4] Haoli Bai[3] Lu Hou[3] Yu Wang[1] Qun Liu[3]






[1]Tsinghua University [2]Shanghai Jiao Tong University [3]Huawei [4]Infinigence-AI

foxdoraame@gmail.com, daiguohao@sjtu.edu.cn, baihaoli@huawei.com, houlu3@huawei.com, yu-wang@tsinghua.edu.cn, qun.liu@huawei.com

Tutorial Website: https://haolibai.github.io/emnlp-2025-tutorial-efficiency/

Contents

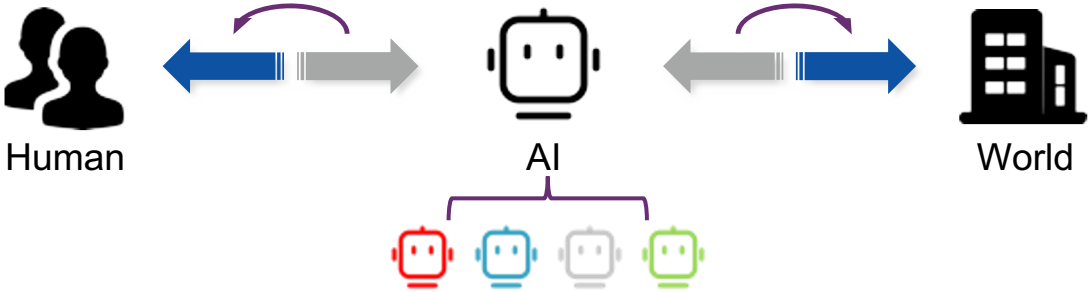
Contents

Contents

Background

Towards general and generative intelligence, scaling up model / data / computation based on Transformer is a mainstream and effective pathway.

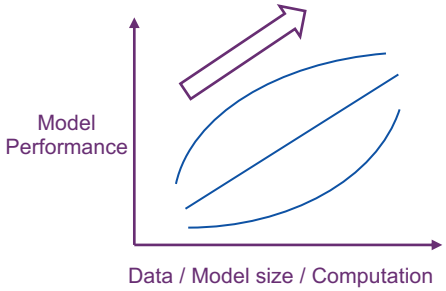
Focus Shift 1: Discriminative => Generative



Focus Shift 2: Specialized => General

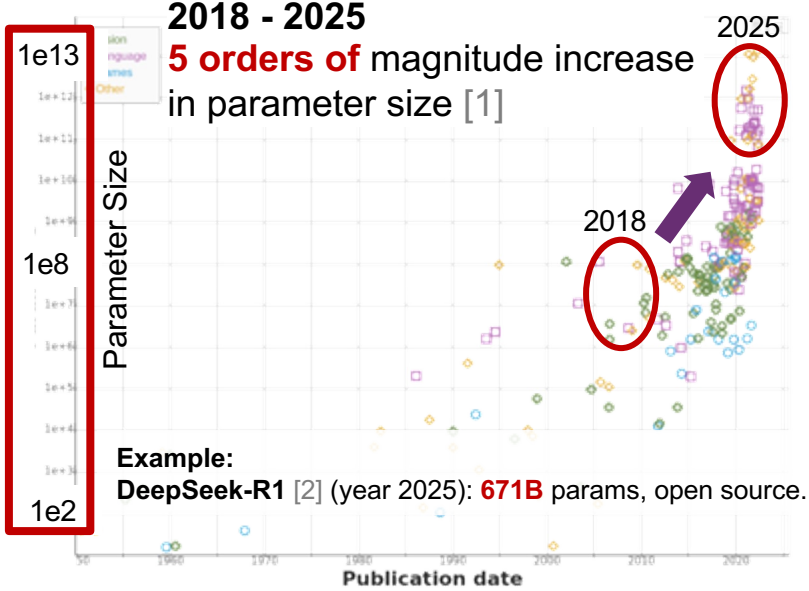
Current mainstream path

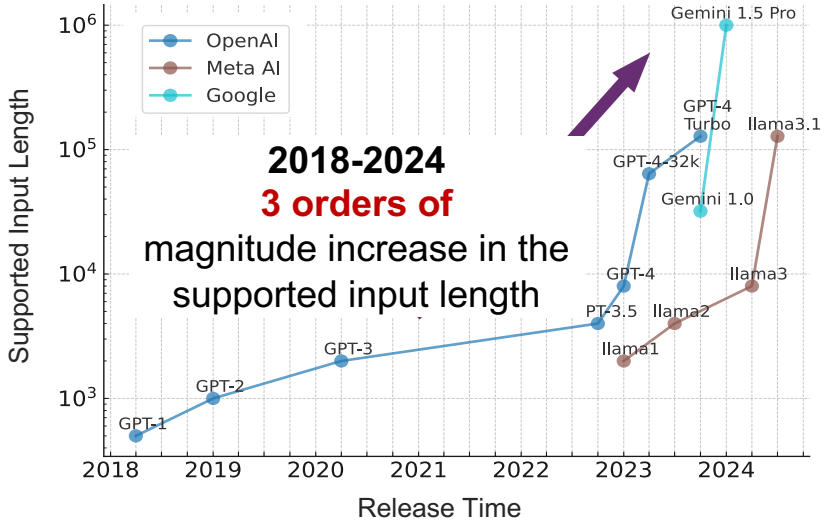
Based on the Transformer architecture, scaling model size , training data / computation, and test computation.



Background

The model scale and input/output length in generative intelligence research and applications have increased significantly.





[1] Villalobos et al. “Machine Learning Model Sizes and the Parameter Gap.” arXiv 2022.
[2] Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.*arXiv 2025*.
[3] Esser, Patrick et al., Scaling rectified flow transformers for high-resolution image synthesis, ICML 2024.
[4] Achiam, Josh, et al. "Gpt-4 technical report." arXiv 2023.
[5] Reid, Machel, et al. "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context." arXiv 2024.

Background

Application demands e.g., multimodal input, advanced tasks, may continually drive increases in model scale and input/output length.

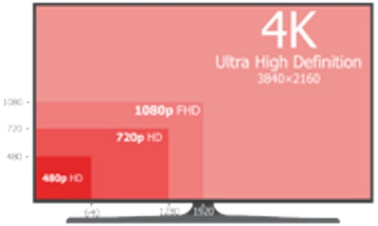
Multimodal Input

Modeling multimodal data *may* push the scaling saturation point of model size higher.

~100B ⟹ ~1000B ?

texts → texts, images, videos, audios, …

Supporting high-resolution images and longer videos requires extended input/output contexts.



Advanced Tasks

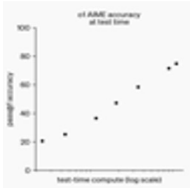
Agentic pipelines for broad applications require stronger models and longer input/output contexts.

Single-Agent: Memory, Planning, Action

Multi-Agent Collaboration

Test-time compute scaling (especially CoT) for reasoning require longer input/output contexts.

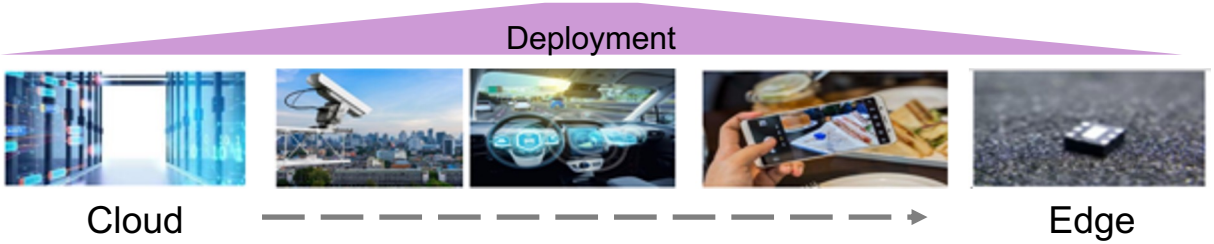
OpenAI o1



Application Challenge

This scaling of the model size and input/output poses challenges for efficient inference across platforms and application scenarios.

Model size, input/output length ⇨ Inference cost (time, storage, energy)



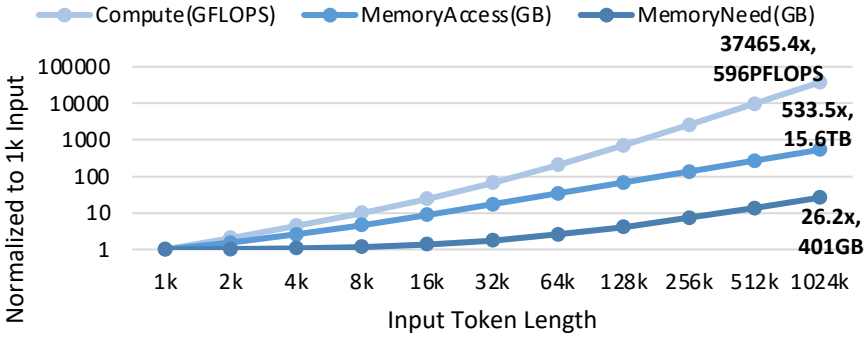
Requirements or constraints of application scenarios and platforms

High Throughput	Low Latency	Small Storage	Low Energy Consumption

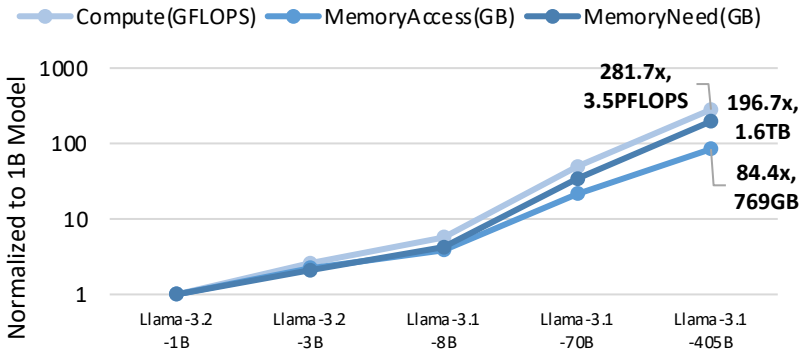
Application Challenge

Time to first token (TTFT) and time per output token (TPOT) w.r.t. model size and input/output lengths, estimated with three costs (compute, memory access and memory footprint) of the model spec and device spec.

Different #Prefill Tokens using Llama-3.1-8B (BS=1)



Different Models with 4k Input Tokens (BS=1)



8B Model	Compute Amount	Memory Access	VRAM Consume	GPU	TTFT*	TPOT*
1K token	15.9 TFLOPs	29.2 GB	15.3 GB	1xA100	73~170 ms	10~24 ms
1M token	**597 PFLOPs**	**15.6 TB**	**401 GB**	**6xA100**	7.6~17.7 min	32~75 ms

4K Tokens	Compute Amount	Memory Access	VRAM Consume	GPU	TTFT*	TPOT*
1B Model	12.4 TFLOPs	19.2 GB	15.3 GB	1xA100	57~132 ms	1.8~4.3 ms
405B Model	**3.5 PFLOPs**	**1.6 TB**	**769 GB**	**10xA100**	1.6~3.7 s	55~128 ms

*TTFT is estimated using **Compute Amount** / (**Peak OPS** x **compute-util**), as prefill is computation-bounded. We assume a compute utilization range 30%~70% to report the estimation.
TPOT is estimated using **Memory Access** / (**Bandwidth** x **bandwidth-util**), as decoding is memory-bounded. We assume a bandwidth utilization range 30%~70% to report the estimation.
A100 Peak Compute Performance (FP16) = 312 TFLOPS; Peak Bandwidth = 2 TB/s

Application Challenge

Per-request energy consumption w.r.t. model size and input/output lengths, *estimated* with actual latency measurement, device & schedule assumptions, and device spec.

Model	Company; Host (Device)	Date	*Estimated* energy consumption (100in-300out) (Wh)	*Estimated* energy consumption (10kin-1.5kout) (Wh)
GPT-4.1 nano	OpenAI; Azure (H200& H100)	Apr, 2025	0.10±0.04	0.45±0.21
GPT-4.1 mini			0.42±0.20	1.59±0.80
GPT-4.1			0.92±0.50	4.23±1.97
LLaMA-3.1-8B	Meta; AWS (H200& H100)	Jul, 2024	0.10±0.02	0.60±0.09
LLaMA-3.1-70B			1.10±0.13	11.63±1.39
LLaMA-3.1-405B			1.99±0.32	20.76±1.80

Jegham et al. made an attempt to estimate the environmental footprint of LLM inference at per-prompt level of commercial AI providers, based on assumptions on the infrastructure and scheduling.

[1] Jegham, Nidhal, et al. "How hungry is ai? benchmarking energy, water, and carbon footprint of llm inference." *arXiv preprint arXiv:2505.09598 (2025).*

Application Challenge

<u>Overall energy consumption</u>, *estimated* with reported usage, device & schedule assumptions, and device spec.

A rough estimation result: Assuming a total of 772 billion queries (estimated with OpenAI 2024 report and the usage growth pattern, assume 80% short queries) annually in 2025, GPT-4o inference require approximately 4×10^{11}**Wh**, exceeding the total electricity consumption of **35,000 U.S. residential households**.

Jegham et al. made an attempt to estimate the environmental footprint of LLM inference at per-prompt level of commercial AI providers, based on assumptions on the infrastructure and scheduling.

[1] Jegham, Nidhal, et al. "How hungry is ai? benchmarking energy, water, and carbon footprint of llm inference." *arXiv preprint arXiv:2505.09598 (2025).*

Contents

Contents

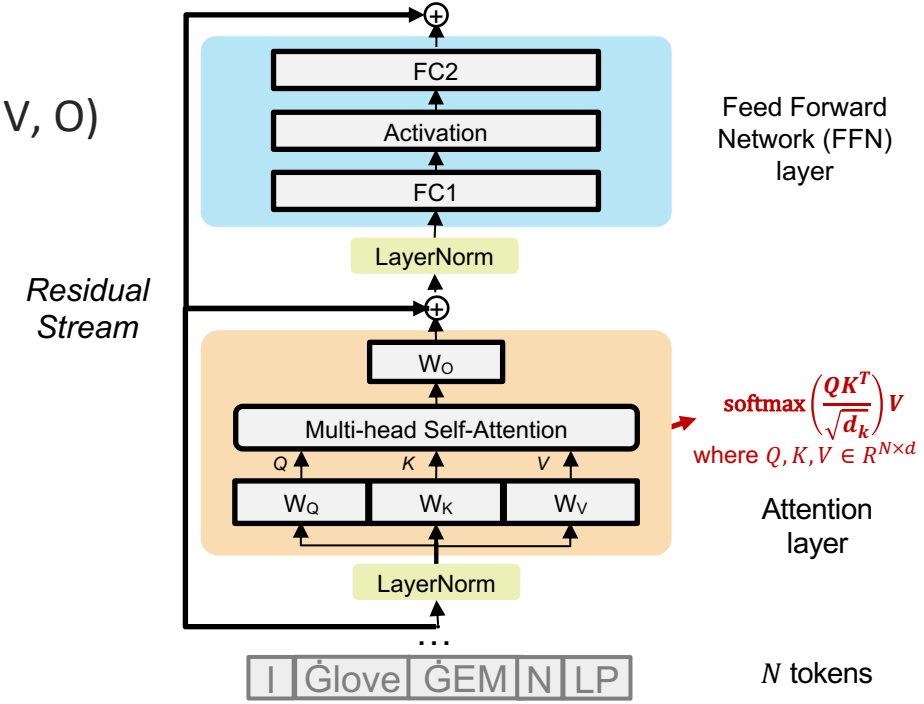
2 Preliminary

Contents

2 Preliminary

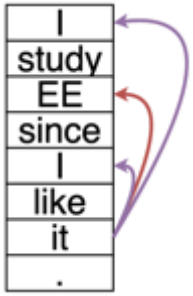
Model: Architecture of LLMs

- Most LLMs are based on Transformer architecture[1], consisting of an input embedding layer, Transformer blocks, and a decoding layer.
- A Transformer block consists of:
  - Attention-Linear (transform for Q, K, V, O)
  - Multi-Head Self-Attention
  - Feed Forward Network (FFN)
  - Layer Norm

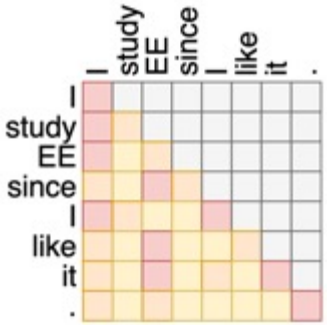


[1] Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).

Attention Layer

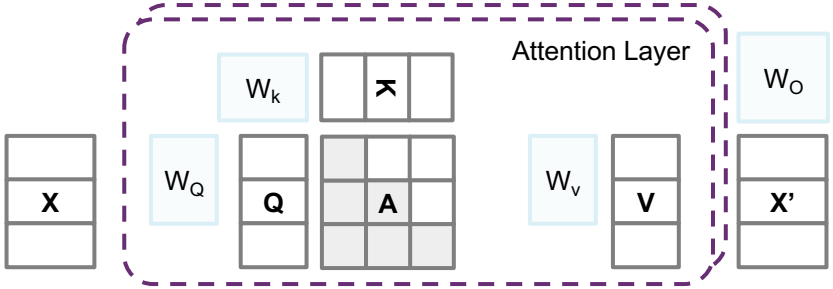


The idea of attention module is to establish **token-to-token "attention" relationships** within a sequence.



This relationship is modeled by an **attention matrix**, where each row represents one token's attention distribution to **previous** tokens (sum up to 1).

Causal attention mask: Each token only has positive attention to previous tokens



Calculation of each attention head:

- $Q = XW_Q; K = XW_K; V = XW_V$ to map $X \in \mathbb{R}^{N\times D}$ to $Q, K, V \in \mathbb{R}^{N\times d}$
- Calculate inner product $S = QK^T$, apply mask and softmax to get the attention matrix $A = \text{softmax}(S + M) \in \mathbb{R}^{N\times N}$
- $X' = AV$ to get the output

Generative Modeling Algo: Autoregressive Model

- Contemporary LLMs are generative models that uses the "autoregressive" generative modeling method.

- The **core task of generative modeling** is to learn a parametrized model $p_\theta(\boldsymbol{x})$ from observed data $\{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$, which in some sense capture the unknown real distribution of data $p_{\text{data}}(\boldsymbol{x})$, and can do stochastic sampling (i.e., sample generation) & probabilistic inference (e.g., likelihood estimation).

- **Autoregressive models** are a family of generative modeling methods that models the joint probability of a token sequence $\boldsymbol{x} = [x_1, x_2, \dots, x_n]$ as a product of conditional probability distributions, each conditioned on the preceding tokens:

$$p(x_1, x_2, \dots, x_n) = p(x_1) \prod_{i=2}^{n} p(X_i = x_i | x_1, \dots, x_{i-1})$$

Sampling Process of LLMs

LLMs are autoregressive models that uses one transformer to model the conditional distributions.

- Denoting the vocabulary set as $|V|$, the transformer maps the token sequence $x_{<i} \in V^{i-1}$ to a sequence of logits $l \in \mathbb{R}^{(i-1)\times|V|}$, where the logits $l[i-1] \in \mathbb{R}^{|V|}$ corresponding to x_{i-1} is regarded as the categorical distribution's parameter of x_i.

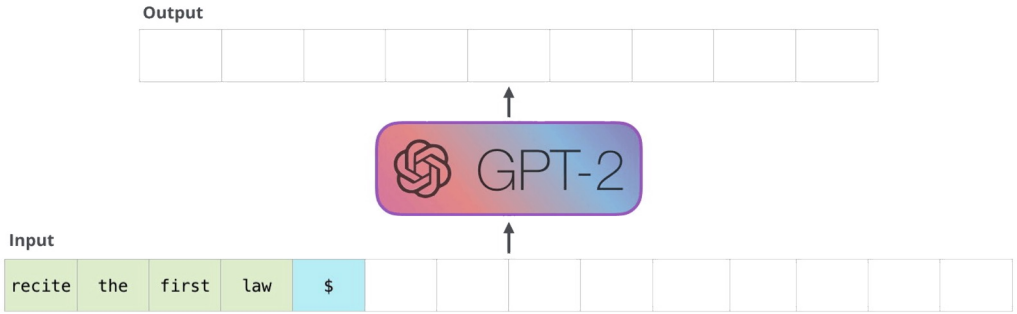
The KV cache design for sampling from causal LLMs:

- At first glance, in each step of autoregressive sampling: $x_i \sim p_\theta(x_{<i})$, the model forward process has $O((i-1)^2)$ computation complexity. Thus the overall sampling process (assume N steps) has $O(N^3)$ complexity.
- Most contemporary LLMs choose to use causal attention: The calculation of features of x_i only attends to $x_{<i}$. In this way, newly sampled tokens $x_{>i}$ don't influence the features corresponding to x_i.
- This enables us to "cache" already calculated features (specifically, the Key and Value) of preceding tokens to avoid recalculating their features, thus reduce overall computation complexity of sampling to $O(N^2)$.

Sampling Process of LLMs

After applying the KV cache technique, a typical LLM inference/sampling* process can be divided into two stages:

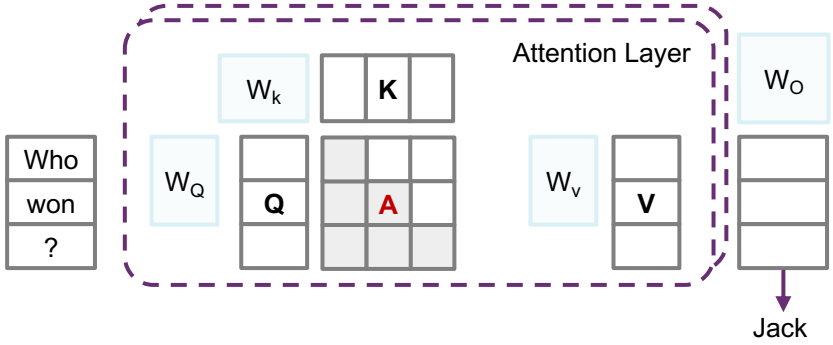
- **Prefilling Stage**: The LLM calculates and saves the KV cache of the initial input tokens, and samples the first output token.
- **Decoding Stage**: The LLM samples the output tokens one by one with the KV cache, and in the meantime updates the KV cache.



*In this tutorial, the terms “inference” and “sampling” will be used interchangeably. Although they differ in the context of probabilistic modeling, here they both refer to either a single sampling step—i.e., a forward pass of the model—or the overall sampling process, depending on context.

Attention Computation in Prefill/Decode Stages

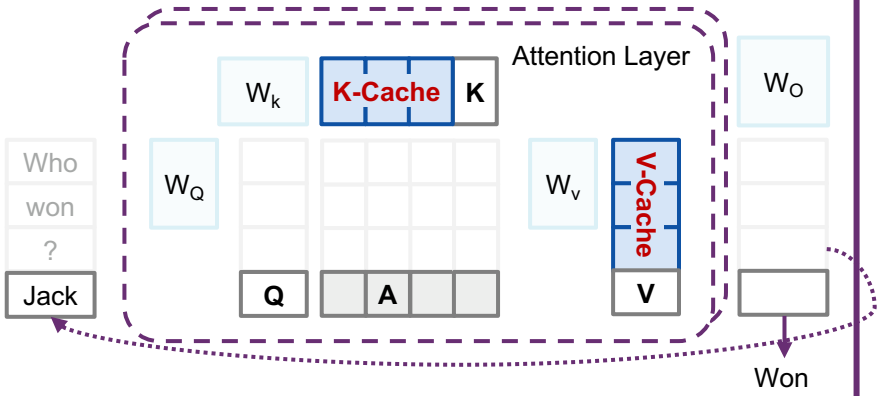
Prefill



The LLM calculates and saves the KV cache of the initial input tokens, and samples the first output token.

Calculate N query, key, value; calculate $N \times N$-sized attention matrix A.

Decode



The LLM samples the output tokens one by one with the KV cache, and in the meantime updates the KV cache.

Calculate 1 query, key, value; read N key, value from KV cache; calculate $1 \times (N+1)$-sized attention matrix A.

Contents

Software, Hardware, Instruction

Let's first have an overview of the most basic terms of interest:

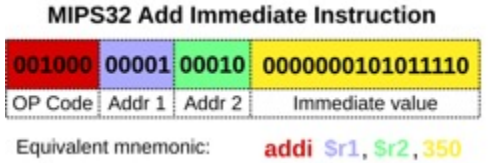
- What is a computing hardware? **Hardware consists of microelectronic components that transfer and transform electrical signals.** Through these physical operations, **hardware realizes high-level abstractions, that is executing functionalities described by instructions, such as storing, transferring, and transforming data.**

- What is software? **All software** -- no matter it is an OS, a game, or an AI model's inference engine -- **is a structured pack of instructions and data**.

- **Instructions are the interface between software and hardware**. **Instruction Set Architecture (ISA)** defines this instruction interface: what instructions are available, how they are encoded and executed.

Instruction Set Architecture (ISA)
Define the instruction interface of a device

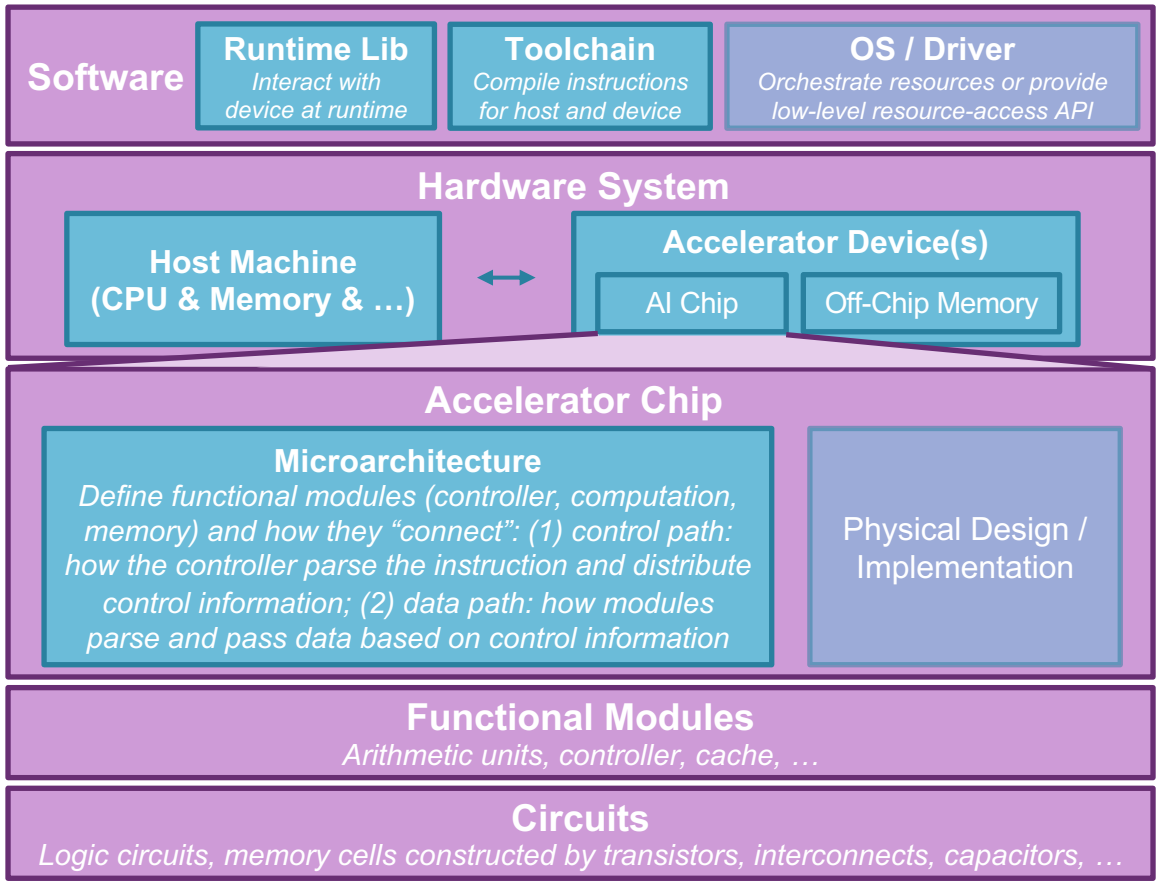
Example 1: CPU ISA defines instruction format and meaning. [1]
Typical ISA for CPU includes MIPS, RISC-V, x86, x86-64 (amd64), etc.



Example 2: NVIDIA Ampere GPU architecture has compute capability sm80, which indicates its supported certain instructions (ISA) and some microarchitectural features

[1] https://en.wikipedia.org/wiki/Instruction_set_architecture

A Conceptual Layered Overview



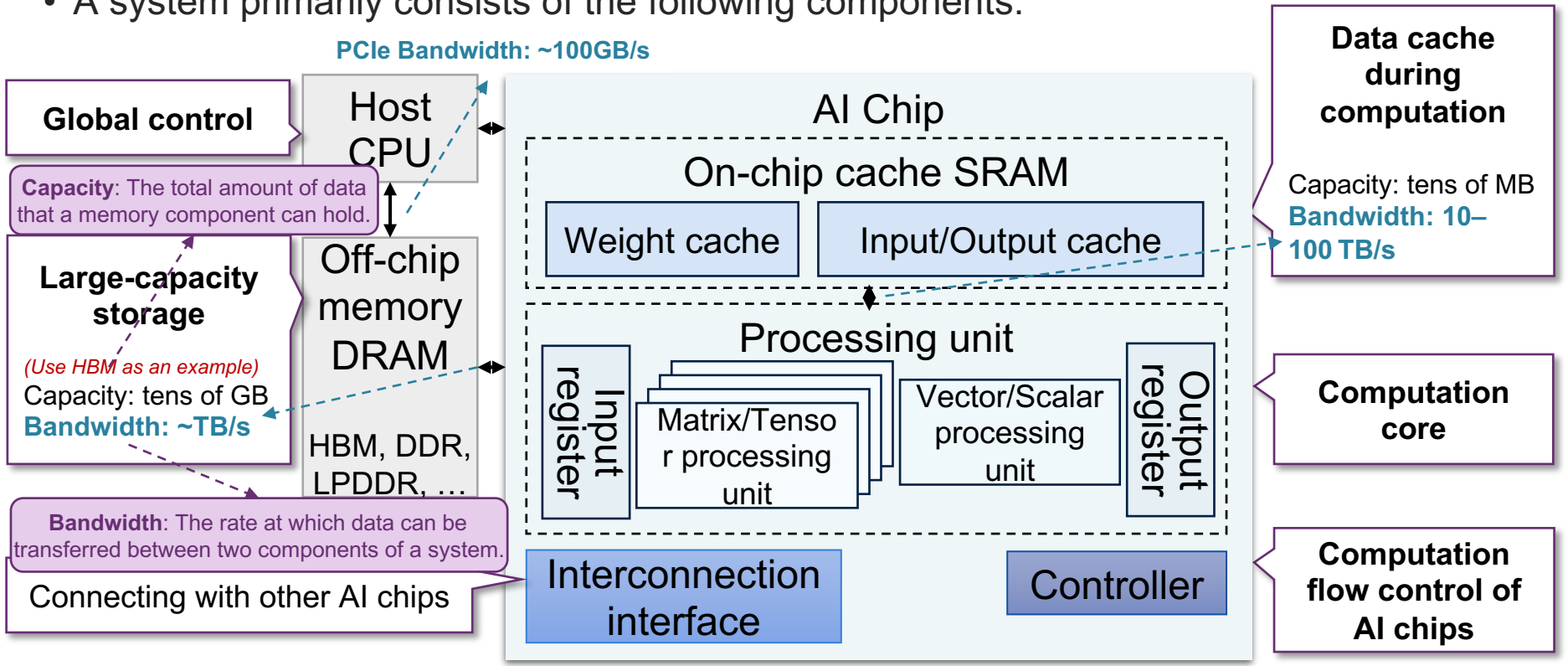
- Toolchain produces hardware-specific instructions.
- Runtime library manages program execution, send instruction to device, transfer data to/from device, and optionally call toolchain dynamically.
- A hardware system consists of host, accelerator devices, and their interconnects

Accelerator's Instruction Set Architecture (ISA)
Define the instruction interface of the AI accelerator device

- A microarchitecture design organizes modules to implement the accelerator's ISA
- A functional module combines circuits to perform certain functions
- Basic logic and memory elements

A Composition Overview

- A system primarily consists of the following components:



PPT credit: Prof. Zhenhua Zhu@Tsinghua University

Common Hardware Types

- **Common hardware types:** CPU, GPU, FPGA, ASIC
  - **Chip-level metrics/characteristics:** peak compute performance (OPS), energy efficiency (OPS/W), power (W), and area (mm²).
  - **Also need to consider**: generality and suitableness to algorithms; comprehensiveness of the software ecosystem.

CPU
Central Processing Unit
High generality
Low AI computing performance

GPU
Graphics Processing Unit
Strong parallel computing capability
High-bandwidth memory

FPGA
Field-Programmable Gate Array
Hardware programmability
Flexibility and reconfigurability

ASIC
Application Specific Accelerator
High specialization
High performance and low power consumption

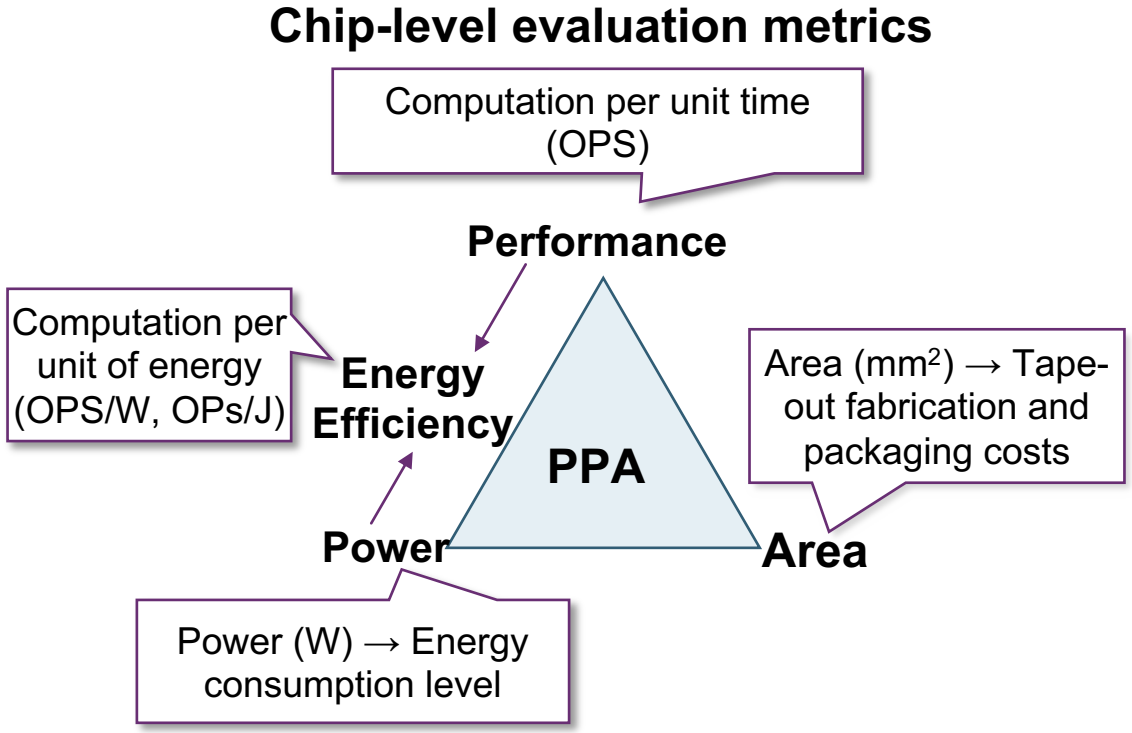
General — Specialized

More general and fine-grained ISA. Rely on software to implement coarse-grained operators, algorithms.

More specialized and coarse-grained ISA or even template-based design that hard code an algorithm.

Chip-level Evaluation Metrics

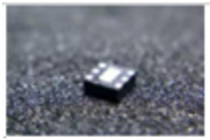
- **Chip-level metrics/characteristics**



PPT credit: Prof. Zhenhua Zhu@Tsinghua University

AI1.0 Accelerators in Different Scenarios

From cloud center to tiny edge device

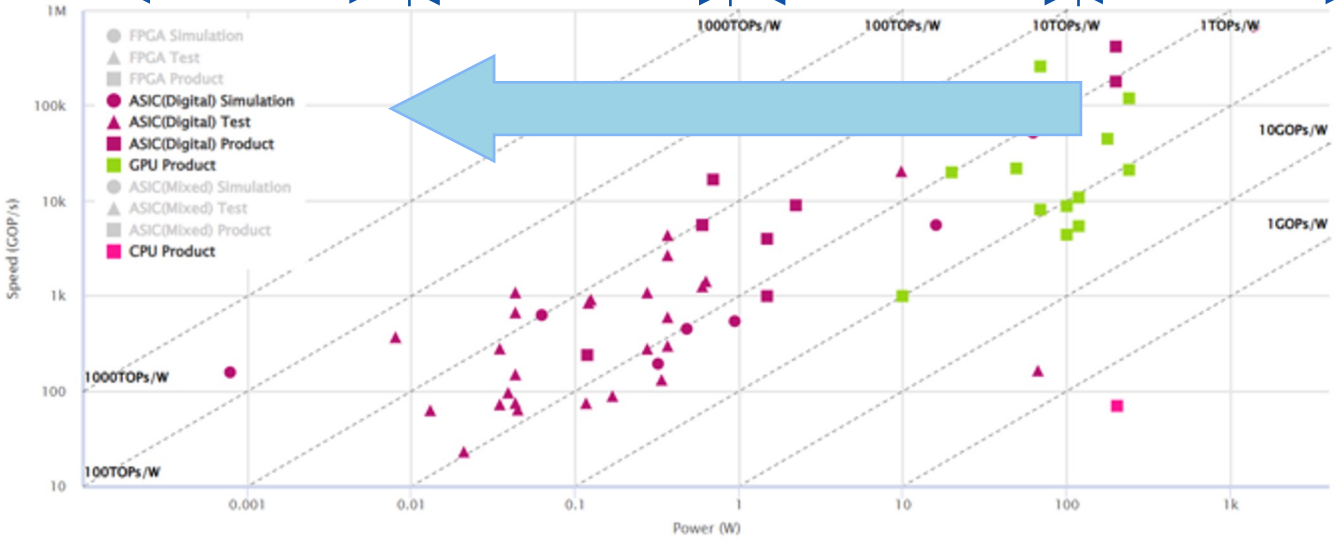

Sensor, Wearable Device


Mobile / IoT Device



Smart City / Auto-driving Car

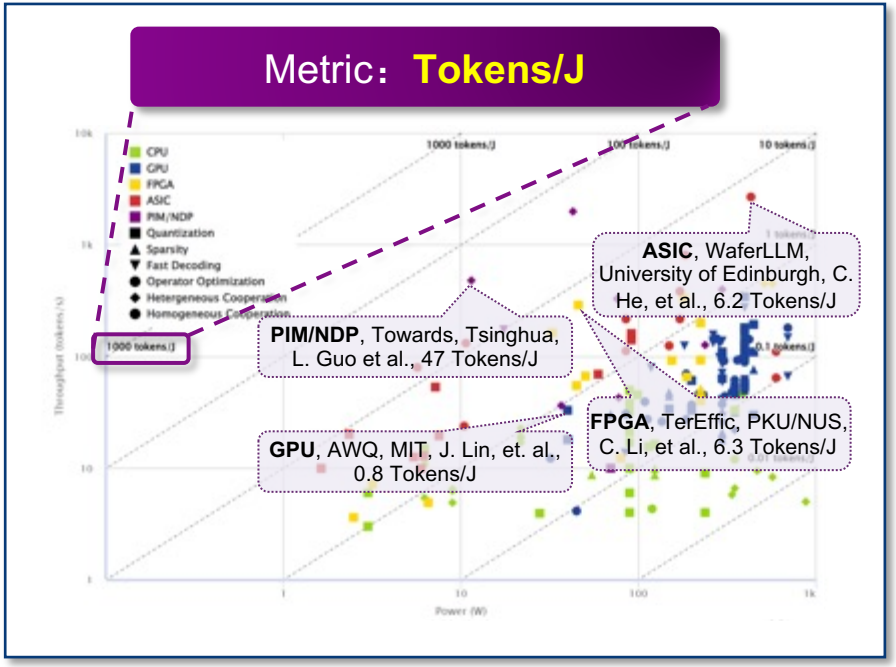

Cloud Center



Webpage

From AI 1.0 to AI 2.0: Energy Efficiency Metric

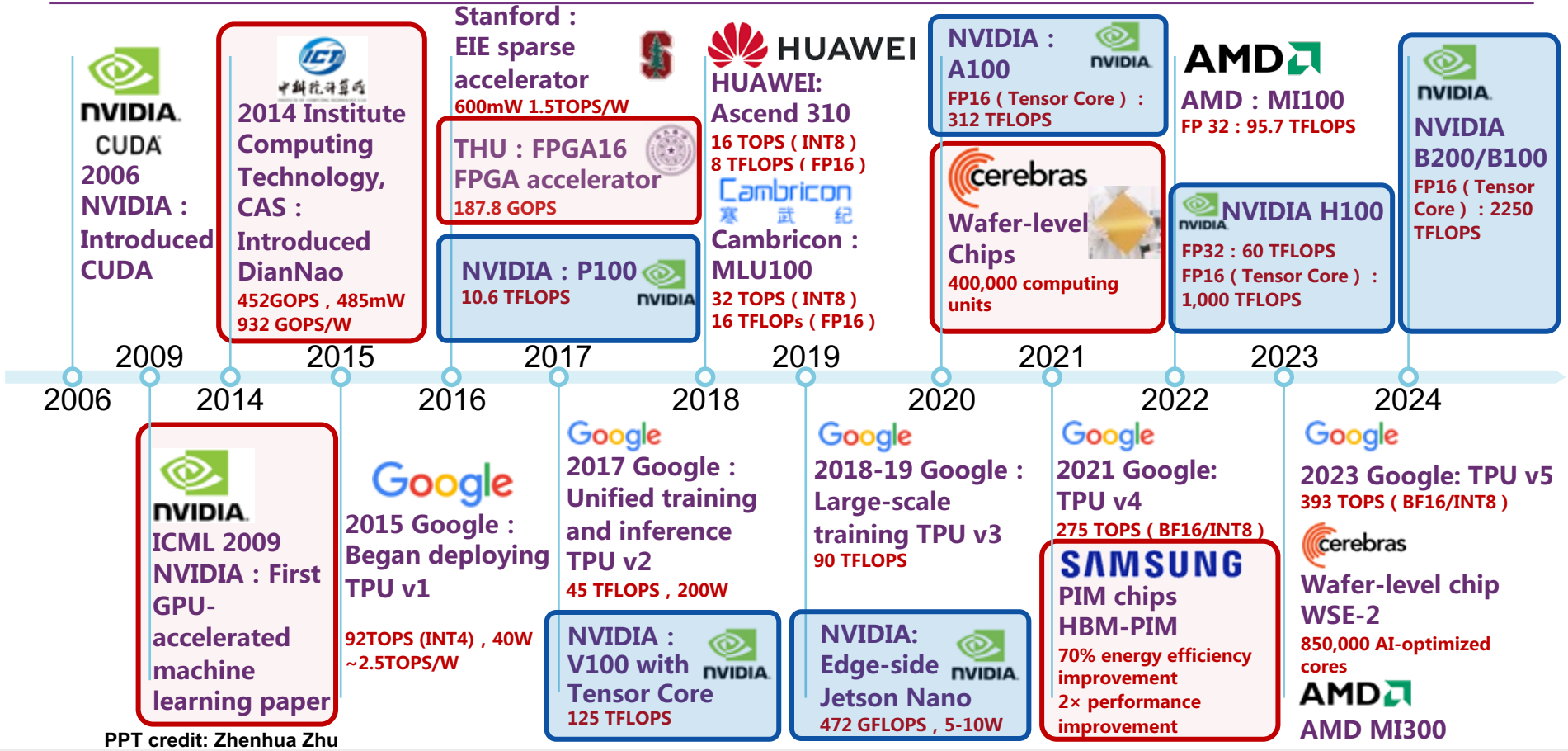
Hardware energy efficiency → Inference-system energy efficiency
TOPs/J → Tokens/J



WebPage

Paper

Basic Knowledge : Development of Chips



PPT credit: Zhenhua Zhu

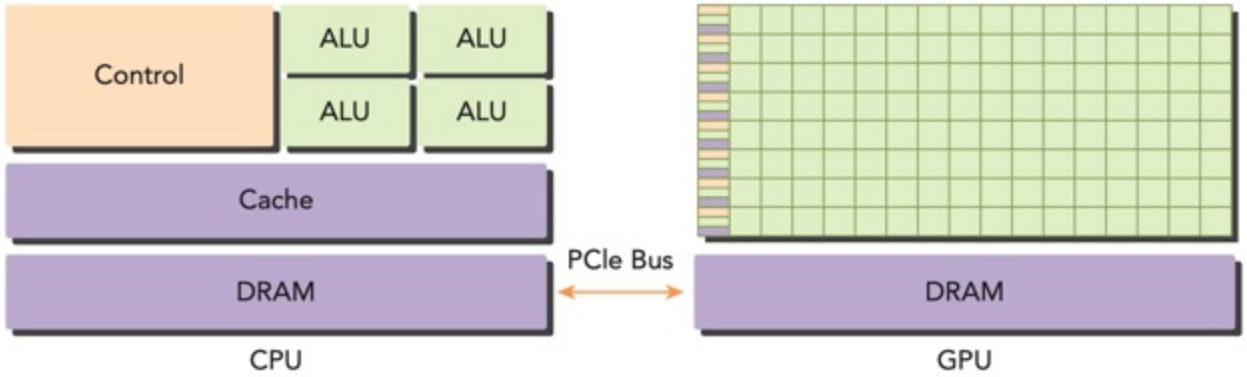
Contents

2 Preliminary

Hardware System & GPU Device

NVIDIA GPU

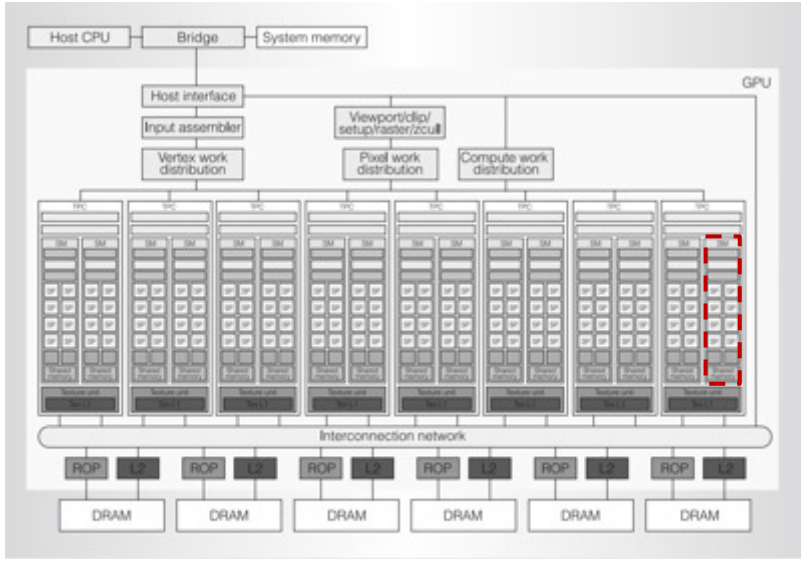
- **GPU device:**
  - GPUs use a **Single Instruction Multiple Threads (SIMT)** architecture.
  - Compared with CPUs, GPUs are better suited for programs featuring **simple control logic and large-scale parallel** computation.
- **Hardware system**: A GPU is **not an independent computing platform**, but rather a **co-processor to the CPU**.



[1] C. John, et al. "Professional CUDA C Programming."

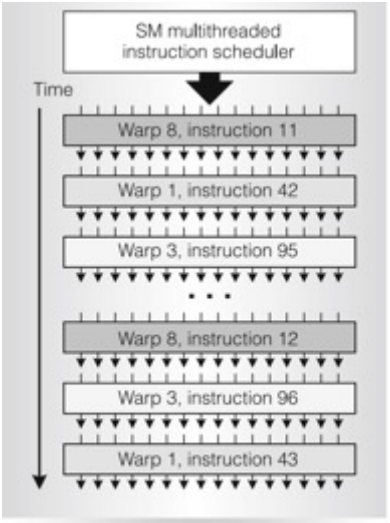
GPU Microarchitecture

- **Design Concept of GPUs**
  - **Single Instruction Multiple Threads (SIMT):** A single instruction is executed in parallel by multiple threads.



GPU Architecture

SM (Streaming Multiprocessor) serves as the basic hardware unit for parallel instructions.

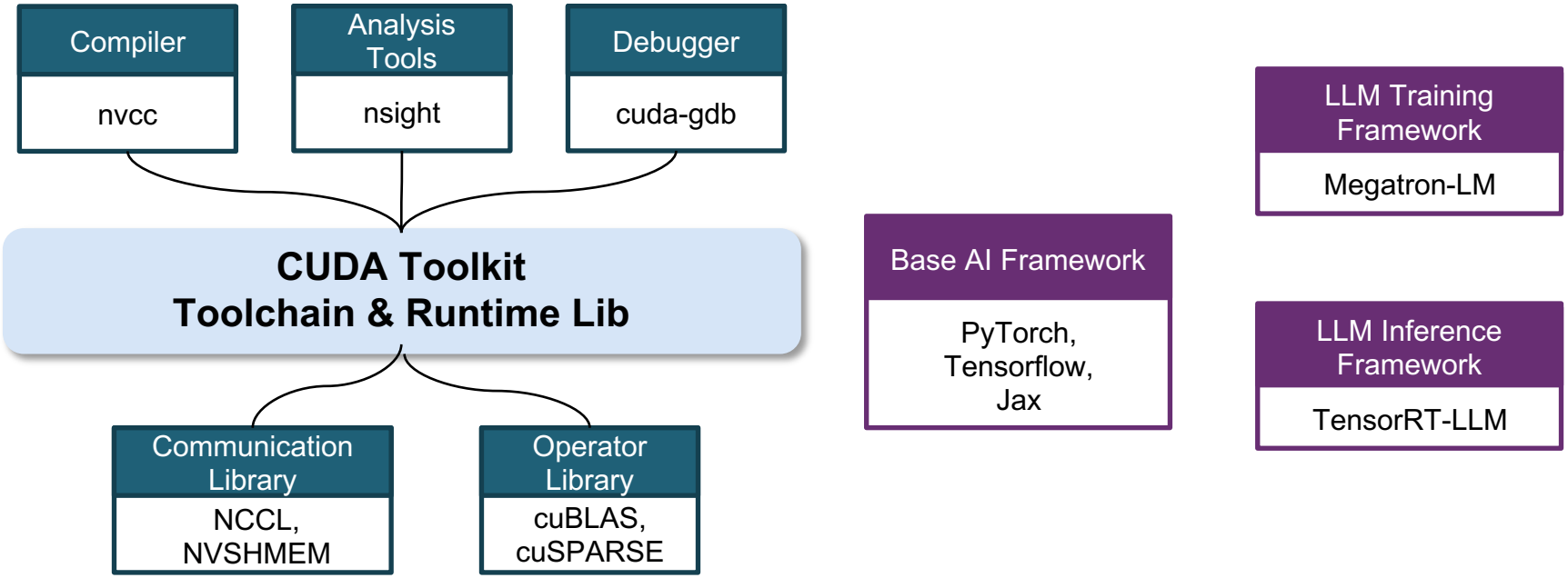


SIMT

[1] E. Lindholm, et al. ”NVIDIA Tesla: A Unified Graphics and Computing Architecture.”, in *IEEE Mirco*, 2008.

GPU Software Stack

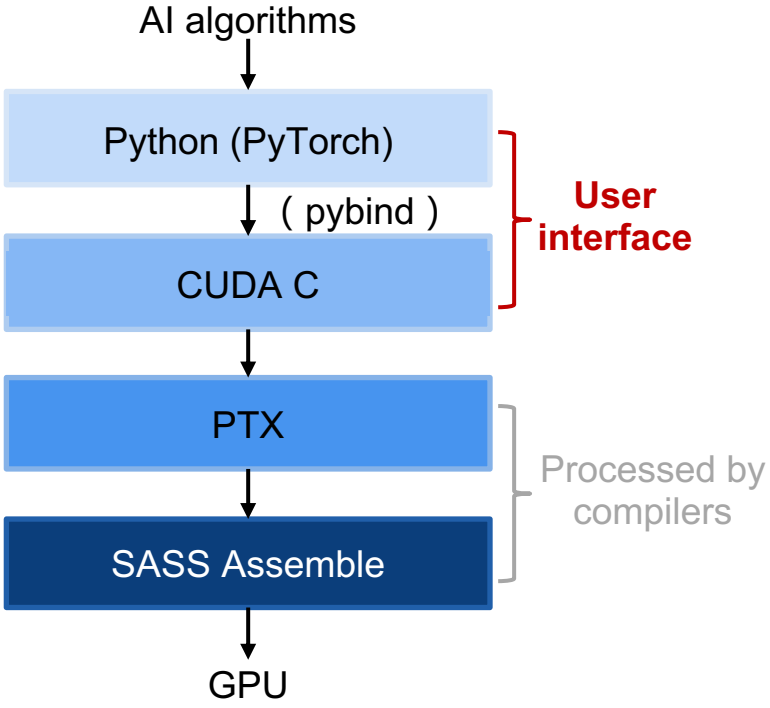
- **NVIDIA GPU Software Stack**
  - CUDA (Compute Unified Device Architecture) is the NVIDIA's GPU parallel programming platform and programming model, featuring **a rich software ecosystem**.
  - AI & LLM frameworks is built on them.



[1] M. Shoeybi, et al. " Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.", arXiv, 2019.

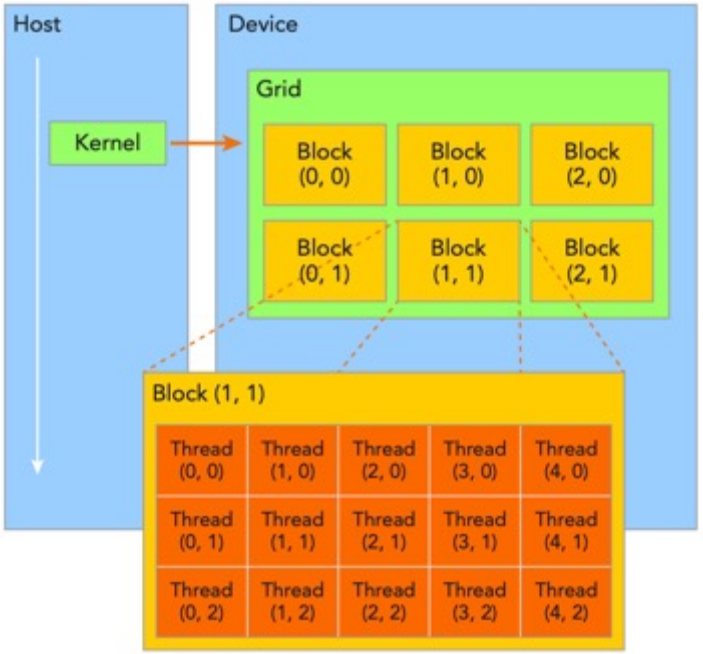
GPU Programming Model

- **NVIDIA GPU Software Stack**
  - CUDA allows writing GPU code in high-level languages such as C/C++, reducing programming complexity.



GPU Programming Model

- Programming Organization: How to Parallelize?



Thread

SIMT: Single Instruction Multiple **Threads**

Thread
Minimal parallel unit

Block
A block contains multiple threads.

Grid
A grid contains multiple blocks, encompassing all threads of a single kernel.

Kernel<<<grid, block>>>

Number of threads per block

Number of blocks per grid

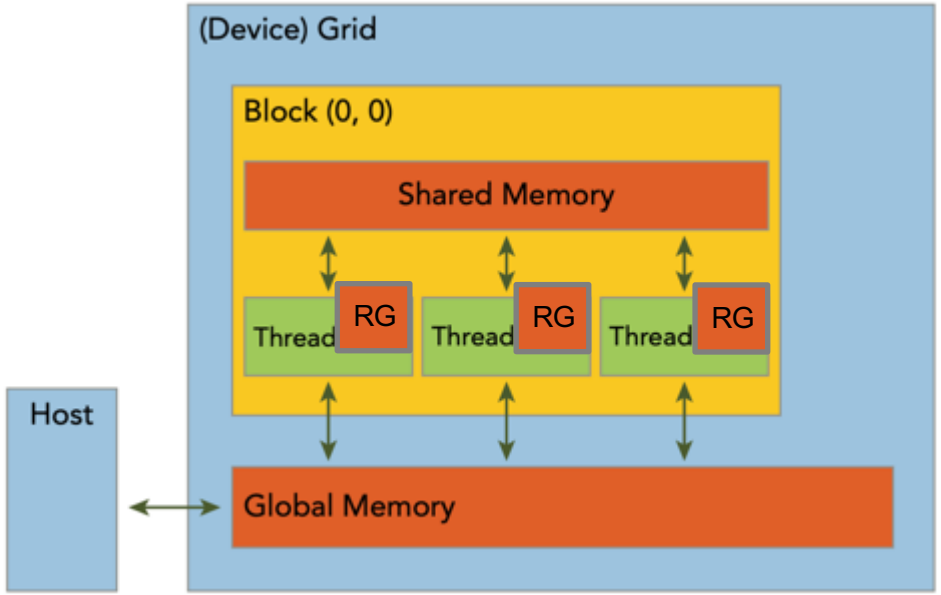
Why define the above hierarchical organization?
It corresponds to the GPU memory hierarchy!

[1] C. John, et al. "Professional CUDA C Programming."

GPU Programming Model

- **GPU Memory Hierarchy**
  - **Thread**
    - Each thread has its own **registers (RG)**.
    - Register contents are not shared between threads.
  - **Block**
    - All threads within a block share **Shared Memory**. **Can cooperate & communicate through it.**
  - **Grid**
    - All blocks access data from **Global Memory** (High Bandwidth Memory, HBM).

SIMT: Single Instruction Multiple **Threads**

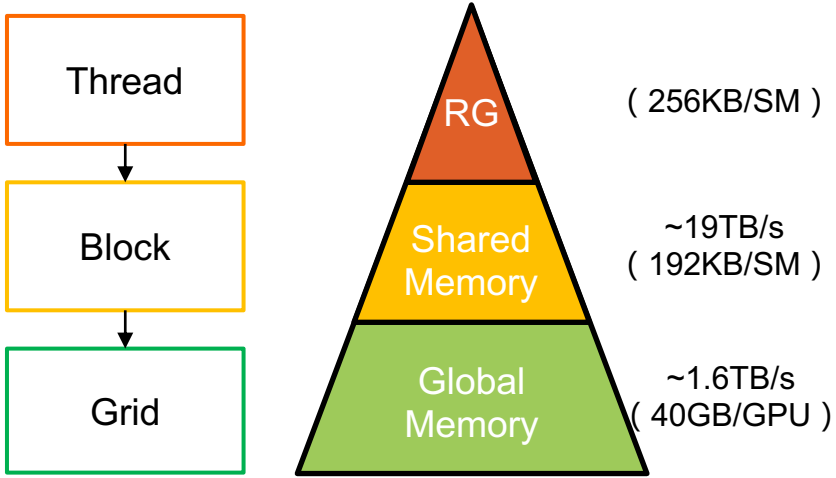


Memory Hierarchy

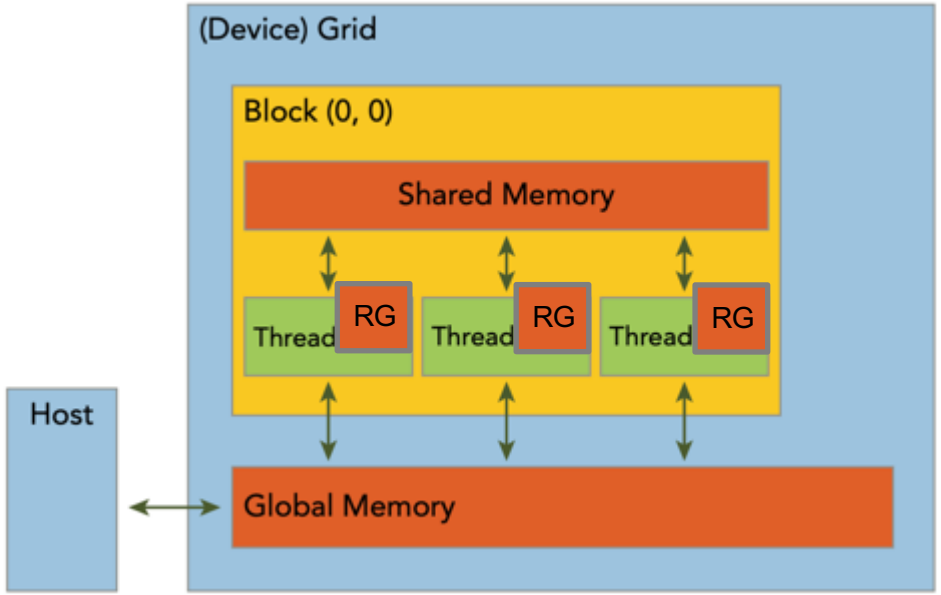
[1] C. John, et al. "Professional CUDA C Programming."

GPU Programming Model

- **GPU Memory Hierarchy**
  - From the perspective of memory access efficiency: **how threads are organized significantly impacts kernel performance.**



Bandwidth and capacity of each memory hierarchy (A100 40GB GPU)



Memory Hierarchy

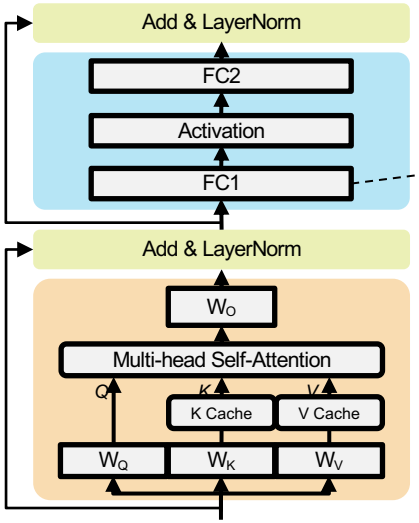
[1] C. John, et al. "Professional CUDA C Programming."

Contents

2 Preliminary

Algorithm Deployment Process

- How AI algorithms are **deployed** on hardware for **actual computation**
  - AI Inference can be seen as forwarding data on a **computational graph**, where each node represents a single operator, edge represents dependency.
  - During deployment, operators are translated into hardware instructions. In the runtime, hardware executes instructions.

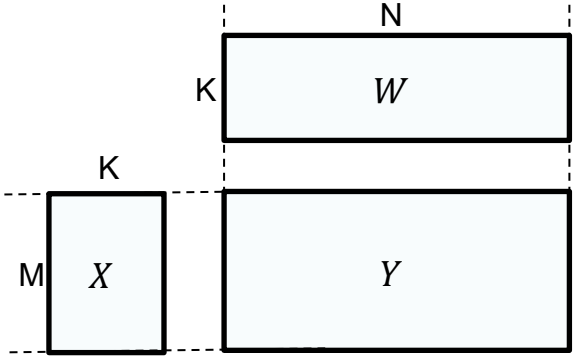


Transformer layer in LLMs

FC1

Taking FC1 as an example

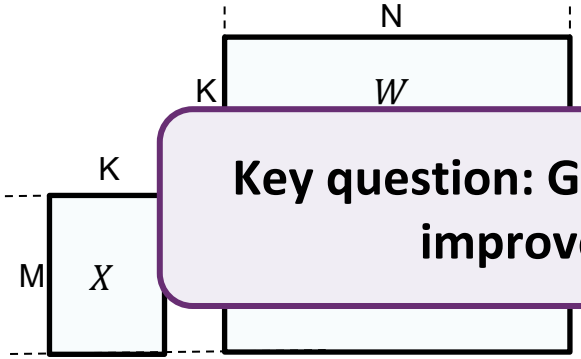
$$Y = W^T X$$



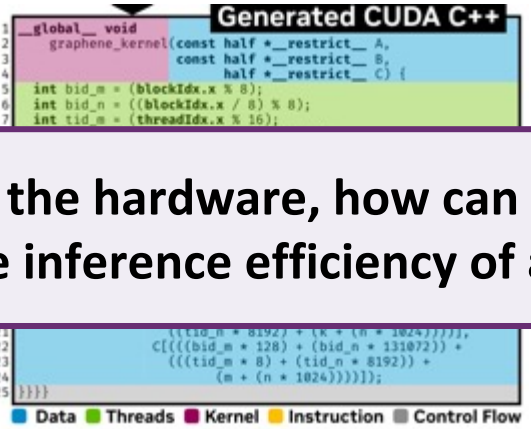
Abstracted as a single GEMM operator

Algorithm Deployment Process

- How AI algorithms are **deployed** on hardware for **actual computation**
  - AI Inference can be seen as forwarding data on a **computational graph**, where each node represents a single operator, edge represents dependency (**software level**).
  - During deployment, operators are translated into hardware instructions. In the runtime, hardware executes instructions.



Abstracted as a GEMM operator



Hardware Instructions (CUDA) Include **computation, memory, and control**.

The hardware executes instructions to perform computations.

Key question: Given the hardware, how can we evaluate and improve the inference efficiency of an LLM?

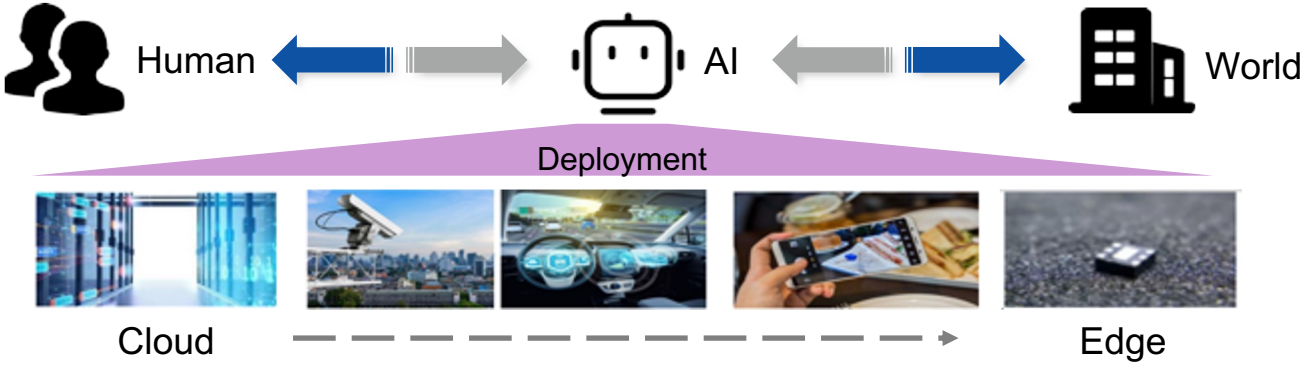
[1] Bastian Hagedorn, et al. “Graphene: An IR for Optimized Tensor Computations on GPUs.” ASPLOS 2023.

目录 Contents

Review: Application Challenge

The scale of the model size and input/output pose challenges for efficient inference across platforms and application scenarios.

Model size, input/output length ⇨ Inference cost (time, storage, energy)



Requirements or constraints of application scenarios and platforms

High Throughput	Low Latency	Small Storage	Low Energy Consumption

Review: Application Challenge

The scale of the model size and input/output pose challenges for efficient inference across platforms and application scenarios.

Model size, input/output length ⇨ Inference cost (time, storage, energy)

To meet requirements or constraint of application scenarios and platforms, we need to optimize the resource consumption of AI inference.

Cloud — — — — — — — — — — — — ▸ Edge

Requirements or constraints of application scenarios and platforms

High Throughput	Low Latency	Small Storage	Low Energy Consumption

Problem Definition: Objectives & Constraints

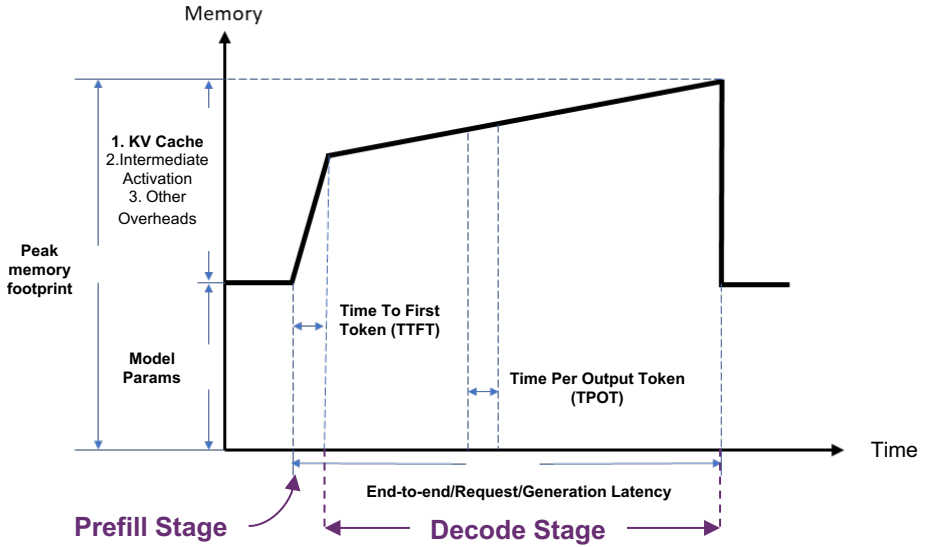
Optimization objective or constraint: Usually, **latency**, **memory**, **energy consumption** or **throughput** will be the ultimate objective or constraint on "efficiency". In the meantime, the intelligence level of AI needs to be retained.

Measured metrics

Measured by testing model on platform, **platform-related**

Directly correspond to objectives / constraints, related to final user experience, resource consumption, etc.

Time	Throughput	Latency
Memory	Memory footprint	
Energy	Energy	



*Note this is only a conceptual illustration. In actual serving framework, the KV cache pool is usually pre-allocated.

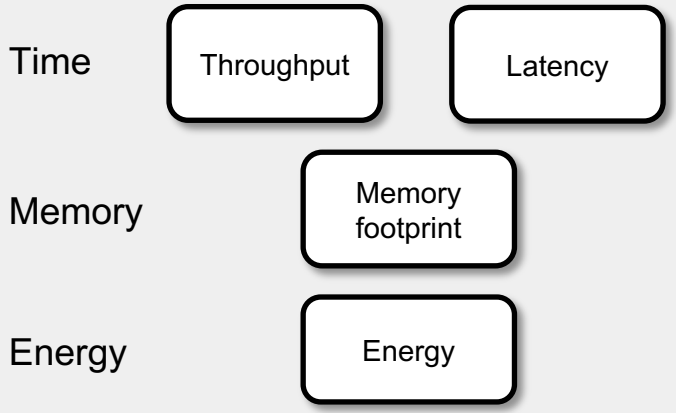
Problem Definition: Objectives & Constraints

Optimization objective or constraint: Usually, **latency**, **memory**, **energy consumption** or **throughput** will be the ultimate objective or constraint on "efficiency". In the meantime, the intelligence level of AI needs to be retained.

Measured metrics

Measured by testing model on platform, **platform-related**

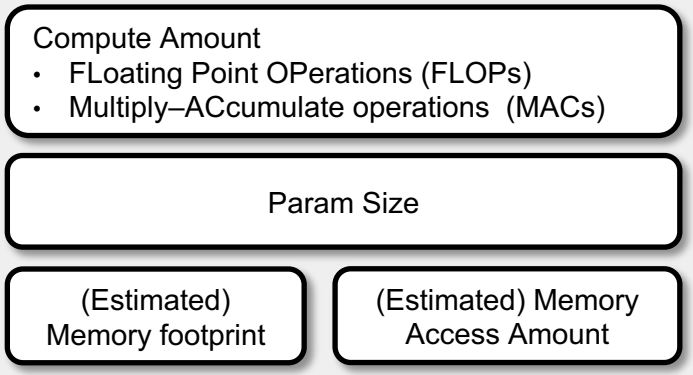
Directly correspond to objectives / constraints, related to final user experience, resource consumption, etc.



Proxy metrics

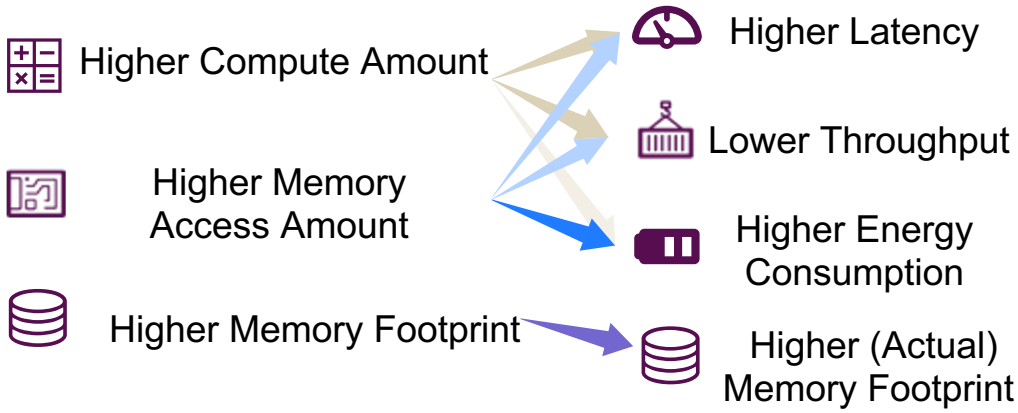
Estimated with only model specification, **platform-agnostic**

Widely used as the objective in some academic work or early stage of model-level optimization.
In practice, they are also useful in diagnosis of the bottleneck and quick estimation of the measured metrics.



How Proxy Metrics Relate with Measured Metrics

- We can use three important proxy metrics to analyze the efficiency
  - **Compute amount**: the amount of operations
  - **Memory access amount**: the amount of data that read or written between off-chip DRAM and GPU chip
  - **Memory footprint**: the occupied off-chip DRAM size to store parameters/KV cache/activation



Then, let's bring hardware specification into the picture.

How Proxy Metrics Relate with Measured Metrics

- Intuitively, $\frac{\text{Compute Amount}}{\text{Peak Compute Perf.}}$, $\frac{\text{Memory Access Amount}}{\text{Bandwidth}}$ are two lower bounds of latency.
- But the compute units might not be fully utilized, the bandwidth might not be fully utilized:

$$\text{Compute\_utilization} = \frac{\text{Achieved Compute Perf.}}{\text{Peak Compute Perf.}} \qquad \text{Bandwidth\_utilization} = \frac{\text{Achieved Bandwidth}}{\text{Peak Bandwidth}}$$

- Review our previous estimation example:

Llama-3.1-8B #Prefill tokens	Compute Amount	Memory Access	VRAM Consume	GPU	TTFT*
1K token	15.9 TFLOPs	29.2 GB	15.3 GB	1xA100	73~170 ms

Why do we use compute amount instead of memory access amount to estimate latency for the prefilling stage? => It's because prefiling stage is usually more **"computation-bounded"**

The compute amount of prefilling 1K token with Llama-3.1-8B is 15.9 TFLOPs.

NVIDIA A100 80G's FP16 peak compute performance is 312 TFLOPS.

$$\textbf{Latency} = \frac{\textit{Compute Amount}}{\textit{Peak Compute Perf.} \times \text{compute\_utilization}} = \frac{15.9\ \text{TFLOPs}}{312\ \text{TFLOPS} \times \text{compute\_utilization}} = \frac{51\ \text{ms}}{\text{compute\_utilization}}$$

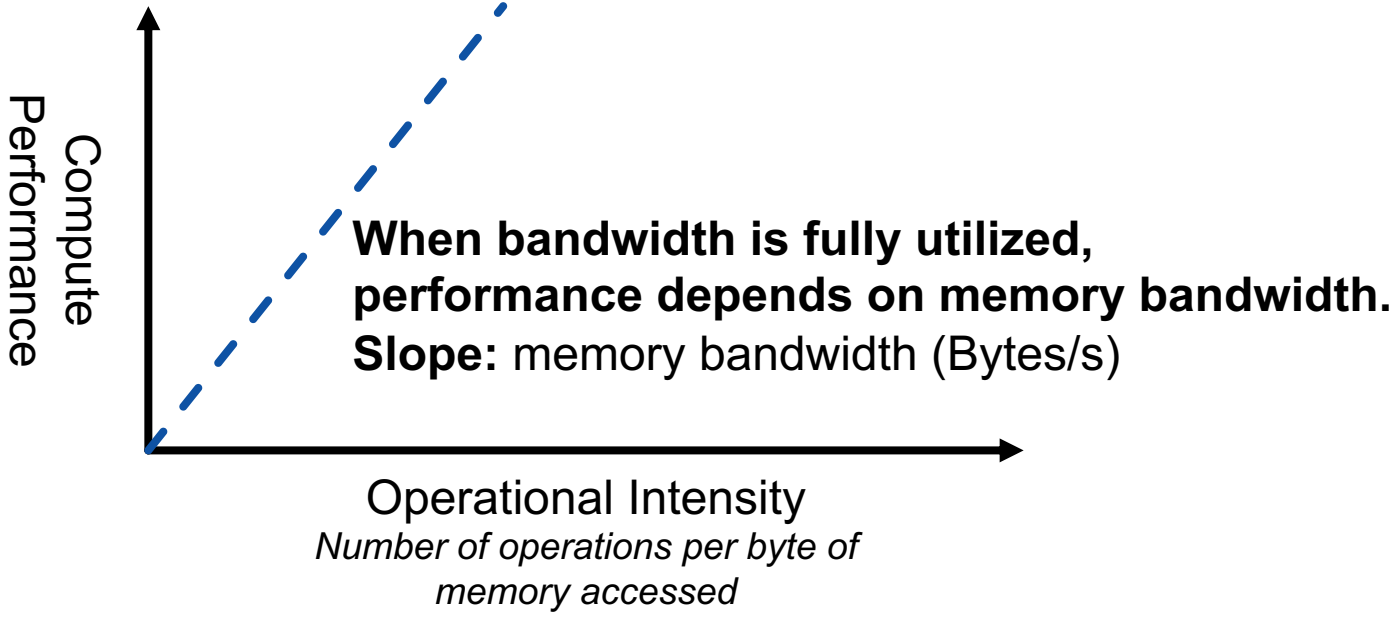
If we assume 30%~70% compute utilization, we can get $73\text{ms}\sim170\text{ms}$.

Performance Analysis Model: Roofline Model

- Proposed by David Patterson in 2009
- Purpose: Uses an algorithm's compute and memory access characteristics (operational intensity) along with the chip's peak performance and memory bandwidth to roughly assess computational bottlenecks and guide subsequent optimization directions.

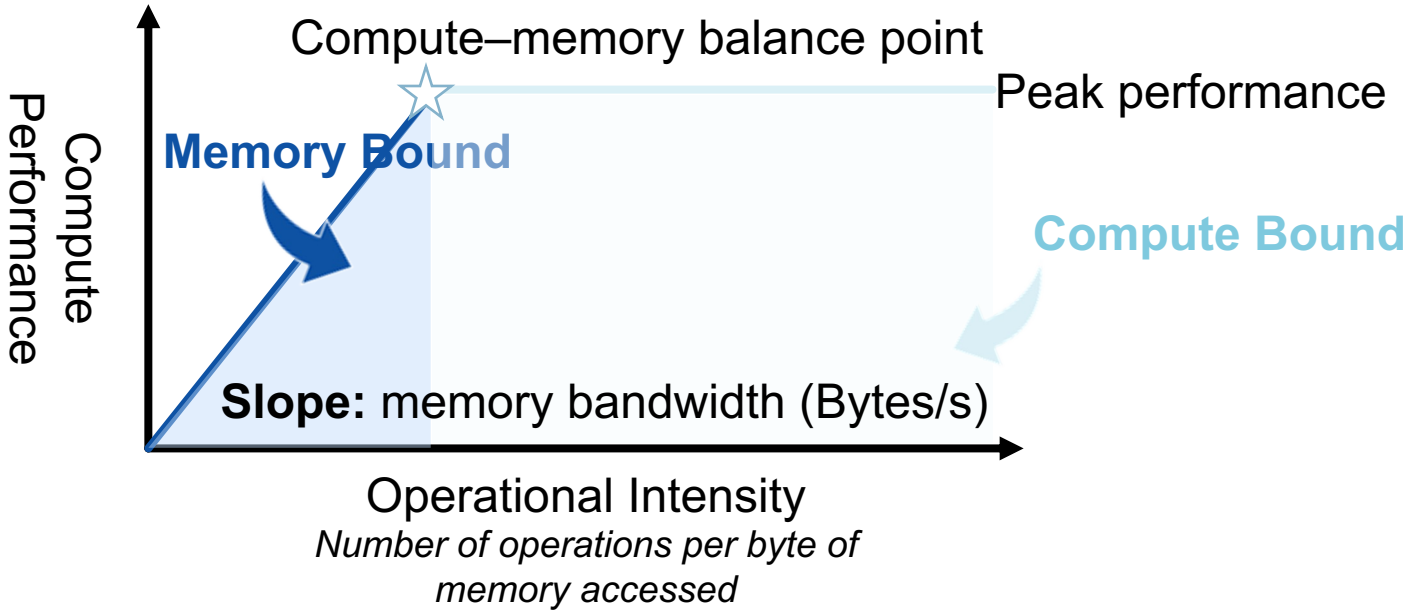
Performance Analysis Model: Roofline Model

- Proposed by David Patterson in 2009
- Purpose: Uses an algorithm's compute and memory access characteristics (operational intensity) along with the chip's peak performance and memory bandwidth to roughly assess computational bottlenecks and guide subsequent optimization directions.



Performance Analysis Model: Roofline Model

- Proposed by David Patterson in 2009
- Purpose: Uses an algorithm's compute and memory access characteristics (operational intensity) along with the chip's peak performance and memory bandwidth to roughly assess computational bottlenecks and guide subsequent optimization directions.

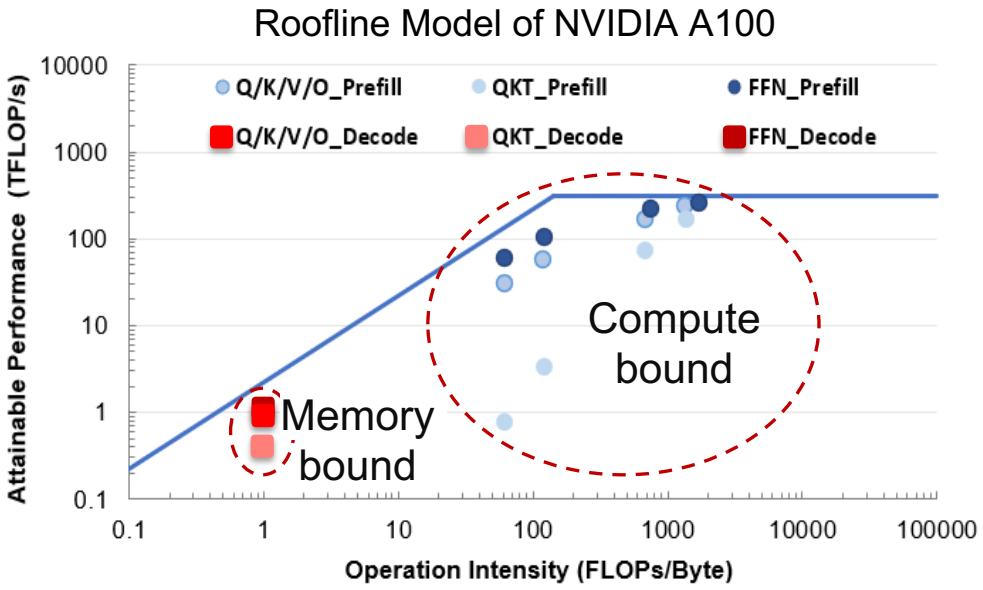


Performance Analysis Model: Roofline Model

- Take LLM as an example:
  - Prefill stage: **Compute bound**; Decode stage: **memory bound**

	Prefill	Decode
$Q/K/V/O$	$\frac{ld}{l+d}$	$\frac{d}{d+1}$
QK^T	$\frac{l}{1} \gg 1$	$\frac{l}{l+1} \sim 1$
$AttenV$	$\frac{l}{1}$	$\frac{l}{l+1}$
FFN	$\frac{ld_{FFN}}{l+d_{FFN}}$	$\frac{d_{FFN}}{d_{FFN}+1}$

Operational intensity at different stages (FLOPs/Byte)



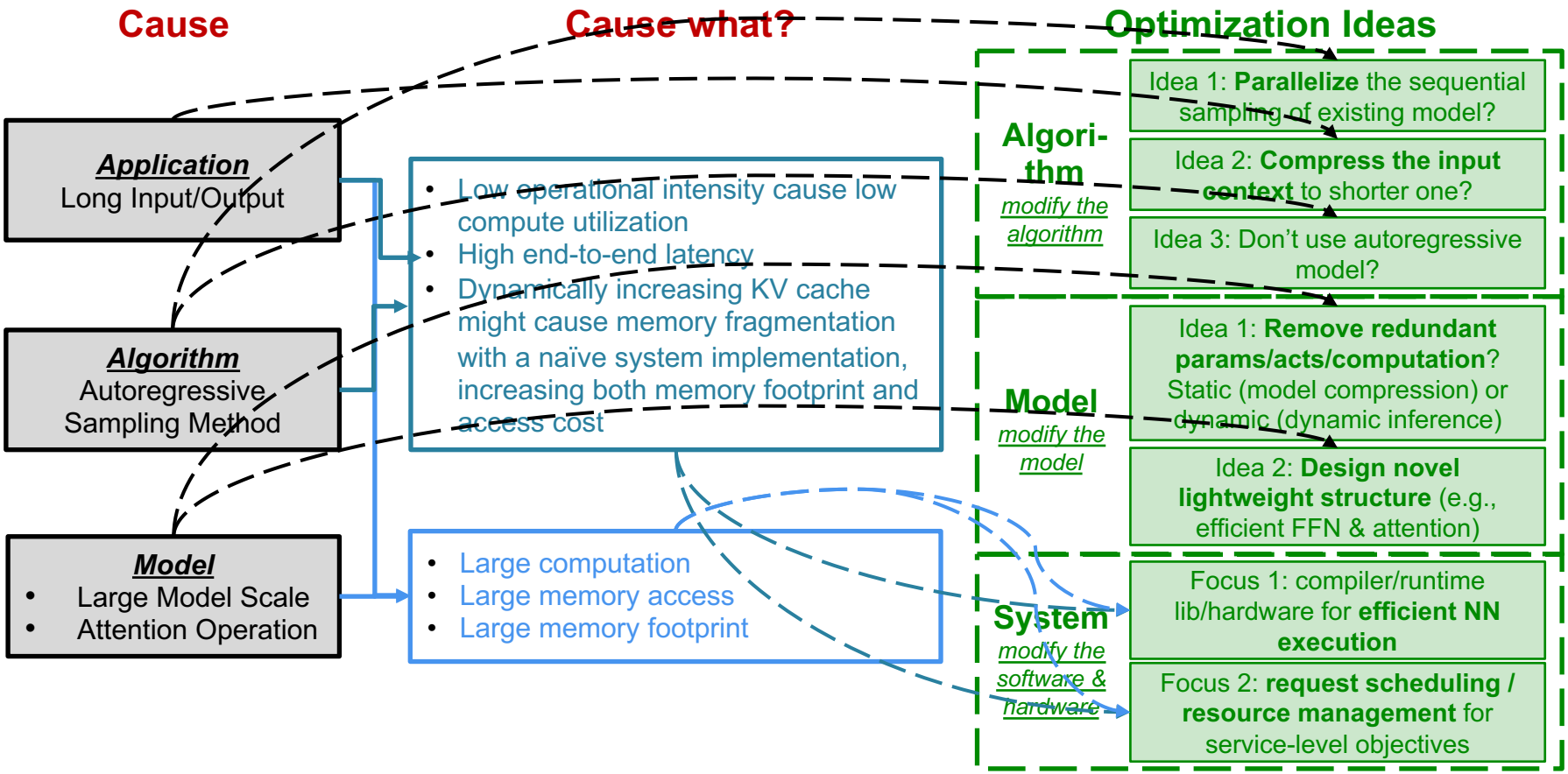
What is the Optimization Space

For designing the optimization space, we need to know the **"bottleneck" modules or properties** of the current algorithm, model, and software that hinders the efficiency of running them on the given hardware.

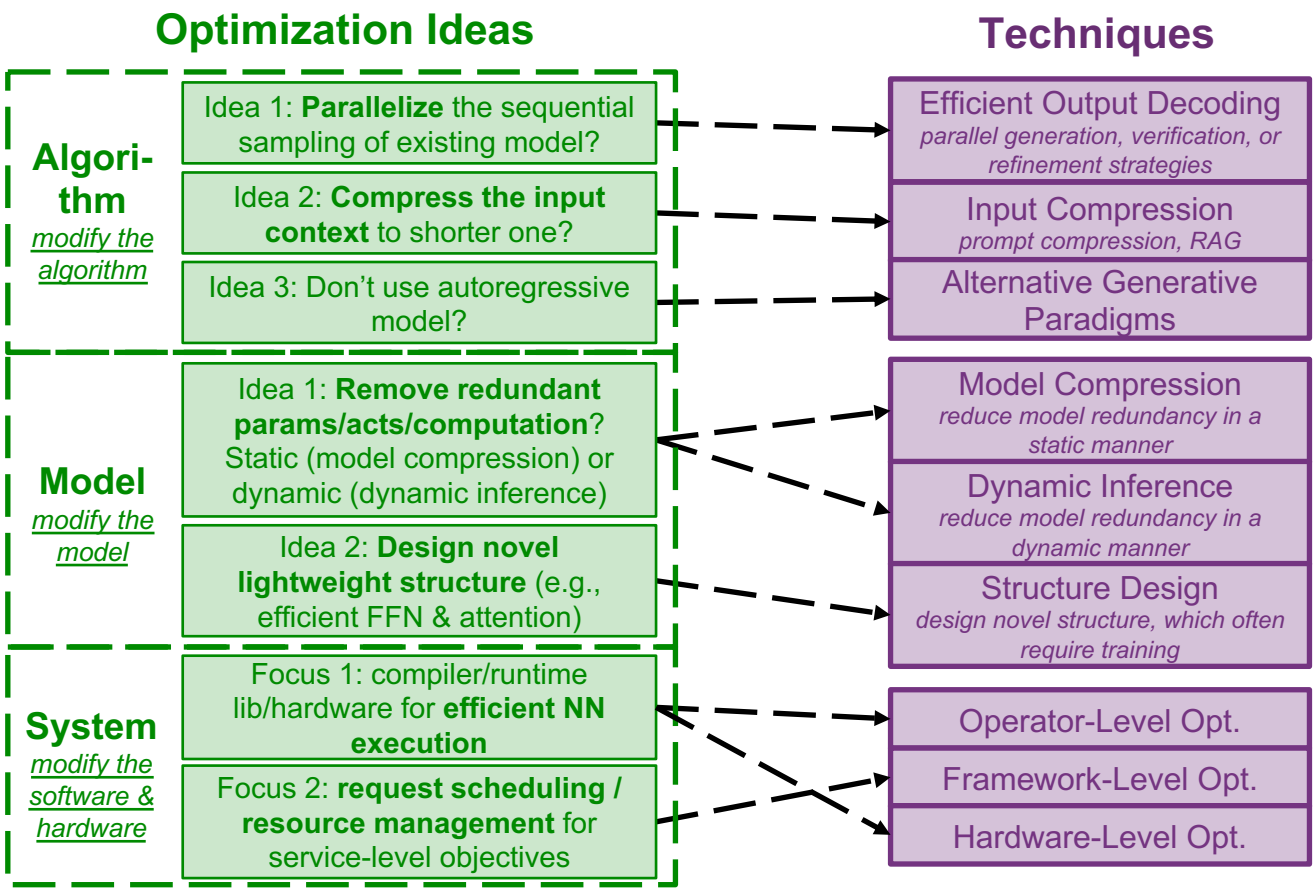
Root causes of LLM inference inefficiency

- **Application:** The input / output token length can be very long.
- **Algorithm**: Autoregressive model samples tokens one by one.
- **Model**: (i) The transformer model has a large number of weights and computations. (ii) Attention modules have quadratic complexity w.r.t. the input token length.

What is the Optimization Space



What is the Optimization Space



Contents

Practical Pipeline of Model/System-Level Method Design

Estimation According to Specifications include Application & Model & Hardware

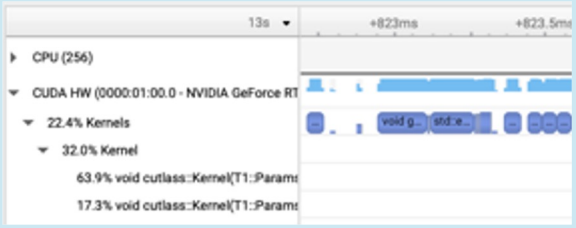
Application Specification
- Objectives, e.g., memory footprint, latency, throughput
- Context lengths
- Batch size
- …

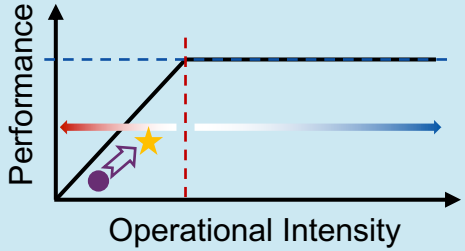
Model Specification
- Num hidden layers
- Num key-value heads
- Hidden size
- Intermediate size
- Num attention heads
- Head dim

Hardware Specification
- Peak Compute Performance
- Memory Capacity
- Memory Bandwidth

Can estimate theoretically:
- Judge compute or memory bound of each module by roofline model
- Estimate bottleneck module
- Estimate overall objectives

Profile/Diagnosis Tool: Nsight System & Nsight Compute





Model-Level Design

- Analyze compressing which dimension / how to redesigning the module might help with the efficiency most
- Analyze algorithmic redundancy & property, how can we retain/restore performance

System-Level Design (Operator-Level)

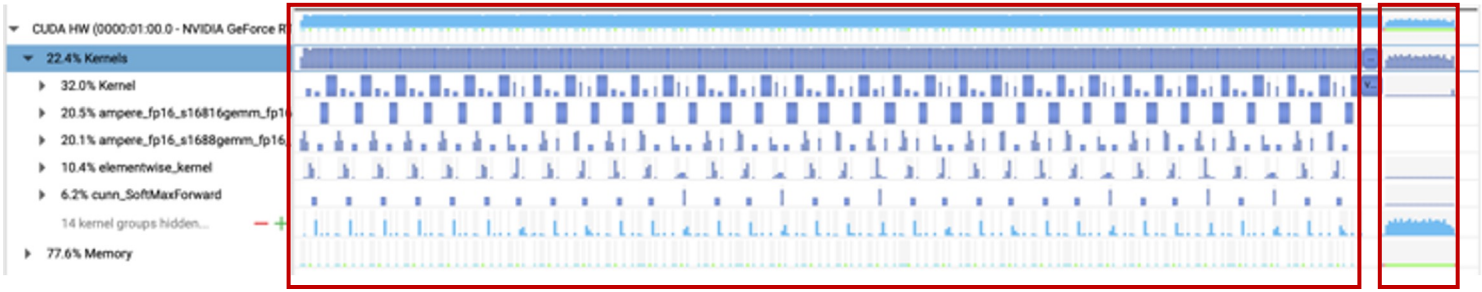
- Operation fusion to reduce memory access and kernel launch overhead
- Reimplement some operations

GPU Software Stack

- **NVIDIA GPU Diagnosis Tool**
  - **Nsight System**
    - A system-level analysis tool used for diagnosing performance bottlenecks from a global perspective and identifying key operators that require optimization.

      nsys nvprof -o {output-file} python3 xxx.py

    - Taking **LLaMA2-7B** as an example, Nsight Systems can visualize all operators and their calls across all inference stages. It enables researchers to quickly pinpoint efficiency bottlenecks from a global perspective.
      - Input length: **2048 tokens**; decoding: **2 tokens**



GPU Software Stack

- **NVIDIA GPU Diagnosis Tool**
  - **Nsight System**
    - A system-level analysis tool used for diagnosing performance bottlenecks from a global perspective and identifying key operators that require optimization.

      nsys nvprof -o {output-file} python3 xxx.py

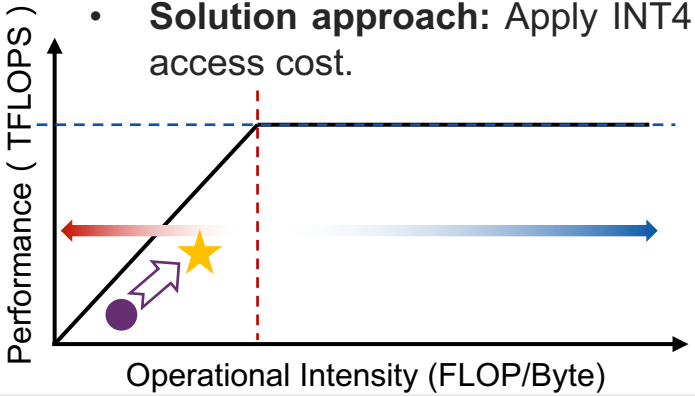
    - Taking **LLaMA2-7B** as an example, Nsight Systems can visualize all operators and their calls across all inference stages. It enables researchers to quickly pinpoint efficiency bottlenecks from a global perspective.
      - Nsight Systems analysis reveals that, during the decode stage, the General Matrix-Vector Multiplication (GEMV) operator in the linear layer is the primary performance bottleneck.

GPU Software Stack

- **NVIDIA GPU Diagnosis Tool**
  - **Nsight Compute**
    - A kernel-level performance analysis tool focused on deep optimization of individual CUDA kernels, used in conjunction with the Roofline Model for performance analysis.

      ncu –set full -o {output-file} python3 xxx.py

    - Taking **LLaMA2-7B** as an example, using Nsight Compute to analyze the GEMV operator in the linear layer shows that the operator is severely **memory-bound**.
      - **Solution approach:** Apply INT4 quantization to the weights to reduce weight's memory access cost.



Input channel	Output channel	FP16 (us)	INT4 (us)
4096	11008	159.3	52.0
11008	4096	45.6	37.6
4096	4096	43.5	23.0

RTX 3090 GPU

GPU Software Stack

- **NVIDIA GPU Diagnosis Tool**
  - **Nsight Compute**
    - Taking **LLaMA2-7B** as an example
      - **Implementation:** To achieve acceleration, the dequantization operator and the GEMV computation operator need to be fused.

```
#pragma unroll
for (int ic_0 = 0; ic_0 < 4; ic_0++){
  // iterate over different uint32_t packed_weights in this loop
  uint32_t current_packed_weight = packed_weights[ic_0];
  half packed_inputs[PACK_FACTOR];
  // each thread load 8 inputs, starting index is packed_group_idx * 128 * 8 (because each iter loads 128*8)
  if (inputs_ptr_delta + ic_0 < IC / PACK_FACTOR) {
    *((float4*)packed_inputs) = *(inputs_ptr + ic_0);
    #pragma unroll
    for (int ic_1 = 0; ic_1 < PACK_FACTOR; ic_1++){
      // iterate over 8 numbers packed within each uint32_t number
      float current_single_weight_fp = (float)(current_packed_weight & 0xF);
      float dequantized_weight = scaling_factor * (current_single_weight_fp - current_zeros);
      //if(blockIdx_x == 0 && blockIdx_y == 0 && threadIdx_x == 0 && threadIdx_y == 0 && ic_0 == 0 && ic_1 == 0 && pi
      psum += dequantized_weight * __half2float(packed_inputs[ic_1]);
      current_packed_weight = current_packed_weight >> 4;
    }
  }
}
```

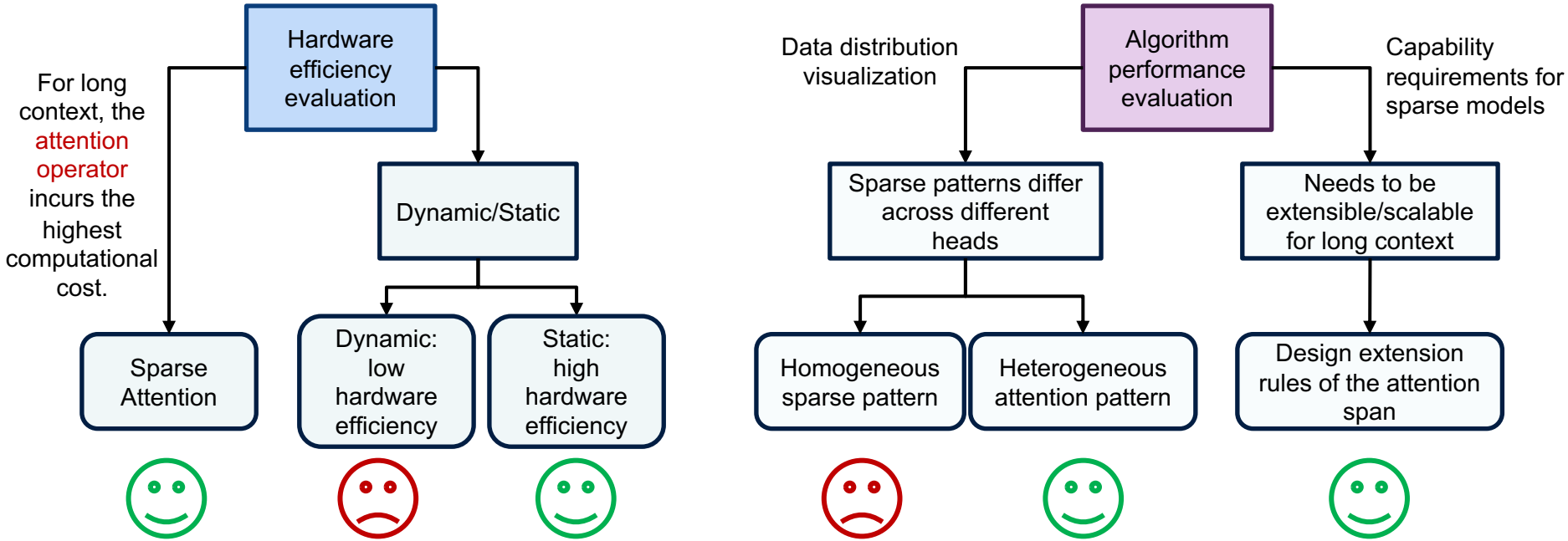
Dequantization

GEMV

Operator Fusion

Example: Design Thought of Model Compression Method

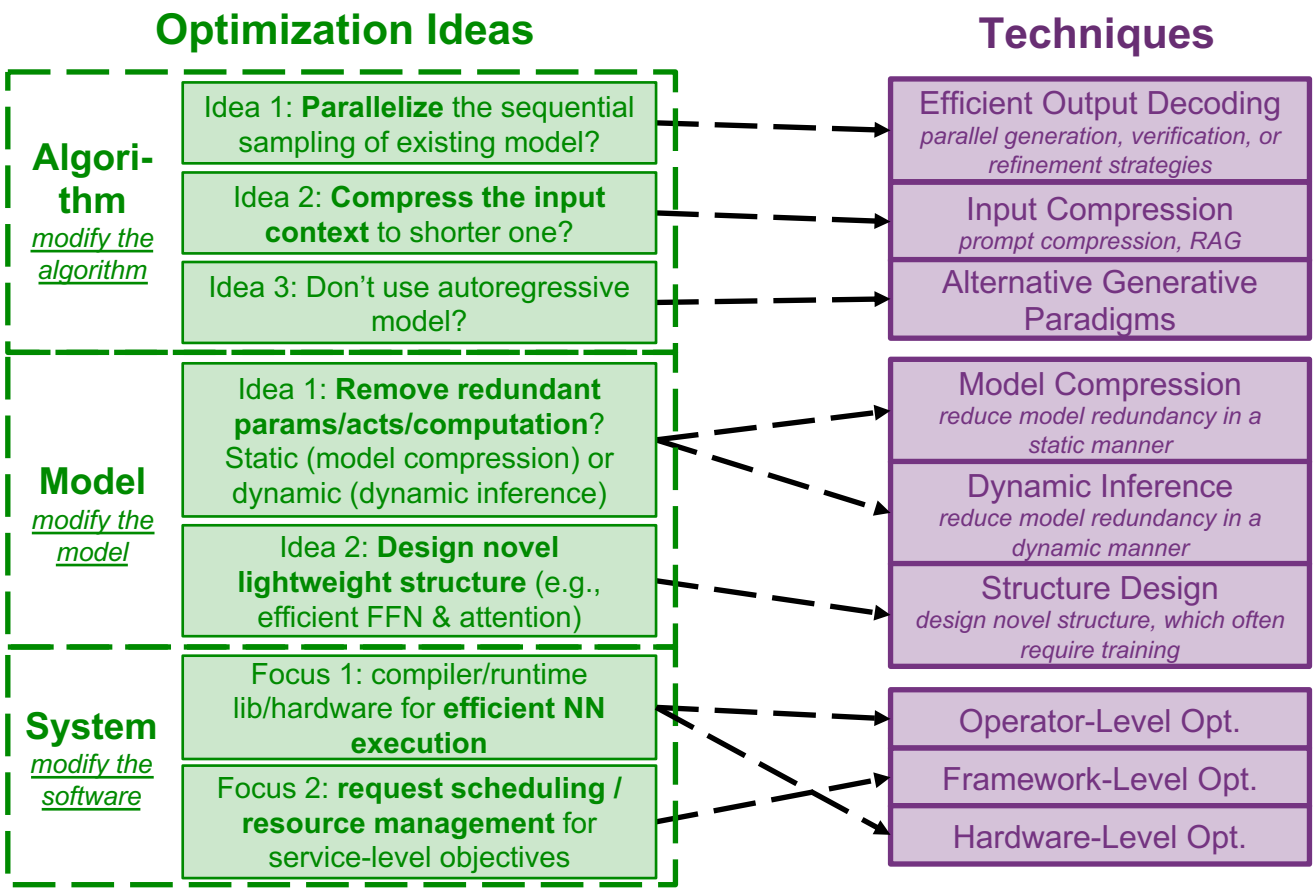
- How to design a sparsification method for a given model and scenario
  - Example: MoA[1]



[1] Fu, Tianyu*, Huang, Haofeng*, Ning, Xuefei*, et al. “MoA: Mixture of Sparse Attention for Automatic Large Language Model Compression.” CoLM 2025.

Contents

What is the Optimization Space

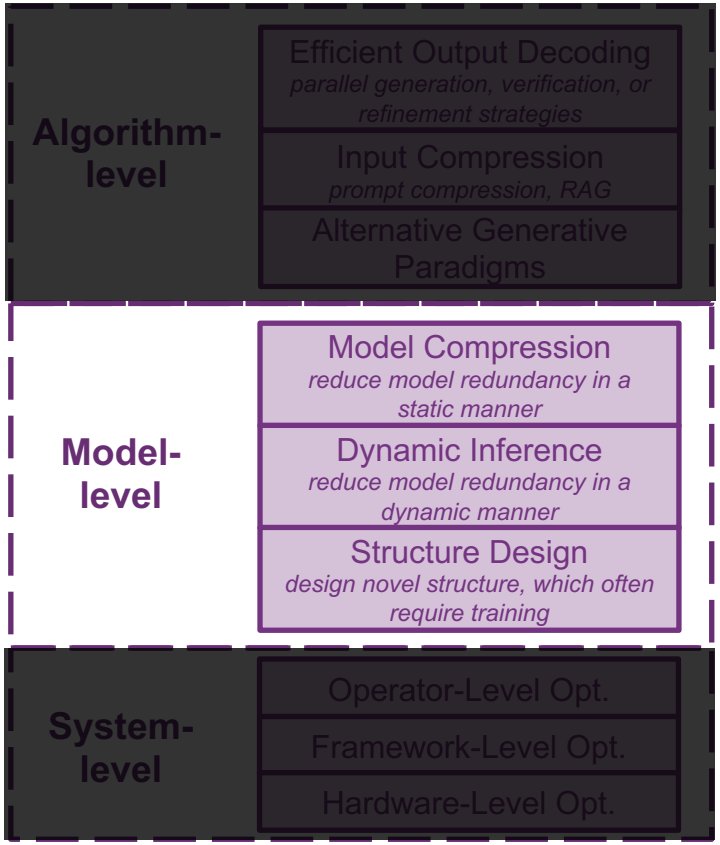


There is no representation/space or optimization formalization general to many work. The design of these methods directly change a core application or algorithm property to improve efficiency.

- Model Structure (e.g., #layer, #channel)
- Value Representation (e.g., low-bit representation)

- Computational Graph (e.g., fusion)
- Kernel Implementation
- Request scheduling, resource management, model placement
- Framework Implementation
- Hardware Implementation

Menu of Techniques



Model Compression

- Quantization
- Sparse Attention
- Weight Pruning
- Sharing
- Knowledge Distillation

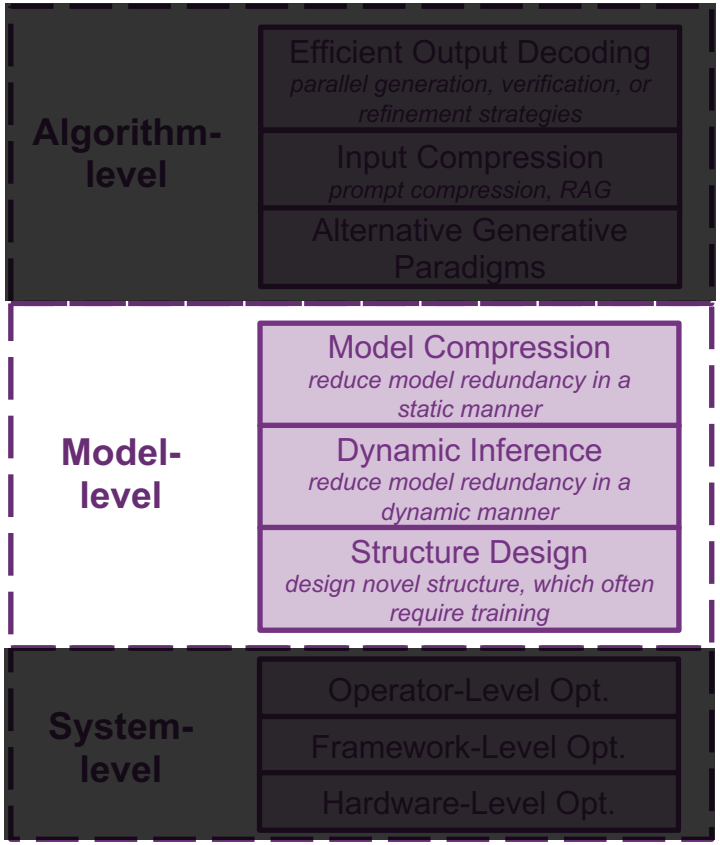
Dynamic Inference

- Module-granularity
- Model-granularity

Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

Menu of Techniques



Model Compression

- **Quantization**
- Sparse Attention
- Weight Pruning
- Sharing
- Knowledge Distillation

Dynamic Inference

- Module-granularity
- Model-granularity

Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

Definition

- **Motivation**

Neural network weights and activations are typically represented using high-precision floating-point formats. However, there exists numerical redundancy in neural network computations, and using lower-precision arithmetic does not significantly affect the accuracy of the network [1].

- **Definition**

Quantization: Represent weights & activations with low-bit numbers, thereby storing them or computing them with reduced numerical precision.

IEEE 754 standard 32-bit floating-point data 8-bit fixed-point data

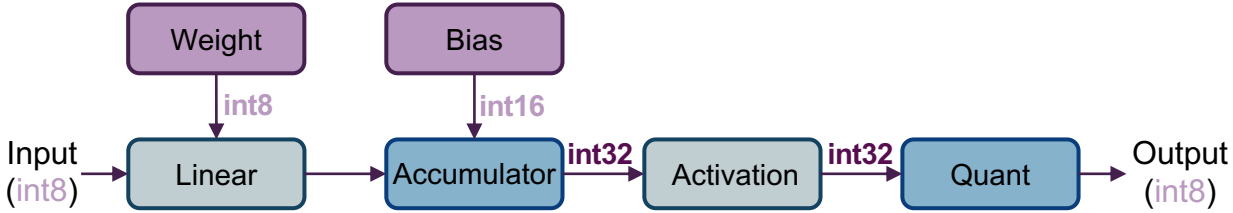
* 图片来源: MIT EfficientML Course

[1] Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, Han et al., ICLR 2016.

Two Procedures

- **Offline stage: Model quantization procedure**
  - Convert the FP weights into low-bit-width weights
  - Determine the quantization parameters for activations (optional, if using low-precision activation and not using online quant) Quantize floating-point weights (FP32/FP16) into low-bit weights (INT8/INT4).

- **Online stage: Quantized inference procedure**
  - Low-Precision computation: LP arithmetic -> Requantization



  - High-Precision computation: Dequant -> HP arithmetic -> (optional) Quant

[1] https://nics-effalg.com/assets/ppt/2023-08-30-QLLMIntro.pdf

Model Quantization Procedure

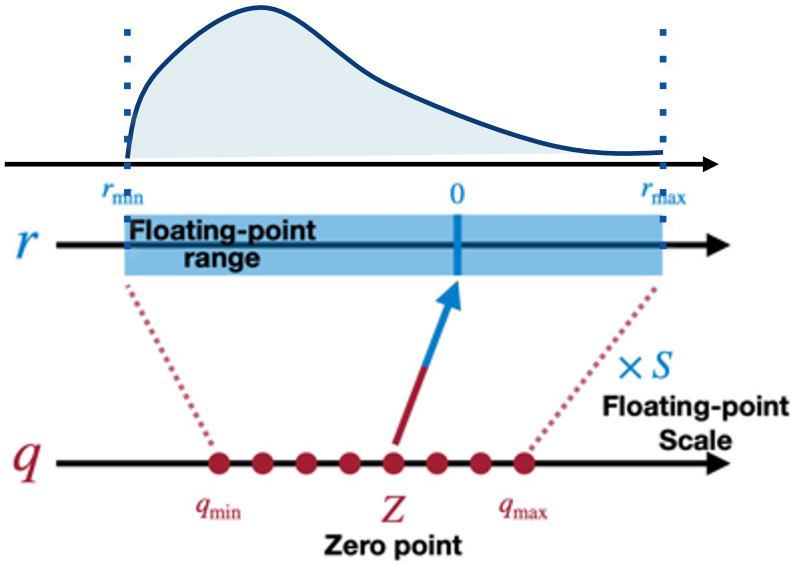
- **Example: Uniform Fixed-Point Quantization**
- **A quantization method needs to answer: "How to convert floating-point numbers into fixed-point number representation? "**

$$q = (\lfloor r/S \rceil - Z)$$

Fixed-point **Floating-point** **Rounding function**

- Quantization parameters: **scaling factor (S)**; **zero point (Z)**
- **How to decide quantization parameters?** Take asymmetric quantization ($Z \neq 0$) as an example, if we want the float-point range to cover $[r_{min}, r_{max}]$:

$$S = \frac{r_{max} - r_{min}}{2^{N-1}} \qquad Z = \frac{r_{max} + r_{min}}{2 * 2^{N-1}}$$

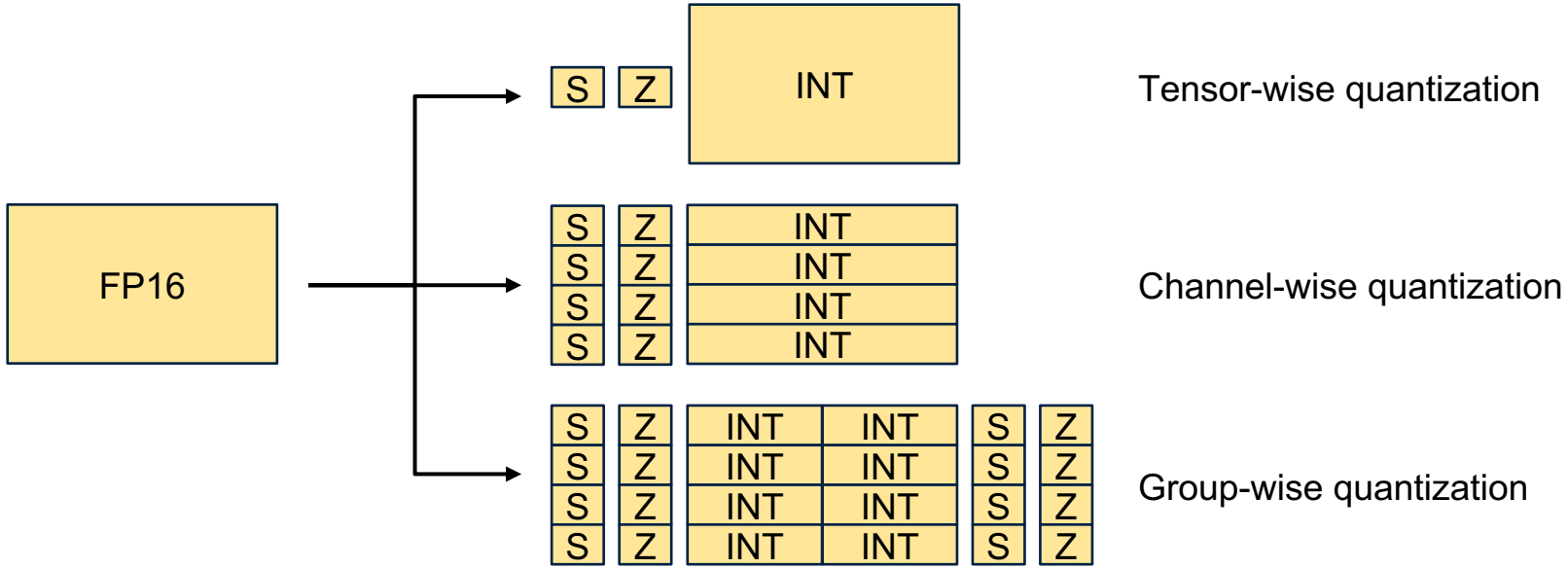


* 图片来源: MIT EfficientML Course

[1] Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, Han et al., ICLR 2016.

Quantization

- **Quantization Granularity**
- **For actual storage decrease and speed-up, a group of values needs to share the same quantization parameters (the group size is called quantization granularity, e.g., tensor-wisely, channel-wisely).**

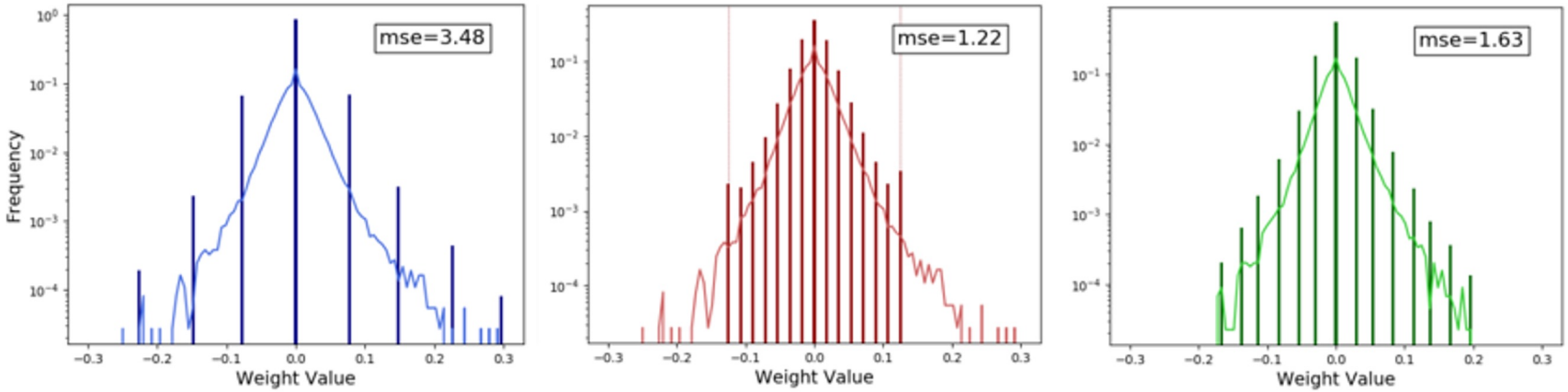


* 图片来源: MIT EfficientML Course

[1] Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, Han et al., ICLR 2016.

Quantization

- **Core focus of quantization parameter decision:** Appropriately balance representational range and precision (i.e., balancing truncation error and rounding error).



Large rounding error
(large scale factor)

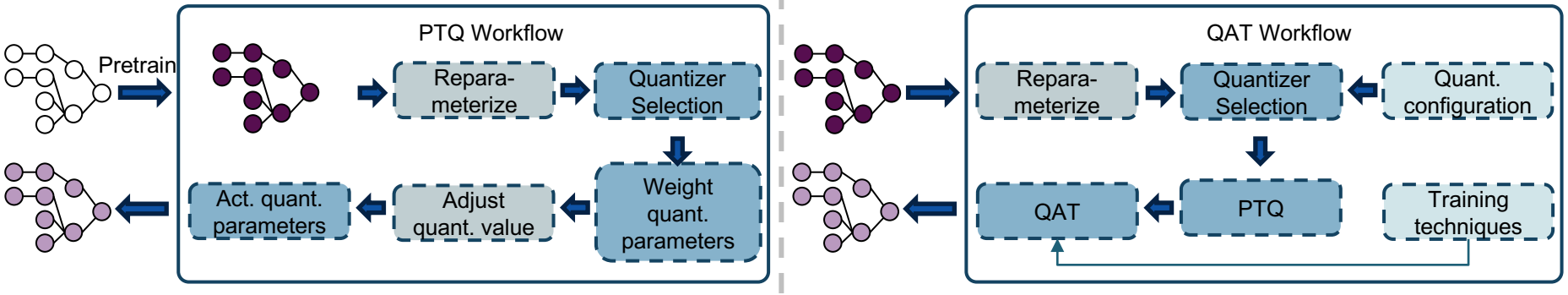
Large truncation error
(small scale factor)

Well-balanced
(appropriate scale factor)

[1] Zhao, Ritchie, et al. "Improving neural network quantization without retraining using outlier channel splitting." International conference on machine learning. PMLR, 2019.

Quantization

- **Two types of quantization process**
  - **PTQ v.s. QAT**



Pros	Cons	Pros	Cons
No weight optimization, No training cost, Fast quantization process	Lack of recovery, Difficult to quantize models to ultra-low bitwidth	Quantization with performance recovery, Enabling lower bitwidth	High computational cost and data demand

Quantization

- **Representative studies**

	Quantized Tensor Type			Quantized Format	Quantized Criterion	Quantized Value Update
	Weight	Activation	KV Cache			
GPTQ	√			Uniform	Statistic-based	√
AWQ	√			Uniform	Search-based	√
SqueezeLLM	√			Non-uniform	Statistic-based	
GPT3.int8()	√	√		Uniform	Statistic-based	
SmoothQuant	√	√		Uniform	Statistic-based	√
RPTQ	√	√		Uniform	Statistic-based	
OminiQuant	√	√		Uniform	Search-based	
FlexGen	√		√	Uniform	Statistic-based	
Atom	√	√	√	Uniform	Statistic-based	
KVQuant			√	Non-uniform	Statistic-based	
KIVI			√	Uniform	Statistic-based	

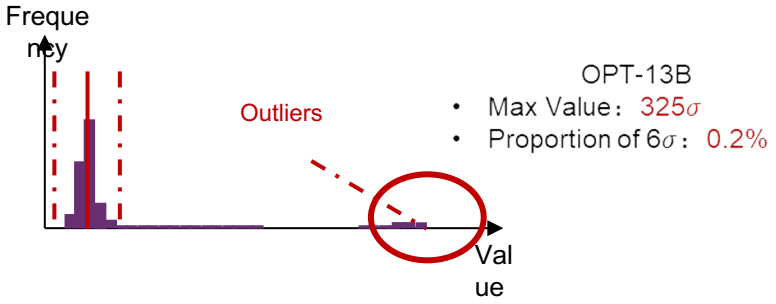
[1] Frantar, Elias, et al. "Gptq: Accurate post-training quantization for generative pre-trained transformers." arXiv preprint arXiv:2210.17323 (2022).
[2] Lin, Ji, et al. "Awq: Activation-aware weight quantization for on-device llm compression and acceleration." Proceedings of machine learning and systems 6 (2024): 87-100.
[3] Kim, Sehoon, et al. "Squeezellm: Dense-and-sparse quantization." arXiv preprint arXiv:2306.07629 (2023).
[4] Dettmers, Tim, et al. "Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale." Advances in neural information processing systems 35 (2022): 30318-30332.
[5] Xiao, Guangxuan, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." International conference on machine learning. PMLR, 2023.
[6] Yuan, Zhihang, et al. "Rptq: Reorder-based post-training quantization for large language models." arXiv preprint arXiv:2304.01089 (2023).
[7] Shao, Wenqi, et al. "OmniQuant: Omnidirectionally Calibrated Quantization for Large Language Models." ICLR. 2024.
[8] Sheng, Ying, et al. "Flexgen: High-throughput generative inference of large language models with a single gpu." International Conference on Machine Learning. PMLR, 2023.
[9] Zhao, Yilong, et al. "Atom: Low-bit quantization for efficient and accurate llm serving." Proceedings of Machine Learning and Systems 6 (2024): 196-209.
[10] Hooper, Coleman, et al. "Kvquant: Towards 10 million context length llm inference with kv cache quantization." Advances in Neural Information Processing Systems 37 (2024): 1270-1303.
[11] Liu, Zirui, et al. "KIVI: a tuning-free asymmetric 2bit quantization for KV cache." Proceedings of the 41st International Conference on Machine Learning. 2024.

SmoothQuant

- **Typical PTQ Method: SmoothQuant[1]**

Motivation

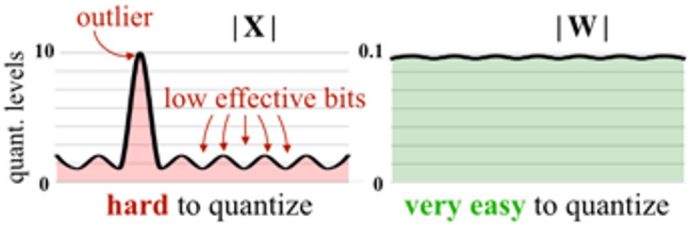
- In large models, activations often contain extremely large outliers, leading to significant quantization errors during W8A8 quantization.
- This study focuses on analyzing the data distribution patterns of the model and exploring methods to reduce outliers, aiming to achieve nearly lossless W8A8 quantization.



Insight

Analysis of weight and activation data distribution

- Outliers in large-model activations appear in specific channels. This property can be leveraged to balance the data distribution of weights and activations.



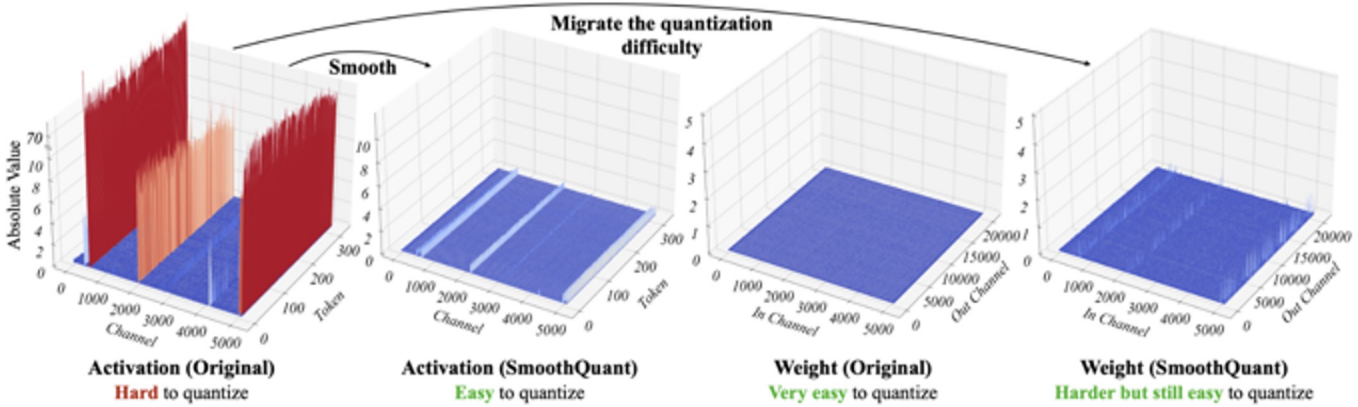
[1] Xiao, Guangxuan, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." ICML, 2023.

SmoothQuant

Method

- Introducing channel equalization to suppress outliers in activations.
  - Activation channels with large outliers are divided by an equalization factor s_j to reduce their magnitude.
    The corresponding weight channels are multiplied by the same factor s_j to appropriately increase the weight values.

$$\mathbf{Y} = (\mathbf{X}\mathrm{diag}(\mathbf{s})^{-1}) \cdot (\mathrm{diag}(\mathbf{s})\mathbf{W}) = \hat{\mathbf{X}}\hat{\mathbf{W}} \qquad \mathbf{s}_j = \max(|\mathbf{X}_j|)^{\alpha} / \max(|\mathbf{W}_j|)^{1-\alpha}$$



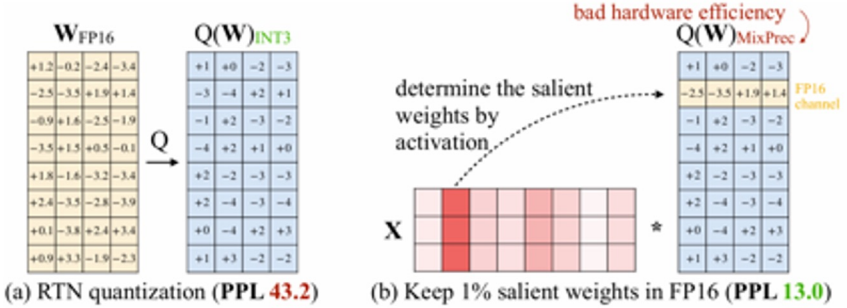
[1] Xiao, Guangxuan, et al. "Smoothquant: Accurate and efficient post-training quantization for large language models." ICML, 2023.

AWQ

- **Typical PTQ Method: AWQ[1]**

Motivation

- Not all weights in an LLM are equally important.
- Protecting only 1% salient weights can greatly reduce quantization error.



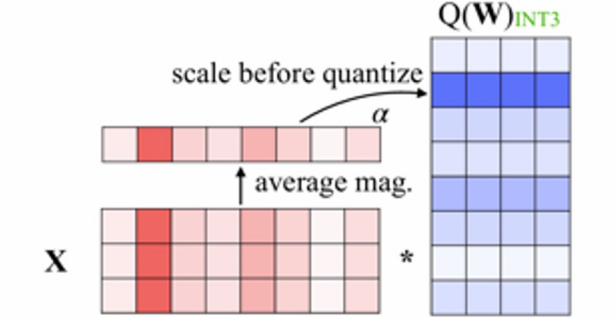
(a) RTN quantization (**PPL 43.2**) (b) Keep 1% salient weights in FP16 (**PPL 13.0**)

Methods

To consider bothsalient and non salient weights, AWQ searches for an optimal scaling factor that minimizes the reconstruction error for a certain layer.

$$\mathbf{s}^* = \arg\min_{\mathbf{s}} \mathcal{L}(\mathbf{s})$$

$$\mathcal{L}(\mathbf{s}) = \|Q(\mathbf{W} \cdot \mathrm{diag}(\mathbf{s}))(\mathrm{diag}(\mathbf{s})^{-1} \cdot \mathbf{X}) - \mathbf{W}\mathbf{X}\|$$

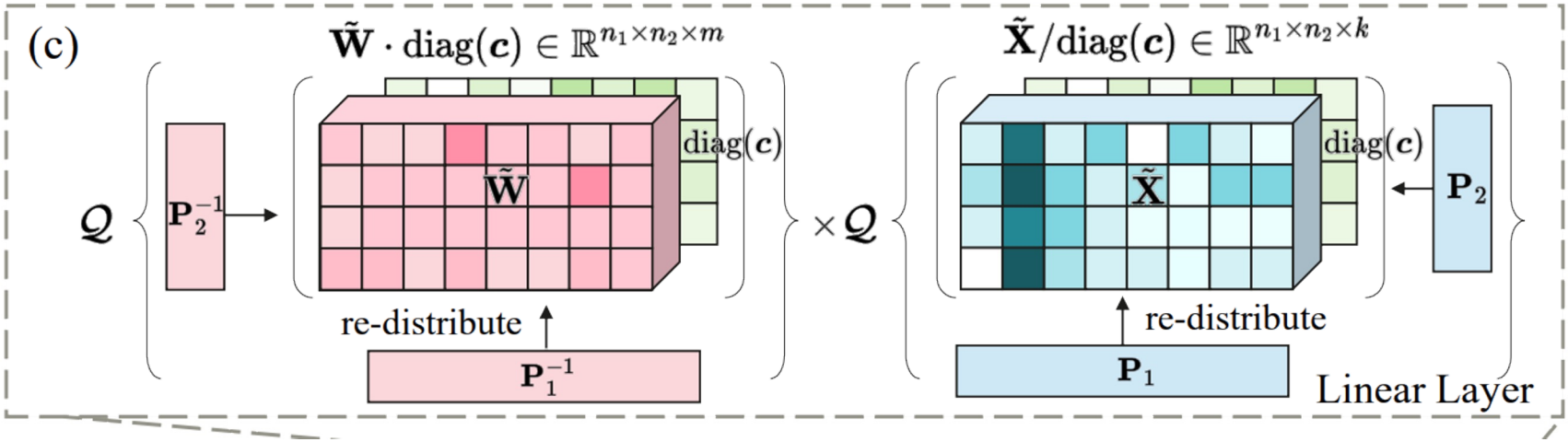


[1] Lin, Ji, et al. "Awq: Activation-aware weight quantization for on-device llm compression and acceleration." Proceedings of machine learning and systems 6 (2024): 87-100.

FlatQuant: Flatness Matters for LLM Quantization

- **Motivation**: Affine transformations are more powerful to suppress outliers
- **Methodology**: Learning affine transformations for each linear layer
  - **Reducing transformation overhead**: Kronecker product & kernel fusion

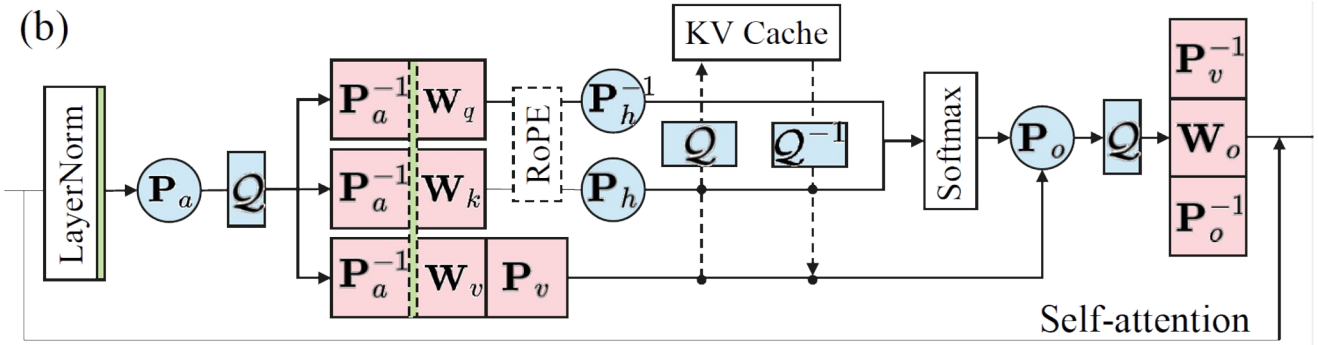
$$\mathbf{P}^* = \arg\min_{\mathbf{P}} \|\mathbf{Y} - \mathcal{Q}(\mathbf{X}\mathbf{P})\mathcal{Q}(\mathbf{P}^{-1}\mathbf{W}^\top)\|_F^2, \qquad \mathbf{P} = \mathbf{P}_1 \otimes \mathbf{P}_2,$$



[1] Sun, Y., Liu, R., Bai, H., et al. "FlatQuant: Flatness Matters for LLM Quantization". *ICML 2025*.

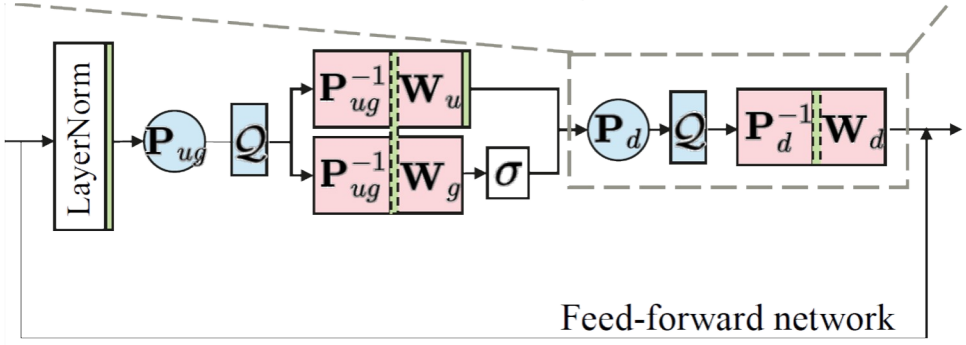
FlatQuant: Flatness Matters for LLM Quantization

How to Integrate FlatQuant with the Transformer architecture?



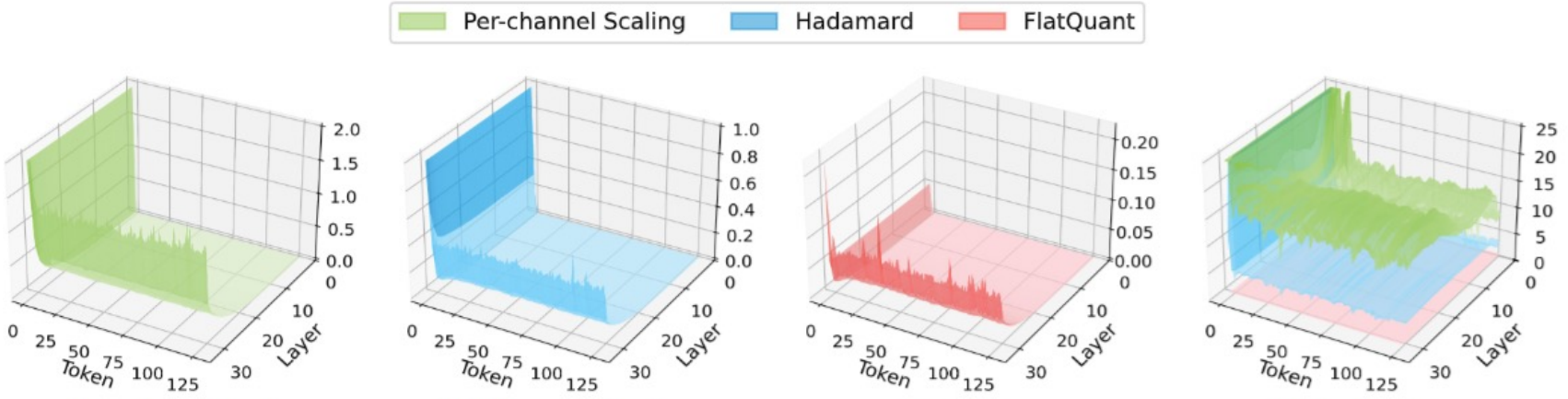
Integration with Self-attention

Integration with feed-forward network



FlatQuant: Flatness Matters for LLM Quantization

The mean square error of quantization along channels & tokens can be effectively reduced



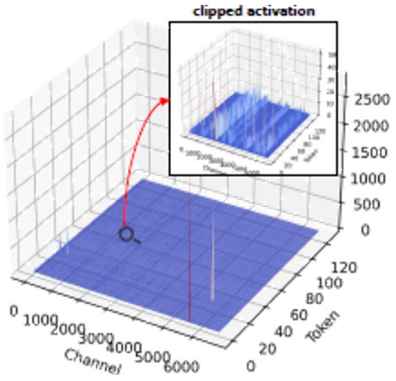
(a) Per-channel Scaling. (b) Hadamard Transfrom. (c) FLATQUANT. (d) Stacked View.

Paper

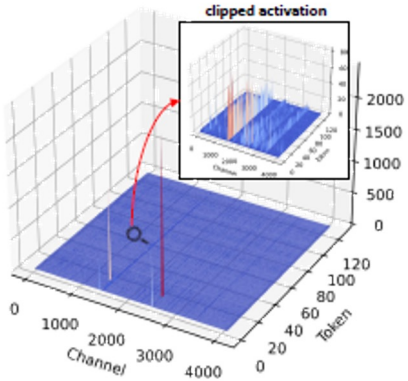
FlatQuant @ Github

IntactKV: Keeping Pivot Tokens Intact

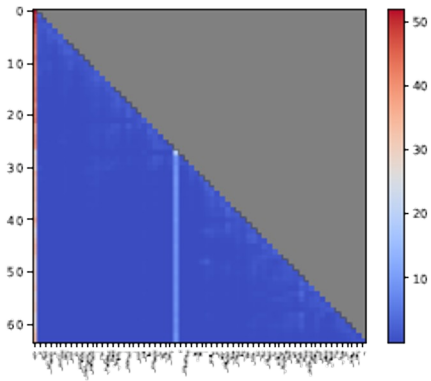
Observation: the pivot tokens exhibit attention sinks together with massive activation outliers.



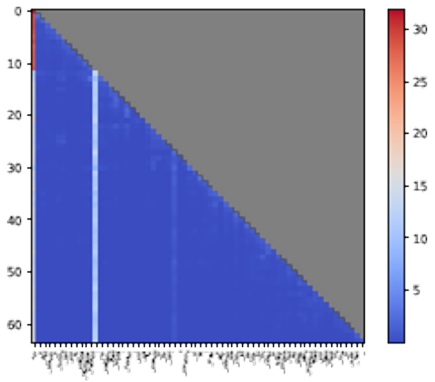
(a) Output activations of LLaMA-30B Layer 24



(b) Output activations of LLaMA-2-7B Layer 24



(c) Attention map of LLaMA-30B Layer 24



(d) Attention map of LLaMA-2-7B Layer 24

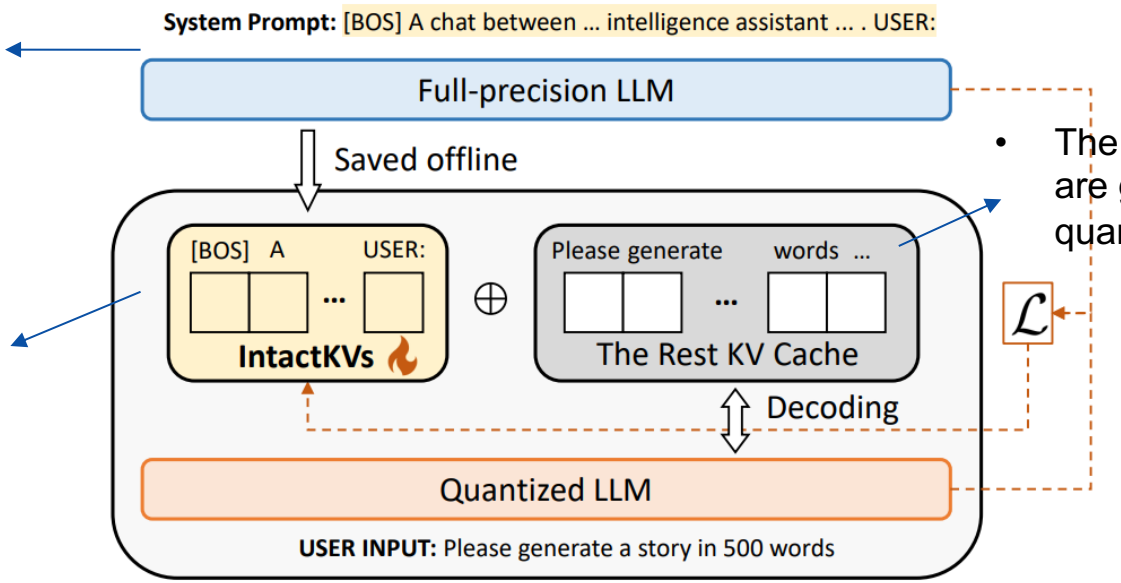
[1] Ruikang Liu, **Haoli Bai†**, et.al. IntactKV: Improving Large Language Model Quantization by Keeping Pivot Tokens Intact. Findings of ACL, 2024.

IntactKV: Keeping Pivot Tokens Intact

Avoid the quantization error accumulated on pivot tokens that are critical to the performance.

- The system prompt contains most pivot tokens with massive outliers

- System KV cache are generated by the BF16 model
- They can be further trained like LLM parameters



- The rest KV cache are generated by quantized model

[1] Ruikang Liu, **Haoli Bai†**, et.al. IntactKV: Improving Large Language Model Quantization by Keeping Pivot Tokens Intact. Findings of ACL, 2024.

Evaluating Quantized Models

[1] Li, S., et al. "Evaluating Quantized Large Language Models." ICML 2024.

- Evaluation Dimensions
  - Effects of quantization on **5 major categories of tasks**
  - Effects of quantization on **11 model families**
  - Effects of quantizing **3 tensor types** on model performance
  - Application scope of **SOTA quantization methods**

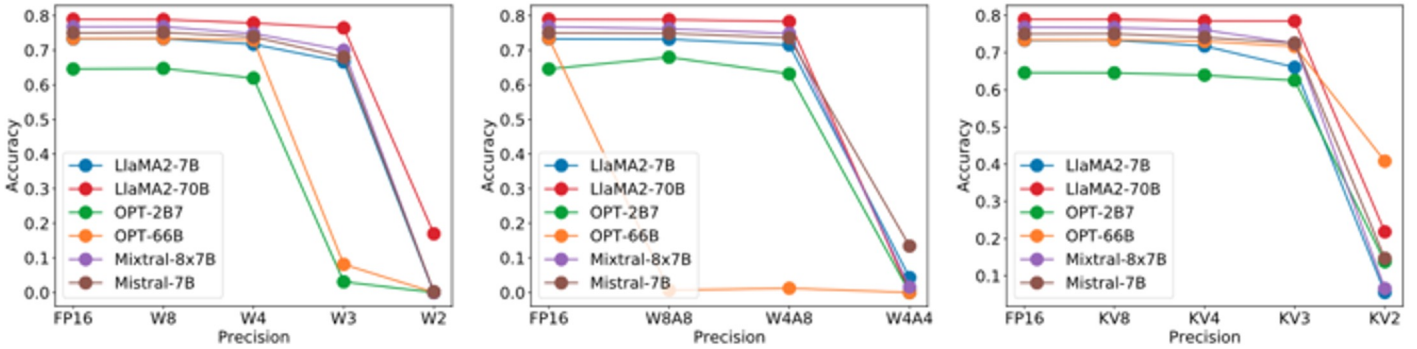
Section	Tasks & Ability	Benchmark	Size	Model Family
Sec. 3	Language Modeling	CHID (Zheng et al., 2019)	2002	OPT (125M-66B), LLaMA2 (7B-70B), Falcon (7B-180B), Bloomz (560M-176B), Mistral(7B, 8×7B)
		Winogrande (Sakaguchi et al., 2021)	1267	
	Understanding	RACE (Lai et al., 2017)	3489	
		LAMBADA (Paperno et al., 2016)	5153	
	Reasoning	SIQA (Sap et al., 2019)	1950	
		PIQA (Bisk et al., 2020)	1876	
Sec. 4	In-Context Learning	MMLU (Hendrycks et al., 2021b)	14079	LLaMA2 (7B-70B), Falcon (7B-180B), ChatGLM (6B), Mistral (7B, 8×7B) Gemma (2B, 7B), Mamba (2.8B)
		CEval (Huang et al., 2023)	13948	
	Multi-Step Reasoning	GSM8K (Cobbe et al., 2021)	1319	
		StrategyQA (Geva et al., 2021)	2290	
	Instruction-Following	Hellaswag (Zellers et al., 2019)	10003	
		ARC (Clark et al., 2018)	7787	
	Self-Calibration	MMLU (Hendrycks et al., 2021b)	14079	
App. D	Ethics	ETHICS (Hendrycks et al., 2021a)	15160	
	Hallucination	TruthfulQA (Lin et al., 2021)	817	
	Robustness	AdvGLUE (Wang et al., 2021)	738	
Sec. 5	Dialogue	MT-bench (Zheng et al., 2023a)	80	(+ StableLM-3B)
Sec. 6	Long-Context	Longeval (Li et al., 2023)	3000	Vicuna (7B, 13B), LongChat (7B, 13B), ChatGLM (6B), Mistral (7B, 8×7B)
		Multi-Doc QA (Liu et al., 2023a)	700	

Paper link

Open source

Evaluating Quantized Models

- Effects of Quantization on Tensor Types
  - The larger the model size, the higher the tolerance for Weight and KVcache Quantization, and the lower the tolerance for Activation Quantization.
  - The larger the model size, the fewer outliers in the Weight and KV Cache tensors, and the more outliers in the Activation tensors.

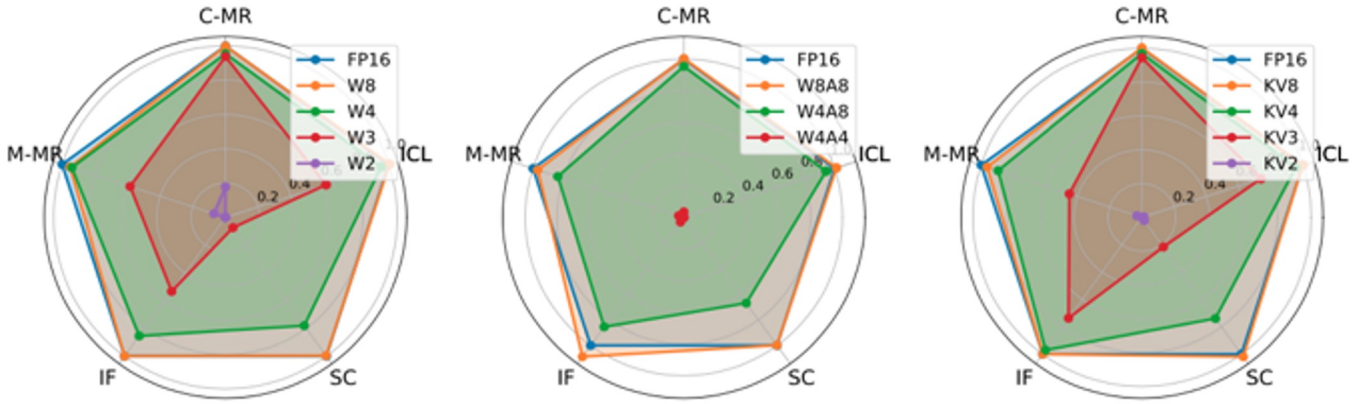


(a) Weight-only Quant. (b) Weight-Activation Quant. (c) KV Cache Quant.

[1] Li, Shiyao, Ning, Xuefei, et al. "Evaluating Quantized Large Language Models." *ICML2024*.

Evaluating Quantized Models

- Effects of Quantization on Emergent Abilities
  - The tolerance to quantization varies across the four abilities, listed in descending order of tolerance: In-context Learning ~ Instruction Following > Multi-Step Reasoning ~ Self-calibration.

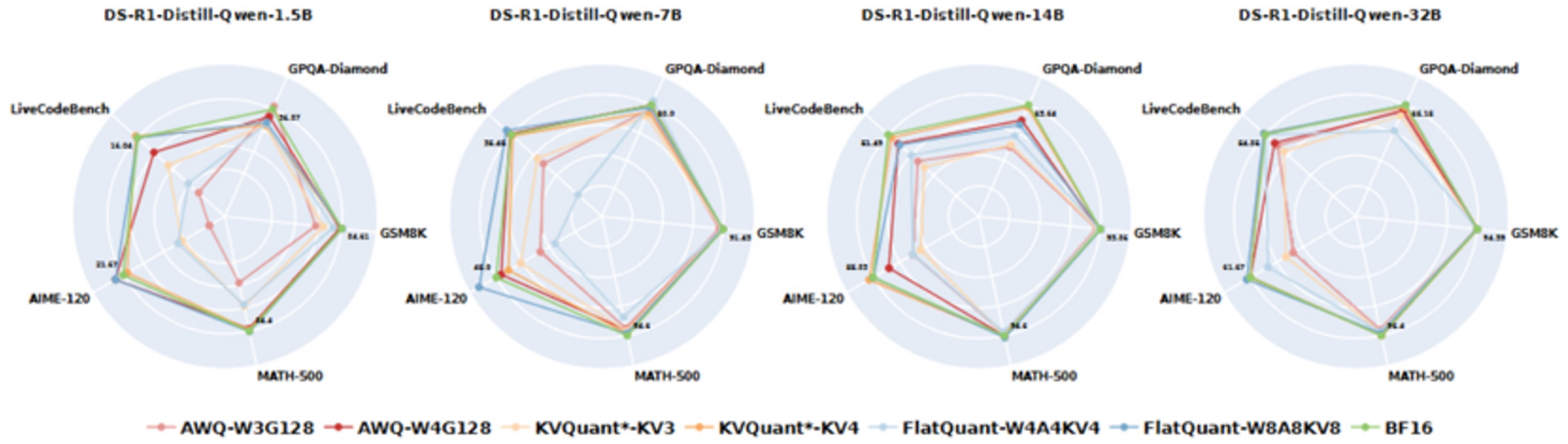


(a) Weight-only Quant. on LLaMA2-7B (b) Weight-Activation Quant. on LLaMA2-7B (c) KV Cache Quant. on LLaMA2-7B

[1] Li, Shiyao, Ning, Xuefei, et al. "Evaluating Quantized Large Language Models." *ICML2024*.

Evaluating Quantized Reasoning Models

- **Reasoning LLMs**: Qwen 1.5B - 32B distilled from DeepSeek-R1
- **Hard tasks** (e.g., **AIME**) suffer more than easier ones (e.g., **GSM8K**)
  - W8A8 and W4A16 is safe to use (<1% acc drop)
  - W4A4 and KV4 can be still risky in practice

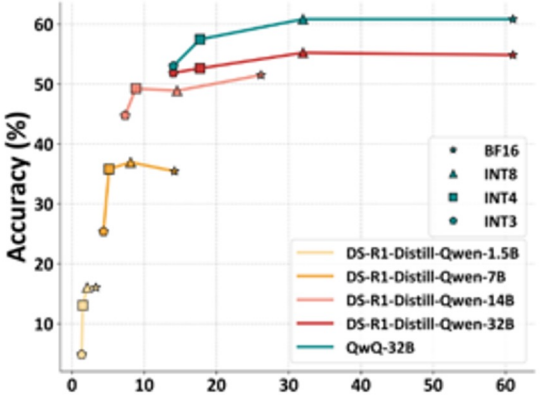


[1] Liu, R., Sun Y., Zhang M., Bai H., et al. "Quantization Hurts Reasoning? An Empirical Study on Quantized Reasoning Models". *COLM 2025*.

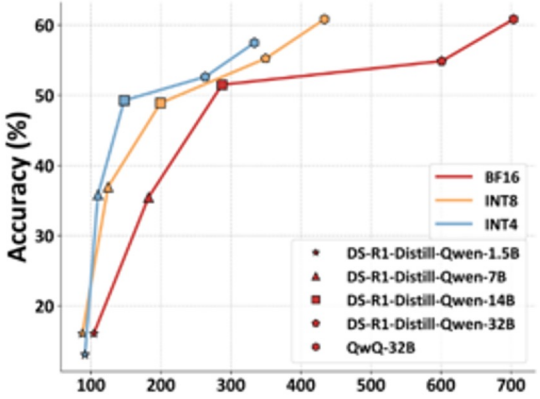
Evaluating Quantized Reasoning LLMs

The scaling effect of quantized reasoning LLMs

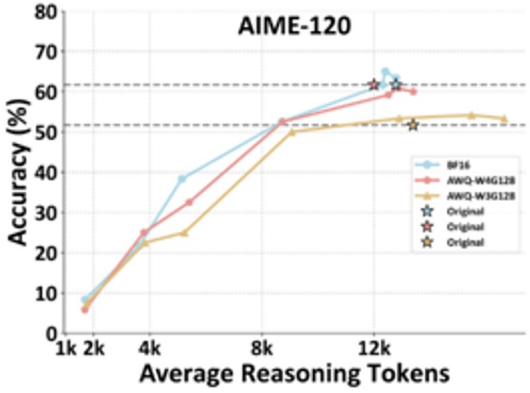
- **(a) & (b):** Large quantized LLMs are preferred to small BF16 LLMs w.r.t. size and latency
- **(c) Test-time scaling**: higher accuracy with more reasoning tokens, but at a slower rate when compared to BF16 models



(a) Model Size (GB)



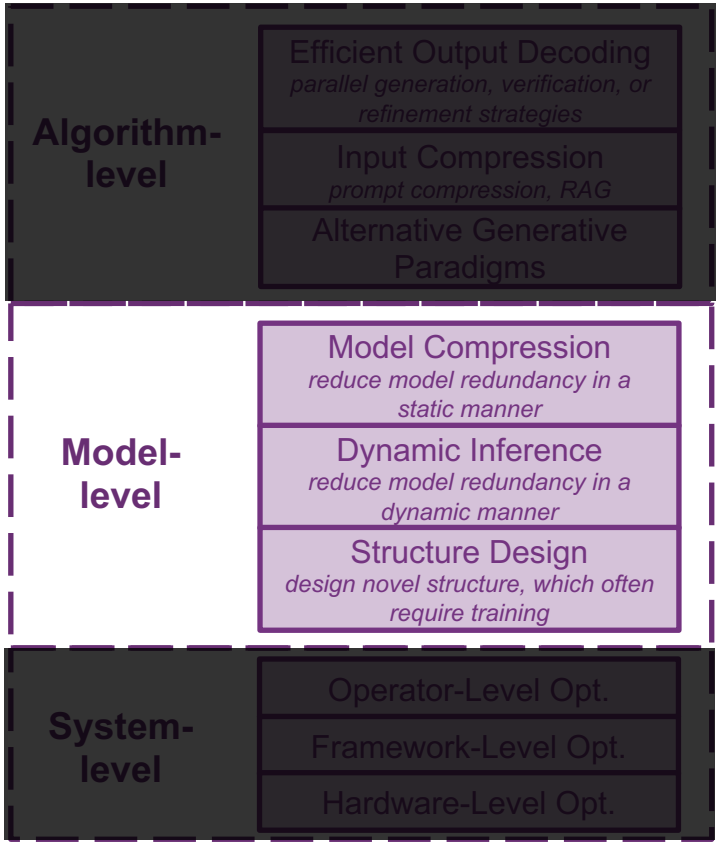
(b) Latency (s)



(c) Test-time Scaling

[1] Liu, R., Sun Y., Zhang M., Bai H., et al. "Quantization Hurts Reasoning? An Empirical Study on Quantized Reasoning Models". *COLM 2025*.

Menu of Techniques



Model Compression

- Quantization
- **Sparse Attention**
- Weight Pruning
- Sharing
- Knowledge Distillation

Dynamic Inference

- Module-granularity
- Model-granularity

Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

Model-level: Sparse Attention

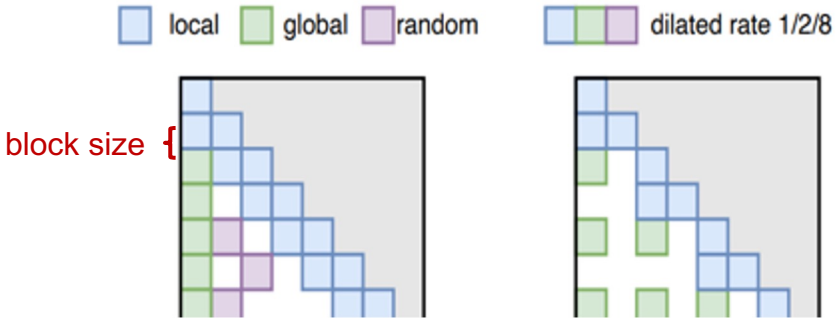
- Sparse Attention
  - Omit certain attention calculations
    - to enhance computational efficiency: saving computation on S and O

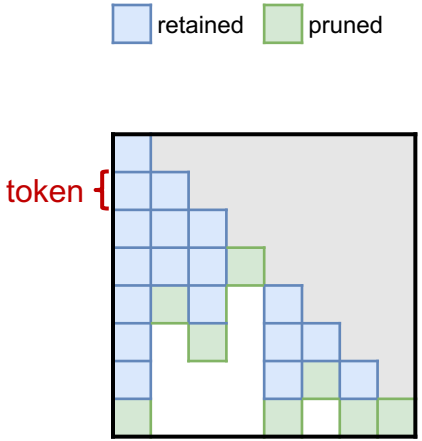
$$\mathbf{S} = \mathbf{Q}\mathbf{K}^T, \quad \mathbf{A} = \text{softmax}(\mathbf{S} + \mathbf{M}), \quad \mathbf{O} = \mathbf{A}\mathbf{V}$$

  - Static vs Dynamic Mask
    - **Static Mask**: the attention mask is predefined and remains fixed.
    - **Dynamic Mask**: the attention mask is determined online based on the input.

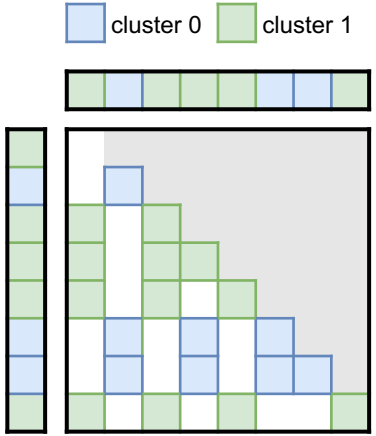
Model-level: Sparse Attention

- Sparse Attention
  - Sparse Pattern: local, global, random, dilated
    - Granularity: blockwise
  - Token pruning vs clustering vs merging

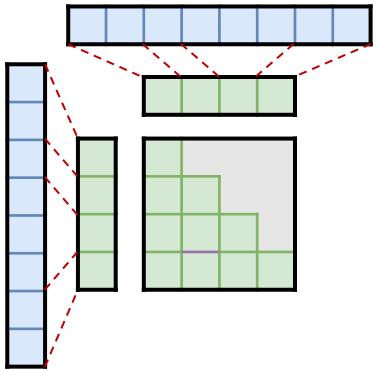




Token pruning



Token clustering



Token merging

Sparse Attention

- Representative studies

	Mask Generation		Sparse Pattern						
	Static	Dynamic	Local	Global	Dilated	Random	Clustering	Pruning	Merging
Sparse Transformers	√		√		√			√	
StreamingLLM	√		√	√					
BigBird	√		√	√		√			
Spatten		√						√	
Reformer		√					√		
H2O		√	√					√	
MoA		√	√	√					
NSA		√	√					√	√

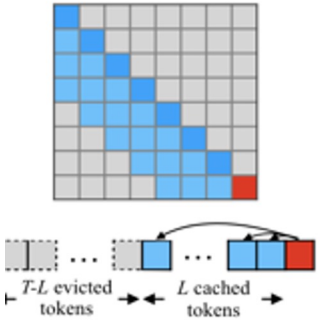
[1] Child, Rewon, et al. "Generating long sequences with sparse transformers." arXiv preprint arXiv:1904.10509 (2019).
[2] Xiao, Guangxuan, et al. "Efficient Streaming Language Models with Attention Sinks." The Twelfth International Conference on Learning Representations.
[3] Zaheer, Manzil, et al. "Big bird: Transformers for longer sequences." Advances in neural information processing systems 33 (2020): 17283-17297.
[4] Wang, Hanrui, Zhekai Zhang, and Song Han. "Spatten: Efficient sparse attention architecture with cascade token and head pruning." 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021.
[5] Kitaev, Nikita, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer." International Conference on Learning Representations.
[6] Zhang, Zhenyu, et al. "H2o: Heavy-hitter oracle for efficient generative inference of large language models." Advances in Neural Information Processing Systems 36 (2023): 34661-34710.
[7] Fu, Tianyu, et al. "Mixture of Attention Spans: Optimizing LLM Inference Efficiency with Heterogeneous Sliding-Window Lengths." Second Conference on Language Modeling.
[8] Yuan, Jingyang, et al. "Native sparse attention: Hardware-aligned and natively trainable sparse attention." arXiv preprint arXiv:2502.11089 (2025).

StreamingLLM

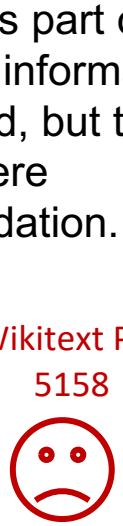
- **Sparse Attention: StreamingLLM**

Motivation

- Sliding-window attention mechanism discards part of the long-term historical information that is rarely needed, but this often results in severe performance degradation.

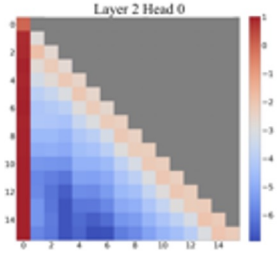


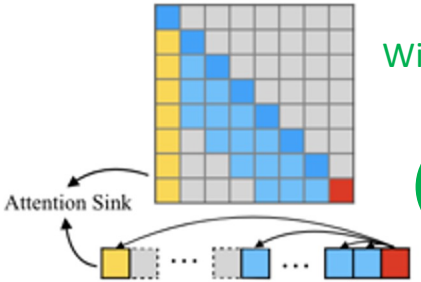
Wikitext PPL
5158



Method

- **Key finding:** LLMs tend to assign attention scores to the initial tokens (attention sink).
- Therefore, StreamingLLM not only retains sliding-window attention but also preserves the attention scores of the initial tokens.





Wikitext PPL
5.40

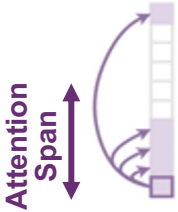
[1] Xiao, Guangxuan, et al. “Efficient Streaming Language Models with Attention Sinks.” ICLR 2024.

MoA

- **Sparse Attention: MoA**

Motivation

- Existing methods apply a homogeneous sparse mask to each attention head, which fails to capture the diverse attention patterns in LLMs and consequently leads to a significant drop in model performance.

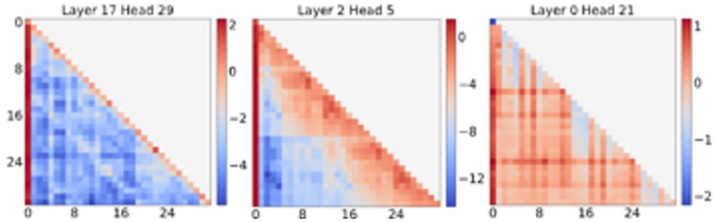


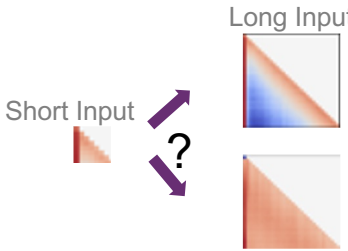
StreamingLLM [2] with fixed-length local attention and global attention on the initial tokens

homogeneous sparse mask

Insight

🔍 Different attention heads have different attention spans, requiring heterogeneous sparse attention.





🔍 Different input lengths have different attention span growth patterns, requiring appropriate length growth rules.

[1] Fu, Tianyu, et al. "Mixture of Attention Spans: Optimizing LLM Inference Efficiency with Heterogeneous Sliding-Window Lengths." Second Conference on Language Modeling.

MoA

Method

- **Different attention heads:** require searching for different sparse attention span.
- **Different input lengths:** require searching for suitable attention span growth rules for different attention heads.

Heterogeneous search space

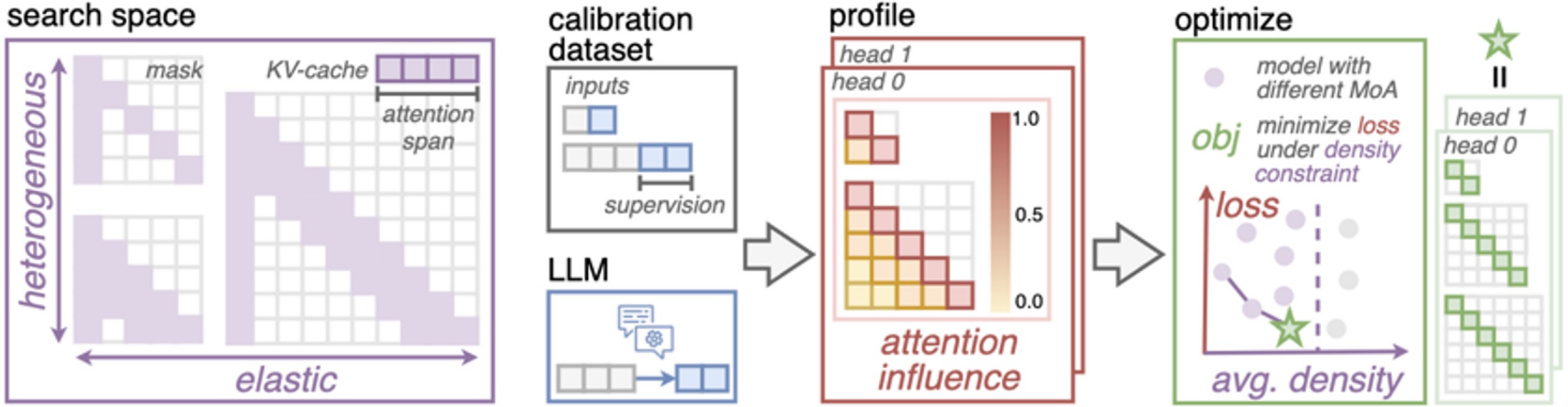
Construct a search space of sparse schemes that includes different sparse attention patterns and their variation rules with sequence length.

Analyze attention value importance

Based on gradients, analyze the impact of different attention values on the prediction results, and obtain the accuracy–sparsity trade-off curve for different sparse schemes.

Optimization

Formulate an optimization problem to select sparse patterns under sparsity constraints, minimizing the impact of sparse attention on accuracy.



[1] Fu, Tianyu, et al. "Mixture of Attention Spans: Optimizing LLM Inference Efficiency with Heterogeneous Sliding-Window Lengths." Second Conference on Language Modeling.

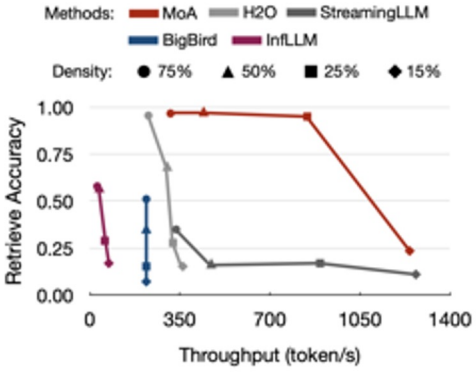
MoA

Results

Increase the inference throughput by about 7×.

Expand the effective context length by 3.9×.

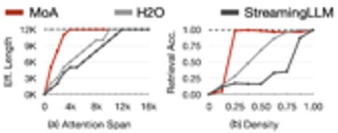
Accuracy-Throughput



Vicuna-7B, 8K input length

Code

Needle-in-a-haystack Task



3.9x

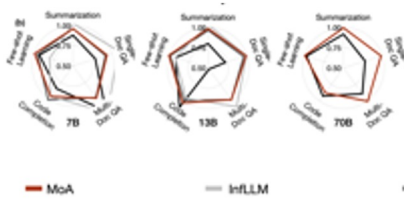
Effective context length

Attention	Retrieve Acc. ↑ 32k	64k	128k	256k
Original	0.98	0.93	0.76	0.37
InfLLM	0.43	0.32	0.25	OOT
StreamingLLM	0.52	0.48	0.41	0.25
MoA	**1.00**	**0.92**	**0.83**	**0.46**

256k

Extrapolatable input length

Long Context Understanding Task



Accuracy Improvement

7-70B models

50% sparsity

Efficiency

1.7x-1.9x

Throughput Improvement

Compared with VLLM on 7B and 13B LLMs using 50% attention sparsity on A100-80GB GPUs

Model	Framework	Attention	4k Batch	4k Throughput	8k Batch	8k Throughput	16k Batch	16k Throughput
7B	vLLM	PagedAttention	30	628.8	15	323.0	8	145.5
	FlexGen	H2O	20	754.9	6	296.3	1	51.7
	HuggingFace	InfLLM	15	62.0	10	37.5	6	19.2
	HuggingFace	StreamingLLM	50	945.1	25	467.3	12	232.0
	HuggingFace	FlashAttention2	30	134.6	15	66.9	8	32.9
		+Static KV-Cache	30	496.1	15	219.5	8	91.6
		+Reduced Attention	30	722.5	15	369.9	8	178.3
		+Increased Batch	50	897.7	25	436.7	12	206.4
		+Kernel (=MoA)	50	**1099.0**	25	**535.7**	12	**257.3**
13B	vLLM	PagedAttention	16	314.8	8	160.5	4	71.1
	FlexGen	H2O	12	330.2	4	138.2	1	37.4
	HuggingFace	InfLLM	8	30.3	5	17.63	3	11.3
	HuggingFace	StreamingLLM	28	478.4	14	241.2	7	116.5
	HuggingFace	FlashAttention2	16	81.3	8	40.8	4	19.8
		+Static KV-Cache	16	264.6	8	111.3	4	62.2
		+Reduced Attention	16	329.6	8	156.4	4	87.3
		+Increased Batch	28	471.5	14	222.6	7	108.3
		+Kernel (=MoA)	28	**550.9**	14	**267.6**	7	**132.3**

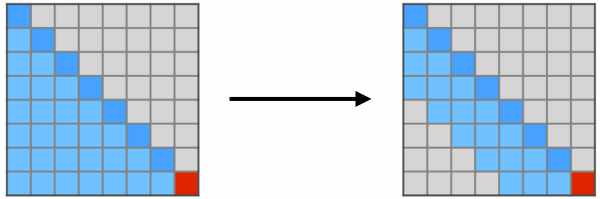
[1] Fu, Tianyu, et al. "Mixture of Attention Spans: Optimizing LLM Inference Efficiency with Heterogeneous Sliding-Window Lengths." Second Conference on Language Modeling.

NSA

- **Sparse Attention: NSA**

Motivation

- Applying sparsity post-hoc forces models to deviate from their pretrained optimization trajectory and causes performance degradation.

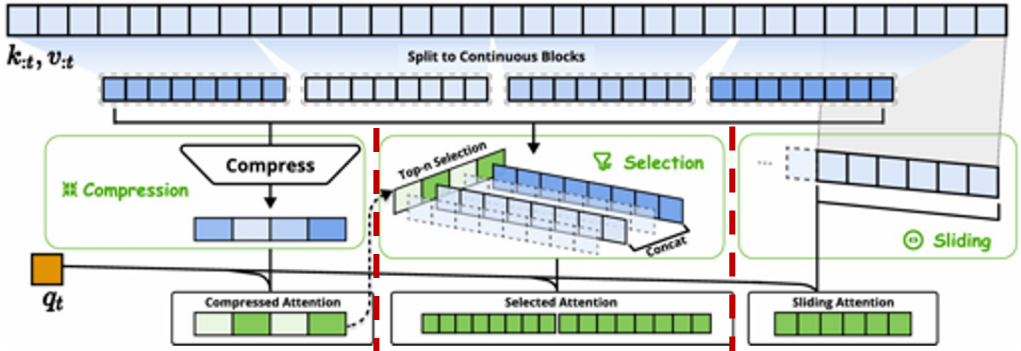


Training:
Dense attention

Inference:
Sparse attention

Method

- Use different sparse patterns and enable end-to-end training.



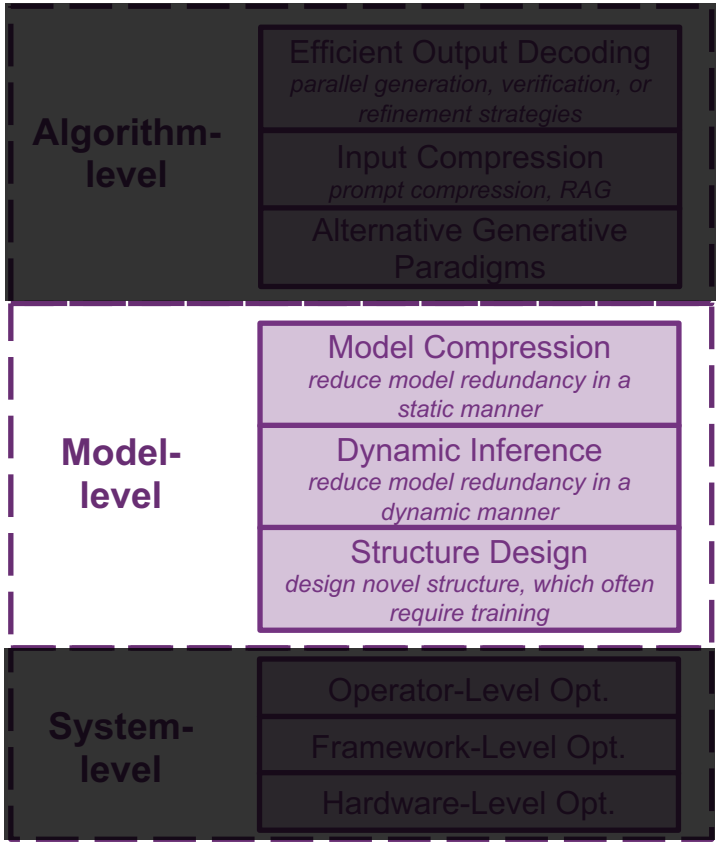
1. Token Merging
Merge multiple tokens into coarse representations.

2. Token Selection
Retain only the most important tokens to apply on fine-grained attention.

3. Local
Attends to nearby tokens within a sliding window.

[1] Yuan, Jingyang, et al. "Native sparse attention: Hardware-aligned and natively trainable sparse attention." arXiv preprint arXiv:2502.11089 (2025).

Menu of Techniques



Model Compression

- Quantization
- Sparse Attention
- **Weight Pruning**
- Sharing
- Knowledge Distillation

Dynamic Inference

- Module-granularity
- Model-granularity

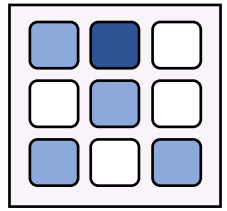
Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

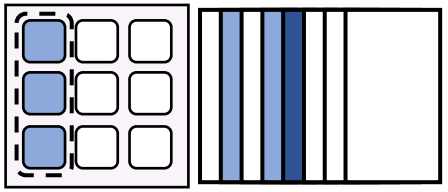
Model-level: Weight Pruning

- Weight Pruning
  - Remove less critical weights and structures from models

	Unstructured Pruning	Structured Pruning
Granularity	Individual weight values	Structural units, e.g., channels, layers, experts
Performance loss	Low	High
Actual speed-up on hardware	No	Yes
Research directions	(1) accelerate pruning process (2) design effective pruning strategies (e.g., pruning metrics, pruning ratios)	(1) decide structured pattern (2) design effective pruning metrics
Representative Studies	**SparseGPT**, Prune and Tune, ISC, BESA	**LLM-Pruner**, LLaMA-Sheard, ZipLM, LoRAPrune, EEP



Unstructured Pruning
Granularity: Weight



Structured Pruning
Granularity: Channel/Group/Layer

SparseGPT

Idea

- Removing weights and updating the remaining ones to compensate for the error.

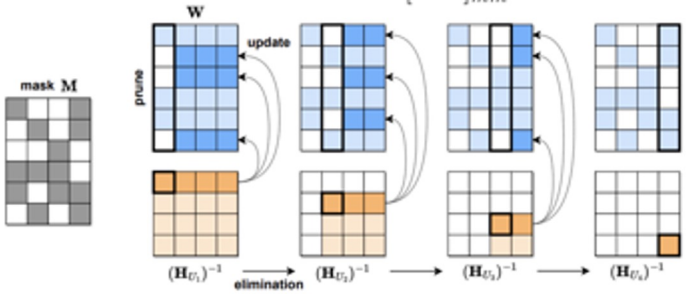
Category

- Type: Unstructured Pruning
- Granularity: Individual weight values

Method

- Incrementally prune weights in each column of the weight, using a sequence of Hessian inverses, and updating the remainder of the weights.

$$\delta_m = -\frac{w_m}{[\mathbf{H}^{-1}]_{mm}} \cdot \mathbf{H}^{-1}_{:,m}$$



$$(\mathbf{H}_{U_{j+1}})^{-1} = \left(\mathbf{B} - \frac{1}{[\mathbf{B}]_{11}} \cdot \mathbf{B}_{:,1}\mathbf{B}_{1,:}\right)_{2:,2:} \qquad \mathbf{B} = (\mathbf{H}_{U_j})^{-1}$$

[1] Elias Frantar, et al. “SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot” ICML 2023.

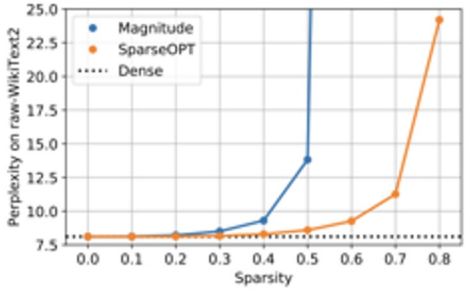
SparseGPT

Results

- **High Sparsity, Low Accuracy Loss**: Prunes OPT-175B to 60% sparsity in one shot with a negligible increase in perplexity
- **High Efficiency**: Process the 175-billion-parameter models in under 4.5 hours, removing more than 100 billion weights.

Table 2. ZeroShot results on several datasets for sparsified variants of OPT-175B.

Method	Spars.	Lamb.	PIQA	ARC-e	ARC-c	Story.	Avg.
Dense	0%	75.59	81.07	71.04	43.94	79.82	**70.29**
Magnitude	50%	00.02	54.73	28.03	25.60	47.10	**31.10**
SparseGPT	50%	78.47	80.63	70.45	43.94	79.12	**70.52**
SparseGPT	4:8	80.30	79.54	68.85	41.30	78.10	**69.62**
SparseGPT	2:4	80.92	79.54	68.77	39.25	77.08	**69.11**



[1] Elias Frantar, et al. “SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot” ICML 2023.

LLM-Pruner

Idea

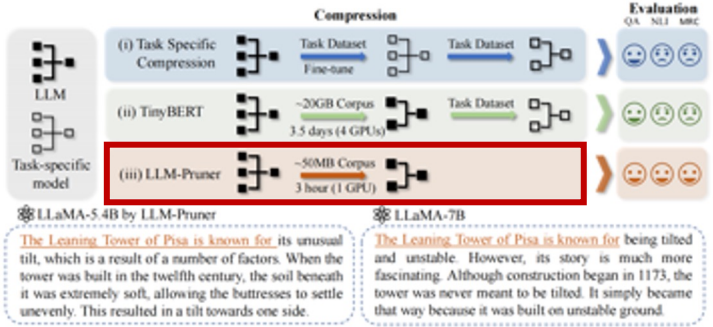
- Identify and remove non-critical, coupled structures.

Category

- Type: Structured Pruning
- Granularity: Head, Channel

Methods

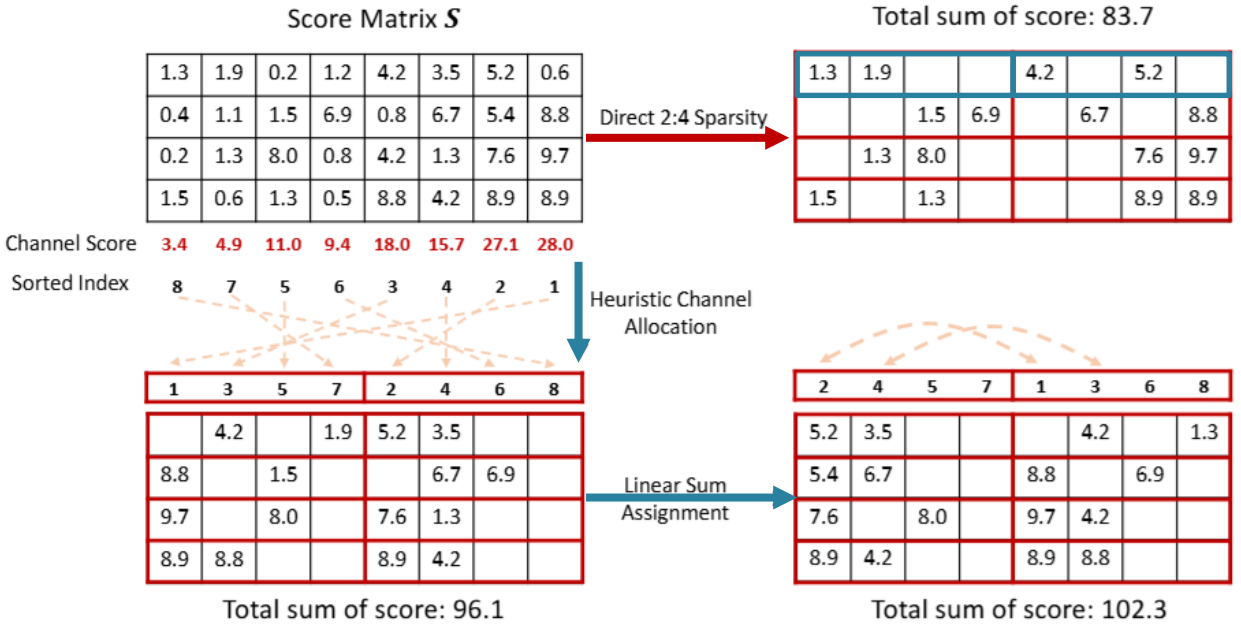
- **LLM-Pruner** automatically identifies and removes non-critical, coupled structures based on gradient information, and recovers performance using a LoRA with a small dataset.



[1] Xinyin Ma, et al. “LLM-Pruner: On the Structural Pruning of Large Language Models” Neurips 2023.

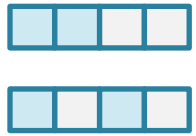
Channel Permutation for Better N:M Sparsity

Find a **permutation strategy** that preserves more important parameters under N:M sparsity



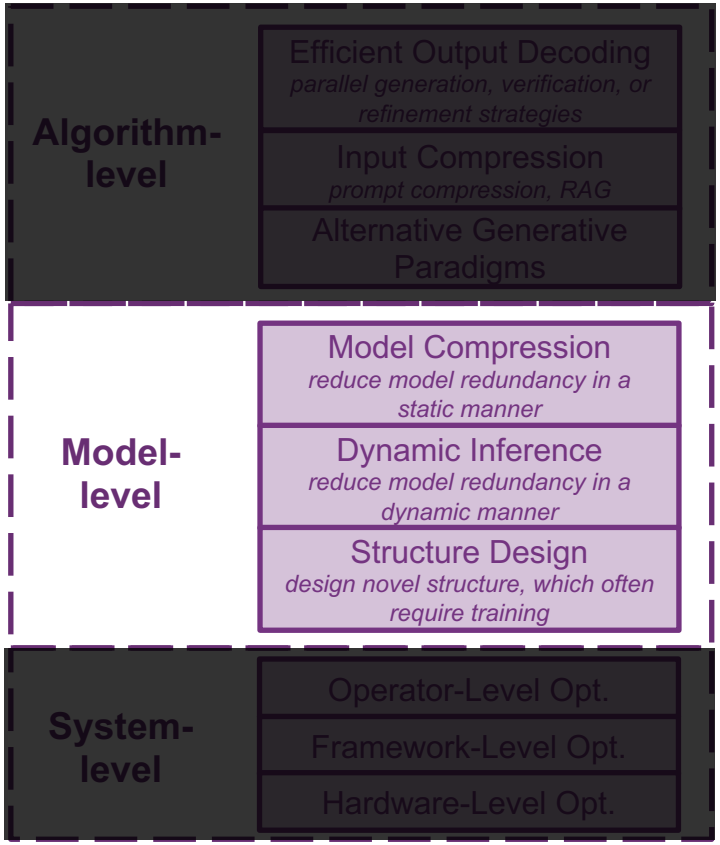
2:4 Sparsity

2 zeros out of every 4 contiguous elements



[1] Zhang, Y., Bai, H., et al. "Plug-and-play: An Efficient Post-training Pruning Method for Large Language Models". *ICLR 2024*.

Menu of Techniques



Model Compression

- Quantization
- Sparse Attention
- Weight Pruning
- **Sharing**
- Knowledge Distillation

Dynamic Inference

- Module-granularity
- Model-granularity

Structure Design

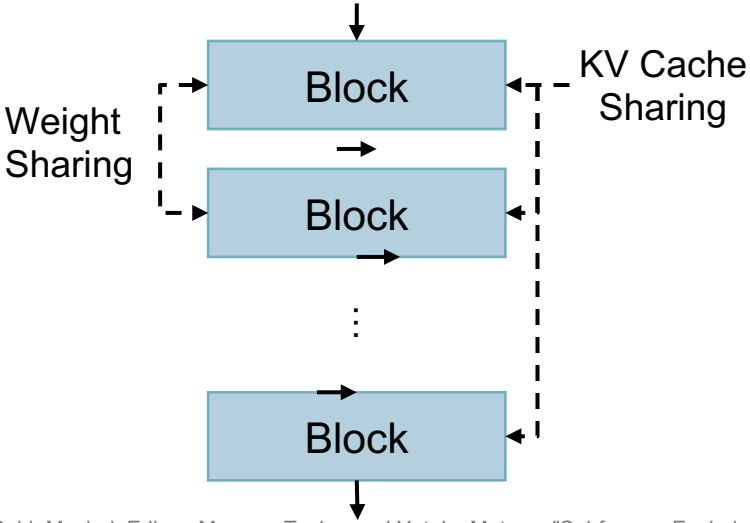
- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

Sharing

- **Definition**

Reuse parameters, states, or intermediate results across different parts of the model.

- **Shared Tensor Type**



	Shared Tensor Type	
	Weight	KV Cache
Subformer	√	
MobileLLM	√	
Dynamic layer tying	√	
LCKV		√
CLA		√

[1] Reid, Machel, Edison Marrese-Taylor, and Yutaka Matsuo. "Subformer: Exploring weight sharing for parameter efficiency in generative transformers." arXiv preprint arXiv:2101.00234 (2021).
[2] Liu, Zechun, et al. "Mobilellm: Optimizing sub-billion parameter language models for on-device use cases." Forty-first International Conference on Machine Learning. 2024.
[3] Hay, Tamir David, and Lior Wolf. "Dynamic Layer Tying for Parameter-Efficient Transformers." The Twelfth International Conference on Learning Representations.
[4] Wu, Haoyi, and Kewei Tu. "Layer-Condensed KV Cache for Efficient Inference of Large Language Models." Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024.
[5] Brandon, William, et al. "Reducing transformer key-value cache size with cross-layer attention." Advances in Neural Information Processing Systems 37 (2024): 86927-86957.

MobileLLM

- **MobileLLM**

Idea

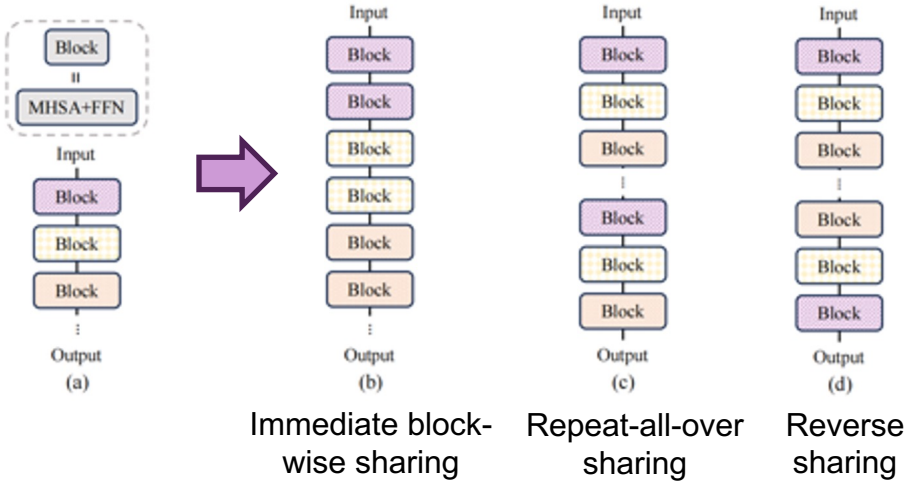
- Weight sharing between two adjacent blocks avoids weight movement, requiring only computing the block twice and incurring minimal latency overhead.

Category

- Type: Weight Sharing

Methods

- Design three different weight-sharing strategies:



Immediate block-wise sharing | Repeat-all-over sharing | Reverse sharing

[1] Zechun Liu, et al. "MobileLLM: Optimizing Sub-billion Parameter Language Models for On-Device Use Cases" ICML 2024.

LCKV

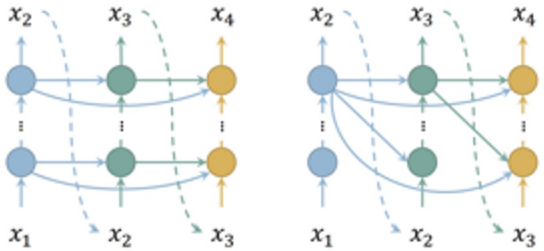
- **Layer-Condensed KV Cache**

Motivation

- Existing methods focus on compressing the KV cache sequence length.
- This approach reduces the number of cached layers, not just the sequence length.

Methods

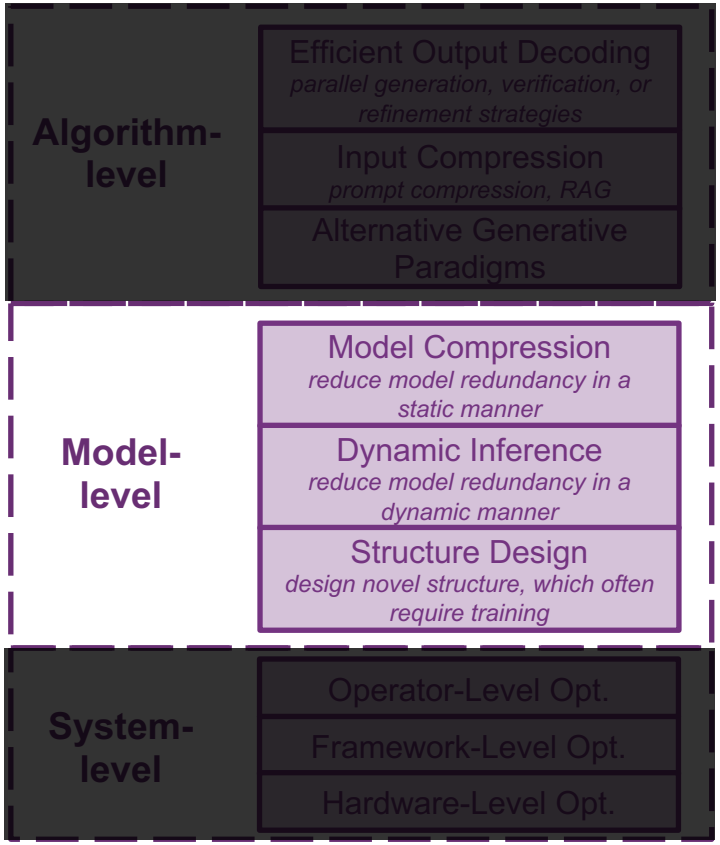
- In LCKV, all layers attend to only **the top layer's KVs**.
- A few "warmup" layers with standard attention are kept to maintain performance.



(a) Standard transformer (b) Our model

[1] Haoyi Wu, et al. “Layer-Condensed KV Cache for Efficient Inference of Large Language Models” ACL 2024.

Menu of Techniques



Model Compression

- Quantization
- Sparse Attention
- Weight Pruning
- Sharing
- **Knowledge Distillation**

Dynamic Inference

- Module-granularity
- Model-granularity

Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

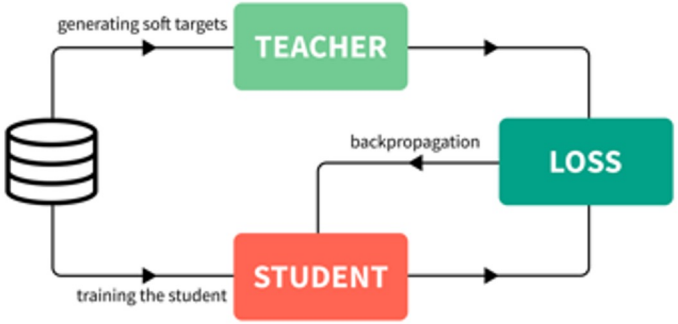
Knowledge Distillation

- **Motivation**

Although the compressed lightweight model achieves better hardware efficiency, its accuracy is lower under conventional training methods. It is necessary to design training approaches to achieve better accuracy recovery.

- **Definition**

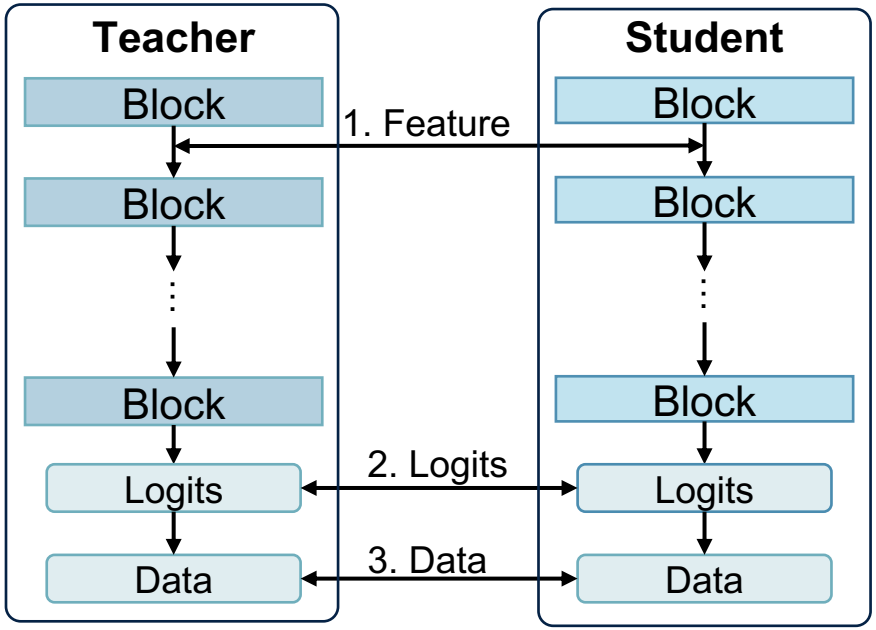
Knowledge Distillation: Use a teacher model to guide the training of a student model, enabling the student model to learn the "knowledge" of the teacher model to help improve its accuracy.



The teacher model (large) helps the student model (small) recover accuracy.

Knowledge Distillation

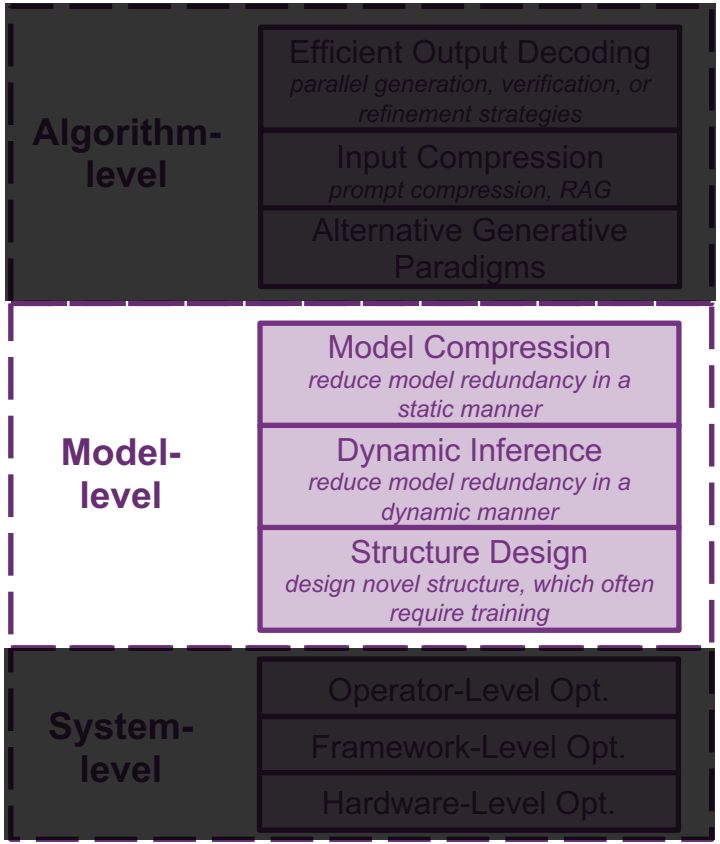
- **Aligning Objective**



	Objective		
	Feature	Logits	Data
TED	√		
MiniLLM		√	
GKD		√	
DISCO			√
MCKD			√
DeepSeek-R1			√

[1] Liang, Chen, et al. "Less is more: Task-aware layer-wise distillation for language model compression." International Conference on Machine Learning. PMLR, 2023.
[2] Gu, Yuxian, et al. "MiniLLM: Knowledge Distillation of Large Language Models." The Twelfth International Conference on Learning Representations.
[3] Agarwal, Rishabh, et al. "Gkd: Generalized knowledge distillation for auto-regressive sequence models." CoRR (2023).
[4] Chen, Zeming, et al. "DISCO: Distilling Counterfactuals with Large Language Models." Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics. 2023.
[5] Zhao, Jiachen, et al. "Multistage collaborative knowledge distillation from large language models." arXiv preprint arXiv:2311.08640 (2023).
[6] Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." arXiv preprint arXiv:2501.12948 (2025).

Menu of Techniques



Model Compression

- Quantization
- Sparse Attention
- Weight Pruning
- Sharing
- Knowledge Distillation

Dynamic Inference

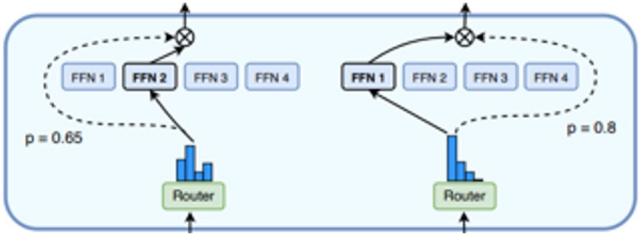
- Module-dimension
- Model-dimension

Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

Model-level: Dynamic Inference

- **Motivation & Definition**
- During neural network inference, not all data needs to go through the same computation flow. The core design idea of dynamic inference algorithms is to determine the required computations based on the runtime input data.
  - **What is the data granularity for dynamic inference** (e.g., query-level, token-level, etc.)
  - **Which dimensions are dynamically adjusted** (e.g., layer, model)
  - **How to dynamically adjust the corresponding dimensions based on input data** (e.g., training a router)



MoE: Token-level dynamic module routing

R2R: Token-level dynamic model routing

[1] Dynamic Neural Networks: A Survey, Han et al., IEEE Transactions on Pattern Analysis and Machine Intelligence.

Mixture-of-Depths

- **Mixture-of-Depths**

Idea

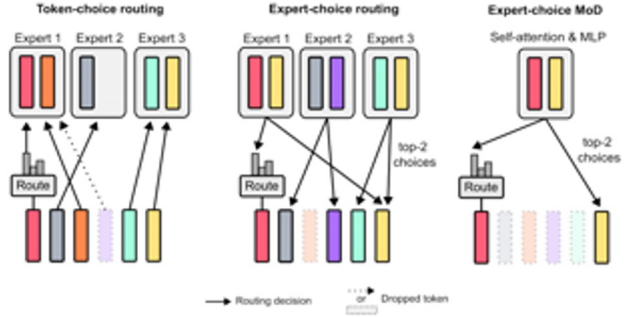
- In language modeling, not all tokens and sequences require the same time or effort to accurately make a prediction.

Category

- **Data Granularity**: token-level
- **Dimension**: layer
- **Method**: training a router

Methods

- At specific layers, a learned router selects the **top-k** most important tokens to be processed by the self-attention and MLP blocks, while other tokens bypass these computations through a simple residual connection.



[1] David Raposo, et al. “Mixture-of-Depths: Dynamically allocating compute in transformer-based language models.” ArXiv 2024.

RouteLLM

- **RouteLLM**

Motivation

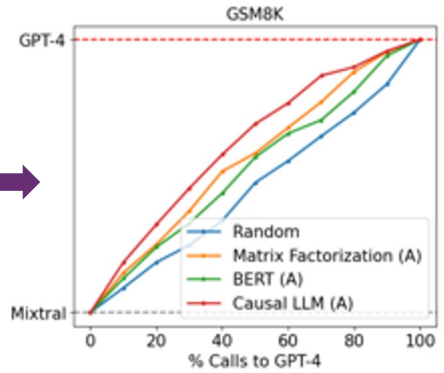
- Direct simpler queries to smaller models and more complex ones to larger models to balance response quality with cost efficiency.

Category

- **Granularity**:query-level
- **Dimension**: model
- **Method**: similarity-based retrieval / training a router

Methods

- RouteLLM trains a router model on human preference data to intelligently direct queries to either a **strong, expensive** LLM or a **weak, cheap** one.
- routers outperform random baselines



[1] David Raposo, et al. "RouteLLM: Learning to Route LLMs with Preference Data." ArXiv 2024.

Recent Work: SLM-LLM Mix Inference (R2R)

Use small language model (SLM) and LLM for different reasoning steps

Motivation

Fast but weak SLM
slow but strong LLM

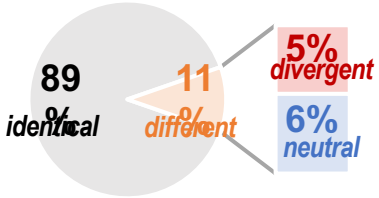
Tested results on AIME'24-25

Type	Model	Accuracy	Latency (s / question)
SLM	R1-1.5B	☹ 9%	☺ 199
LLM	R1-32B	☺ 45%	☹ 498

We should selectively use SLM and LLM for different generation steps, constructing a **fast and strong** mix-inference method

Insight

Given same context, SLM and LLM predictions are often identical



- 89% **identical** predictions
- 11% **different** predictions
  - some are **neutral**, like alternative expressions
  - only few **diverge** the meaning, logic, or conclusion of reasoning

[1] Tianyu, Fu, et al. "Efficiently Navigating Divergent Reasoning Paths with Small-Large Model Token Routing" *Submitted to NeurIPS'25. [Under Review]*

Recent Work: SLM-LLM Mix Inference (R2R)

Label divergent token, then train a neural token-router, utilizing LLMs only for path-divergent tokens during SLM generation

Method **Label divergent tokens generate model preference training data**

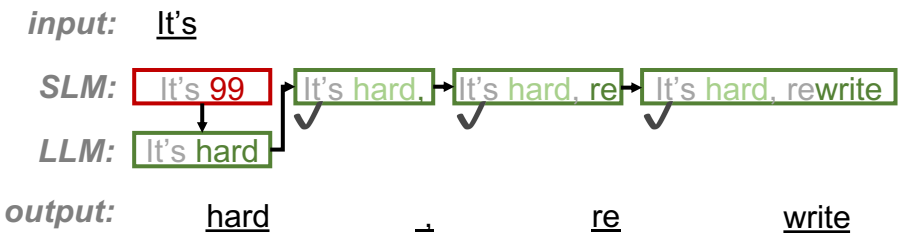
Key idea:

Step1. **find** all predictions where SLM-LLM **differ**

Step2. from the difference, let LLM **continue generation** until the end of current sentence, to understand difference's impact

Step3. ask **another LLM to verify** if difference causes **divergence**

Method **Train neural router, route to LLM for divergent SLM tokens**



Routing scheme:

We train a 56M neural router

Given SLM output token & its last-layer hidden states,

it **classifies** whether this token is **divergent,**

Immediately route to LLM if predicted as divergent

[1] Tianyu, Fu, et al. "Efficiently Navigating Divergent Reasoning Paths with Small-Large Model Token Routing" *Submitted to NeurIPS'25. [Under Review]*

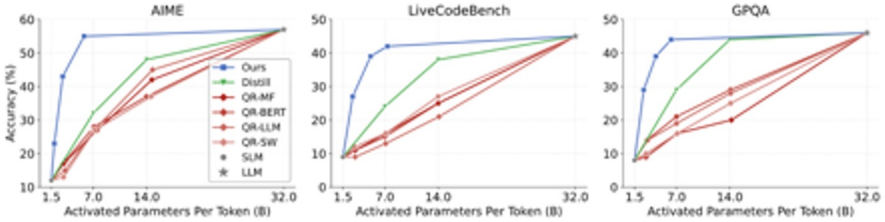
Experimental Results

Mixing R1-1.5B & 32B, uing **R2R** with **5.6B** avg. activated param. per token achieve **performance exceeding R1-14B**

Result

Performance-Efficiency Pareto Frontier

same avg. parameter, better accuracy



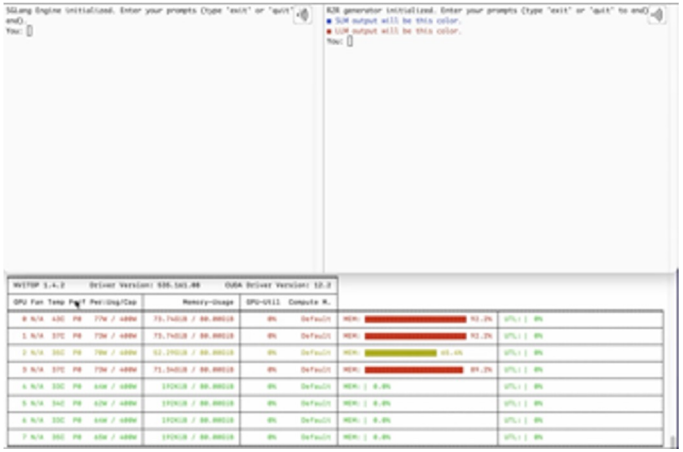
For R2R-5.6B (mix R1-1.5B & 32B)

- **Comparing R1-14B, 1.50x speedup, 1.07x accuracy**
- **Comparing R1-32B, 2.76x speedup,** achieving 92% of its accuracy with only 11%-15% LLM usage

Demo

Reaching 84.3 token/s on two A800-80GB GPUs

source code

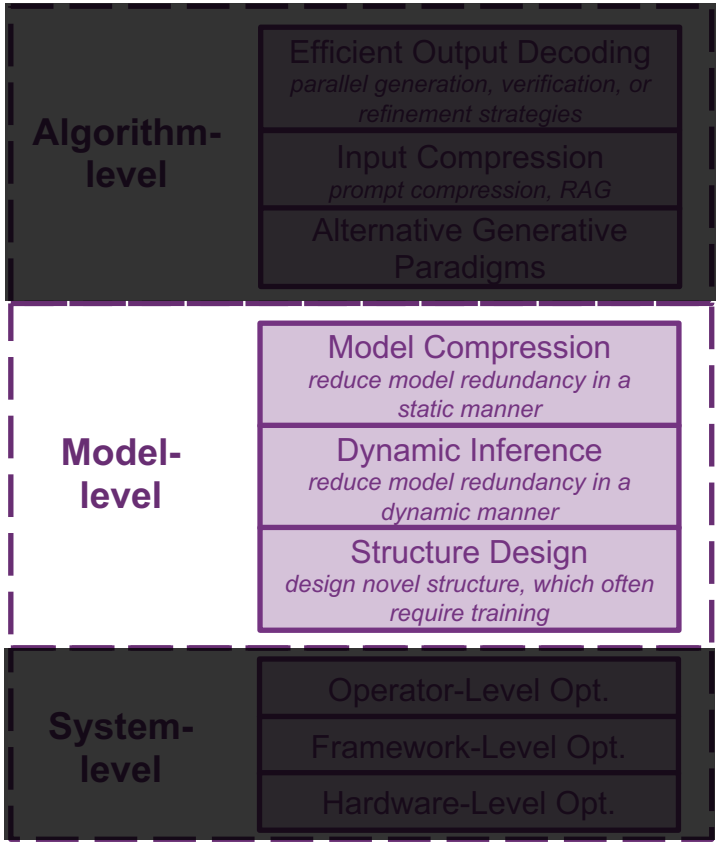


R1-32B
Finished in: **1min 12s**

R2R
Finished in: **32s**

[1] Tianyu, Fu, et al. "Efficiently Navigating Divergent Reasoning Paths with Small-Large Model Token Routing" *Submitted to NeurIPS'25. [Under Review]*

Menu of Techniques



Model Compression

- Quantization
- Sparse Attention
- Weight Pruning
- Sharing
- Knowledge Distillation

Dynamic Inference

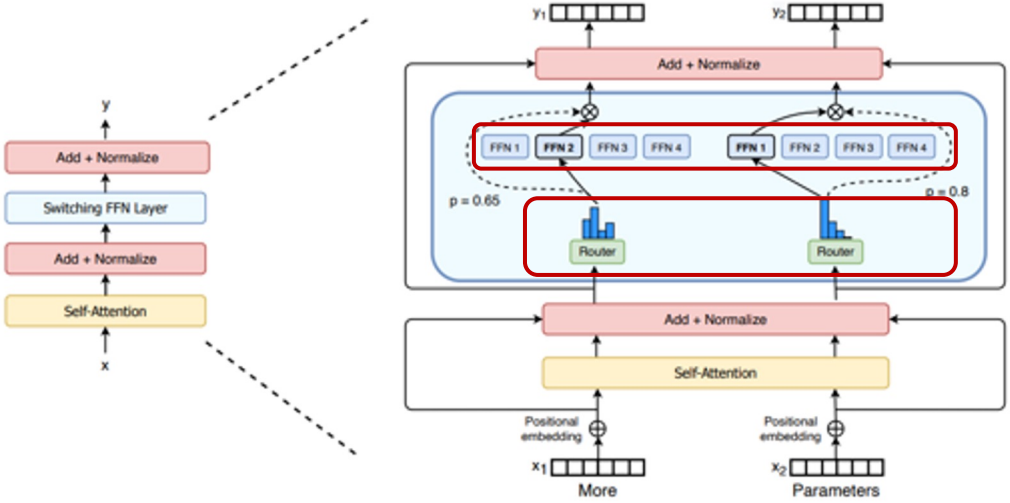
- Module-granularity
- Model-granularity

Structure Design

- Mixture-of-Experts (Efficient FFN)
- Efficient Attention

Model-level: Efficient Structure Design

- Efficient structure design
  - Efficient FFN design: Mixture-of-Expert (MoE, one common architecture follow the dynamic inference idea)



- Expert: module to process different inputs
  - Shared expert: an expert module that is always active for every input
  - Expert granularity: the FFN intermediate hidden dimension
- Router: direct the input to the appropriate expert networks
  - Basically, each token is routed to a fixed number of experts **based on scores produced by the router**.
  - In training or multi-batch inference scenarios, load balancing among experts also needs to be considered.[1]

[1] Mu, Siyuan, and Sen Lin. "A comprehensive survey of mixture-of-experts: Algorithms, theory, and applications." arXiv preprint arXiv:2503.07137 (2025).

MoE

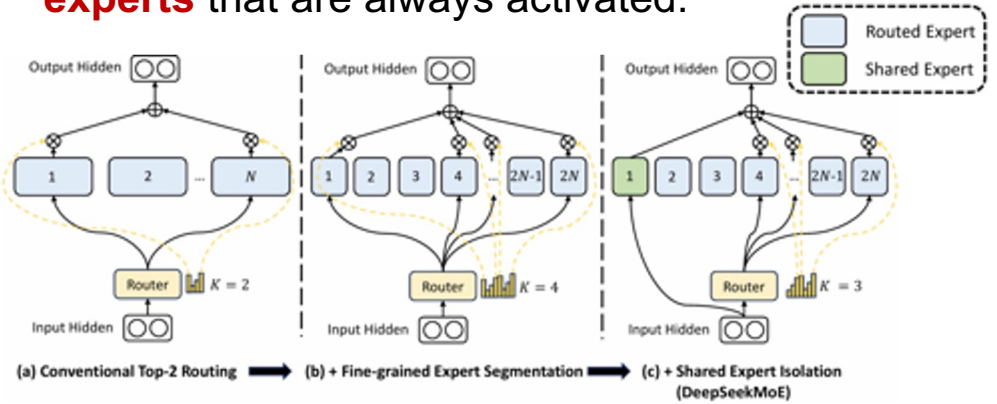
- **DeepSeekMoE**

Motivation

- The designated expert will intend to assemble different types of knowledge in its parameters, which are hard to utilize simultaneously.
- Multiple experts may converge in acquiring shared knowledge in their respective parameters, leading to redundancy in expert parameters.

Methods

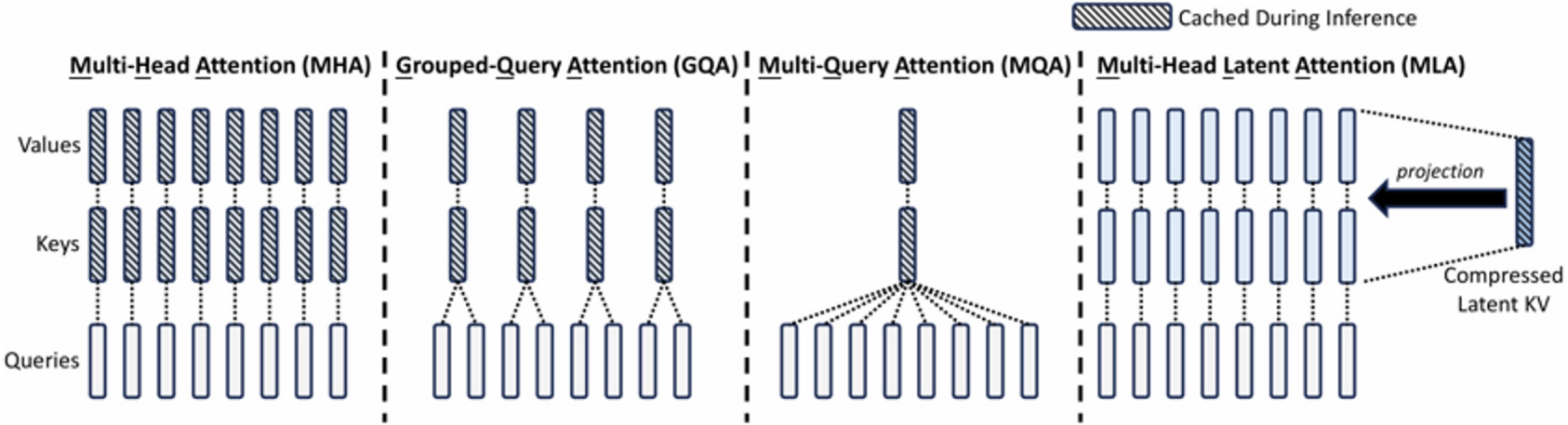
- Segment the experts into a **finer grain** by splitting the FFN intermediate hidden dimension.
- Isolate certain experts to serve as **shared experts** that are always activated.



[1] Dai, Damai, et al. "DeepSeekMoE: Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models." Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2024.

Efficient Attention mechanism

- Efficient structure design
  - Efficient Attention mechanism



Divide query heads into G groups, each of which shares a key and value head.

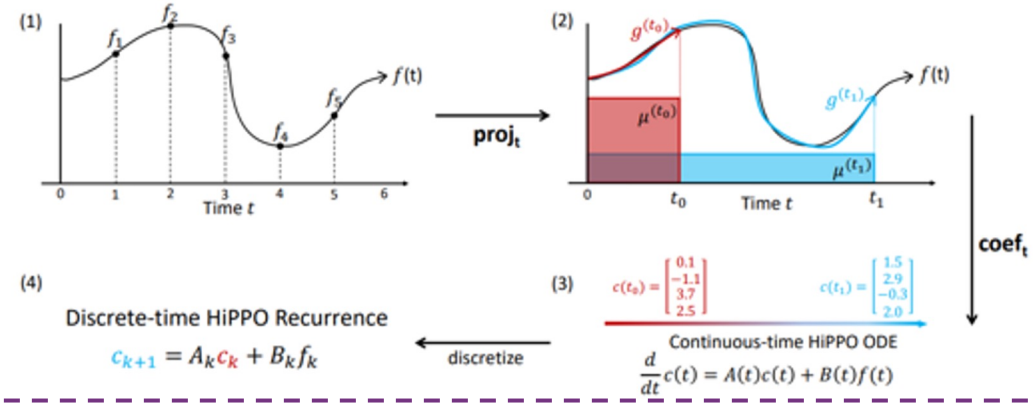
Different heads share a single set of keys and values.

Low-rank compression for keys and values to reduce KV cache.

[1] Liu, Aixin, et al. "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model." arXiv preprint arXiv:2405.04434 (2024).

Model-level: Efficient Structure Design

- Efficient structure design
  - Non-Transformer architecture design
    - State Space Model (SSM)
      - core idea: compress token information into hidden state



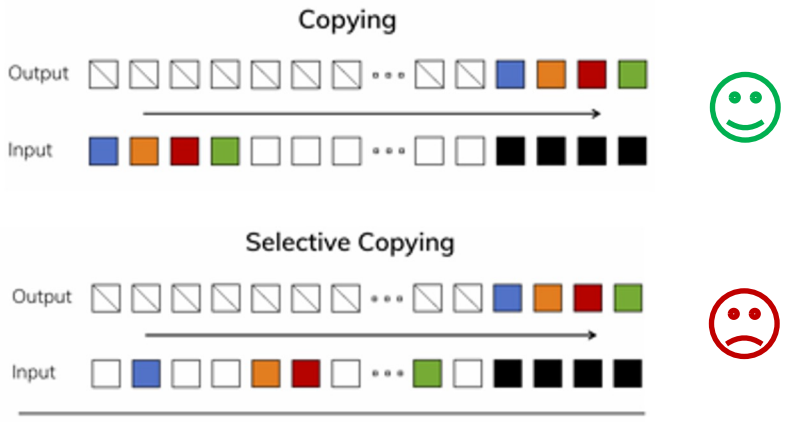
Research directions:
1. Design better parametrization or initialization strategy.
2. Design better model architecture based on SSM.

Mamba

- **Mamba**

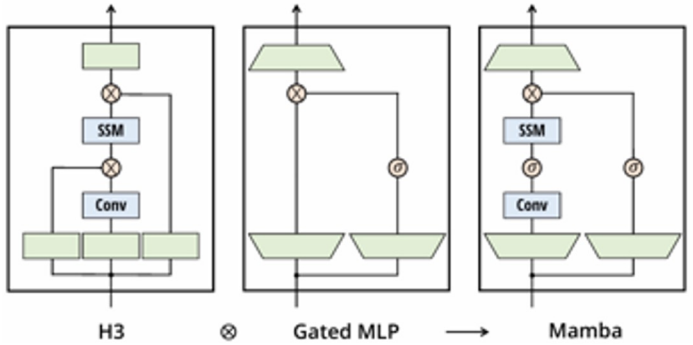
Motivation

- Linear-time-invariant (LTI) SSMs cannot efficiently select data in an input-dependent manner.



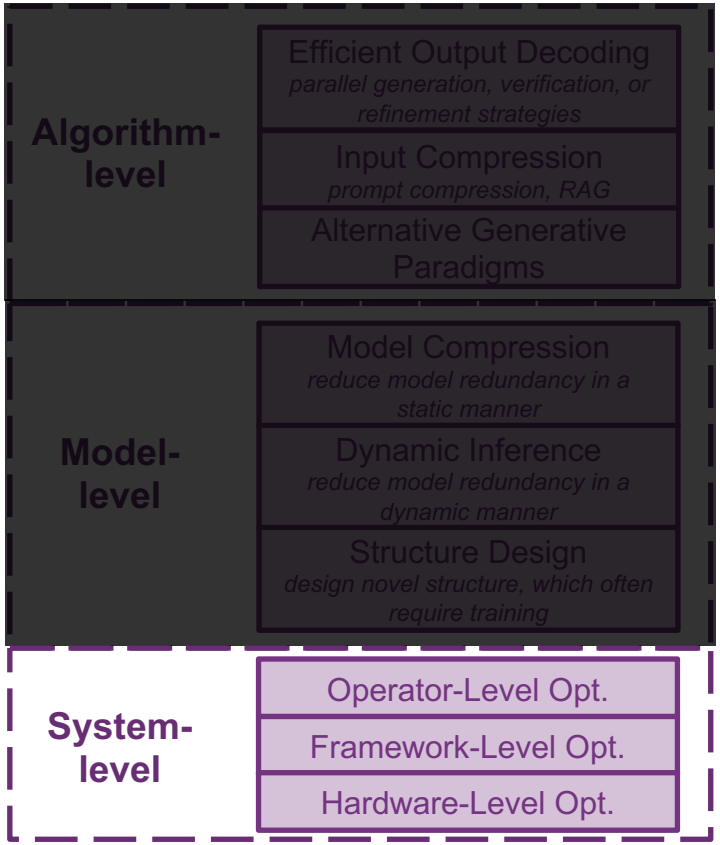
Methods

- Let the SSM parameters be functions of the input, allowing the model to selectively propagate or forget information depending on the current token.



[1] The router can direct simpler queries to smaller models and more complex ones to larger models, thereby balancing response quality with cost efficiency.

Menu of Techniques



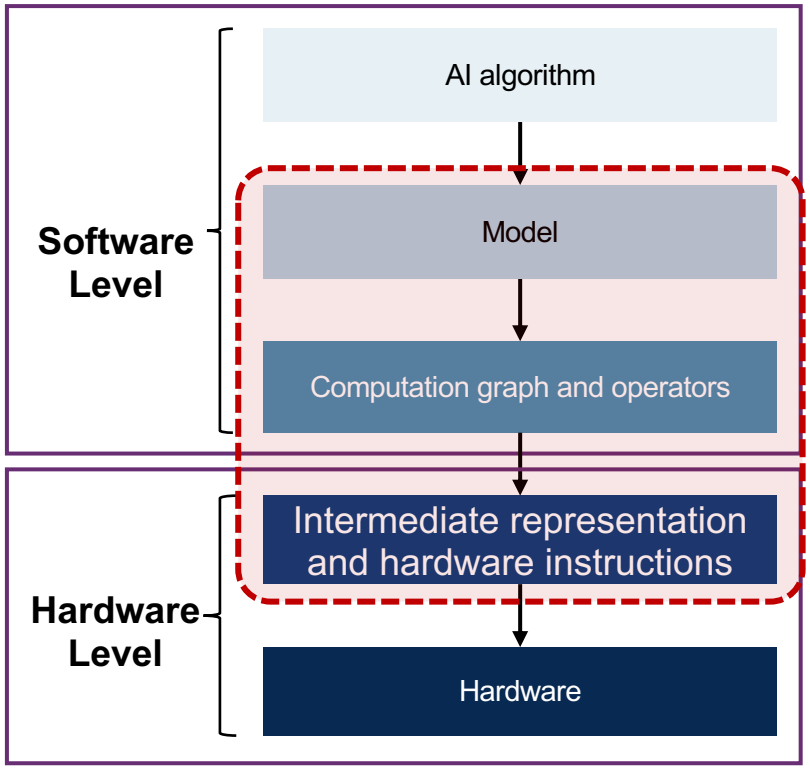
Operator-Level Opt.

Framework-Level Opt.

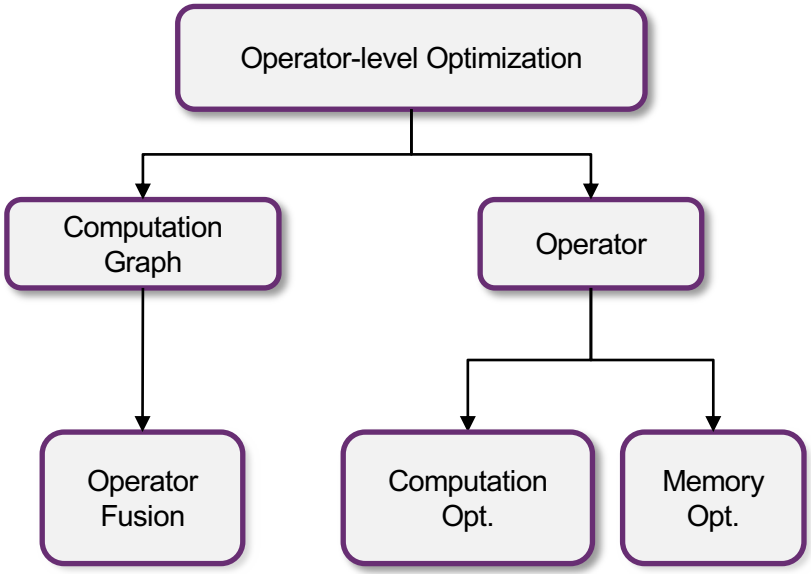
Hardware-Level Opt.

Design Space

- **System Design: Operator-level Optimization**



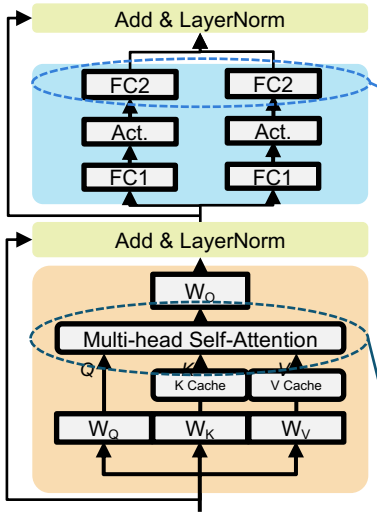
$$\max. utilization,$$
$$\text{s.t.} usage \leq resource.$$



Operator-level Optimization

Method

Optimizes the hardware utilization by tailoring **workload mapping to hardware** specifications



A transformer block

Matmul Operator

Main Operator of NN with Highly Optimized Linear Algebra Libraries

Attention Operator

Core of Transformer

FlashAttention

(NeurIPS 22)
Reduces the memory footprint by fusing attention into one kernel

FlashDecoding++

(MLSys 24)
Reduces the update overhead of attention operator and optimizes GEMM in decoding, achieving up to 4x end-to-end speedup

cuBLAS

(~2007-present)
NVIDIA GPU's official library
Achieves over 90% peak utilization of Tensor Cores

FlashDecoding

(arXiv 23)
Uses sequence partitioning for decoding, achieving up to 50x speedup for long texts

FlashInfer

(MLSys 25 Best Paper)
Dynamically aware workload allocation and using unified mask representation
13%-69% e2e speedup

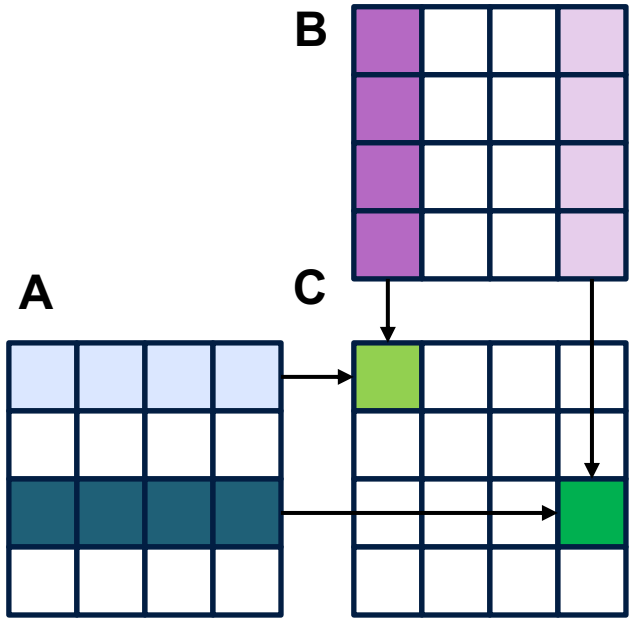
*Matmul: Matrix Multiplication

Matmul Operator

- **Implementation of Matmul on GPU**
  - Observation: No dependencies in C,

  → Parallel computation
  - Method: Each thread loads a row of A and a column of B → computes → writes to C

$$C^{4\times4} = A^{4\times4} \times B^{4\times4}$$



Computational-to-Memory Ratio

An important metric for measuring GPU utilization

Higher value → Higher compute unit utilization

$$\text{Comp-to-Mem Ratio} = \frac{computation\ amount}{memory\ access\ amount}$$

Comp-to-Mem Ratio is only **0.25FLOP/Byte** in this example

*Total computation: 2*4*4*4=128FLOP. Total memory access: 4*(2*4*4*4)=512Byte

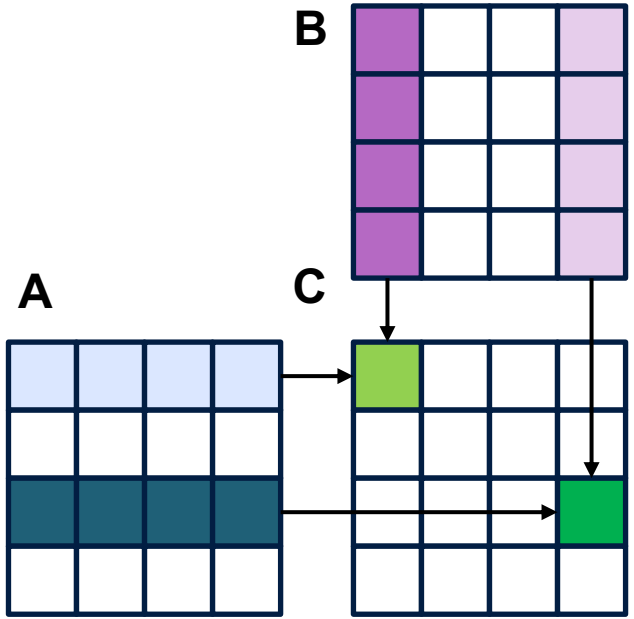
Matmul Operator

- **Implementation of Matmul on GPU**

Operator Implementation:

```
__global__ void MatMul(int *A, int *B, int *C, int width) {
    //eg. width=4
    int row = threadIdx.y;
    int col = threadIdx.x;

    if (row < width && col < width) {
        int sum = 0;
        for (int k = 0; k < width; k++) {
            sum += A[row * width + k] * B[k * width + col];
        } //Each thread processed one row of A, one column of B
        C[row * width + col] = sum;
        //Write result to the corresponding position of C
    }
}
```

$$C^{4\times4} = A^{4\times4} \times B^{4\times4}$$

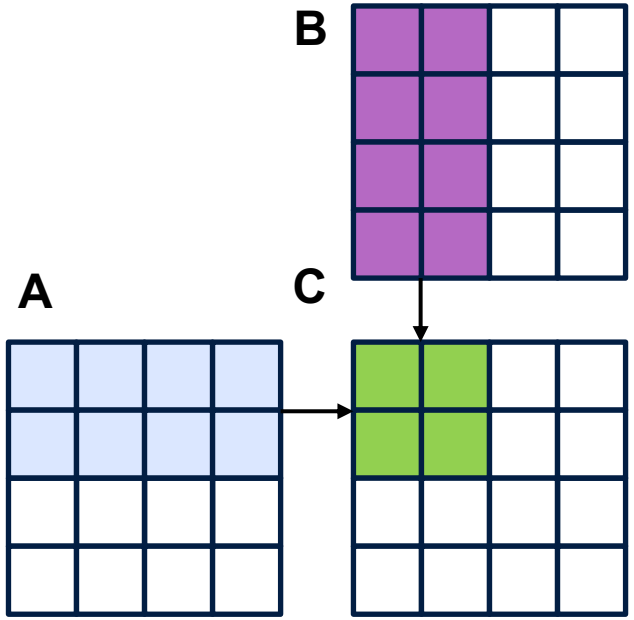


Matmul Operator

- **Computation Optimization: Tiling**
  - Motivation: Maximize GPU compute utilization
  - Method: Increasing tile size.

Example: 1 thread → 2 rows of A & 2 cols of B
→ 4 elements in C

$$C^{4\times4} = A^{4\times4} \times B^{4\times4}$$



Tiling raises Comp-to-Mem Ratio to **0.5FLOP/Byte**

⚠ Oversized tiles reduce parallelism. The key is balance.

Total compute: 128FLOP，Total access reduces to4(2*4*4*4)/2=256Byte

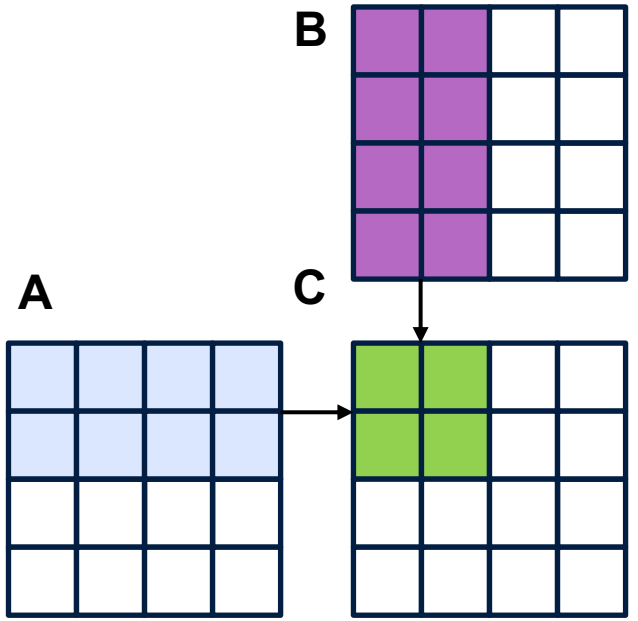
Matmul Operator

- **Computation Optimization: Tiling**

Operator Implementation:

```
__global__ void MatMulTiling(int *A, int *B, int *C, int width) {
    //eg. TILE_WIDTH=2
    int tx = threadIdx.x;  int ty = threadIdy.y;
    for (int i = 0; i < TILE_WIDTH; i++) {
        for (int j = 0; j < TILE_WIDTH; j++) {
            int row = ty * TILE_WIDTH + i;
            int col = tx * TILE_WIDTH + j;

            int sum = 0;
            for (int k = 0; k < width; k++) {
                sum += A[row * width + k] * B[k * width + col];
            } //Read A and B from global memory and compute
            C[row * width + col] = sum;
        }
    } //Each thread processes 2 rows of A and 2 columns of B
}
```

$$C^{4\times4} = A^{4\times4} \times B^{4\times4}$$

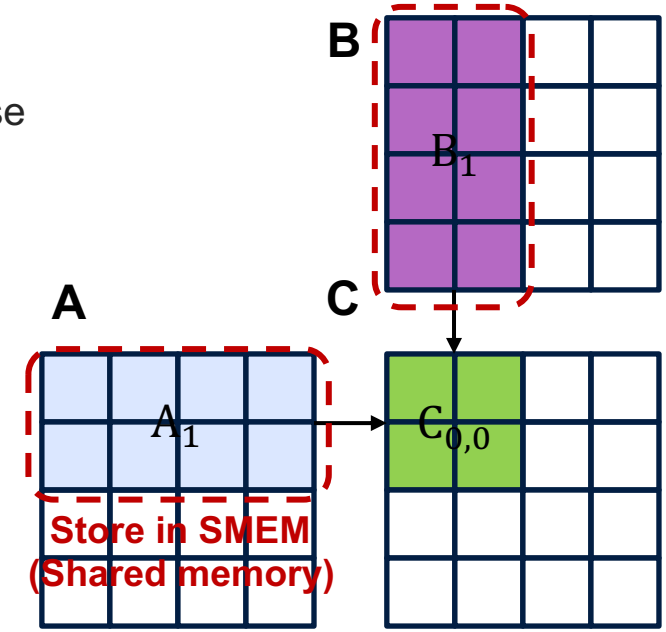


Matmul Operator

- **Memory Optimization: Using Shared Memory**
  - Motivation: Enable data sharing and fast communication between threads
  - Method: Load A and B into shared memory for reuse

→ reduce global memory access

$$C^{4\times4} = A^{4\times4} \times B^{4\times4}$$



Comp-to-Mem Ratio reaches **1FLOP/Byte**

*store A and B into shared memory, total memory access reduces to 128Byte

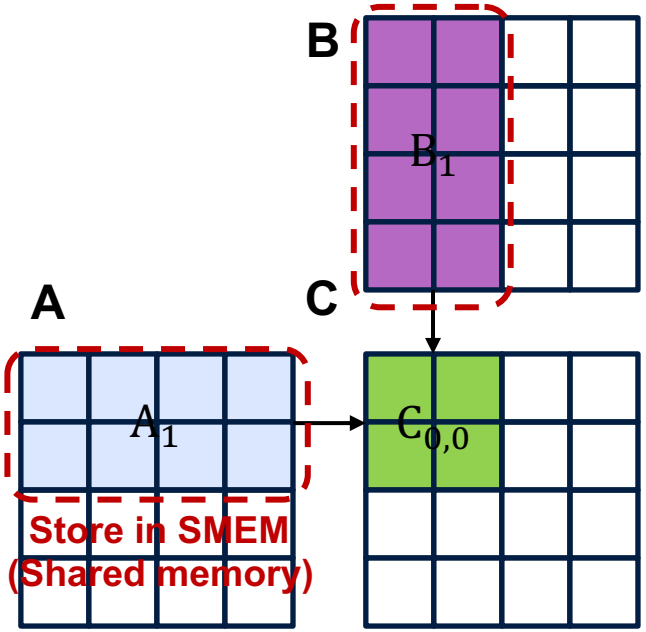
Matmul Operator

- **Memory Optimization: Using Shared Memory**

Operator Implementation:

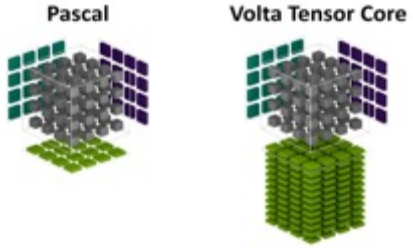
```
__global__ void MatMul_shared(int *A, int *B, int *C, int width) {
    __shared__ int sharedA[width][width]; // Declare shared memory
    __shared__ int sharedB[width][width];
    int tx = threadIdx.x;   int ty = threadIdx.y;

    sharedA[ty][tx] = A[ty * width + tx]; // Write to SMEM
    sharedB[ty][tx] = B[ty * width + tx];
    __syncthreads(); // Sync all threads
    for (int i = 0; i < TILE_WIDTH; i++) {
        for (int j = 0; j < TILE_WIDTH; j++) {
            // Read from SMEM & compute (row/col calculation are omitted)
            for (int k = 0; k < width; k++) {
             sum += sharedA[row][k] * sharedB[k][col];
            }
            C[row * width + col] = sum;
        }
    }
}
```

$$C^{4\times4} = A^{4\times4} \times B^{4\times4}$$



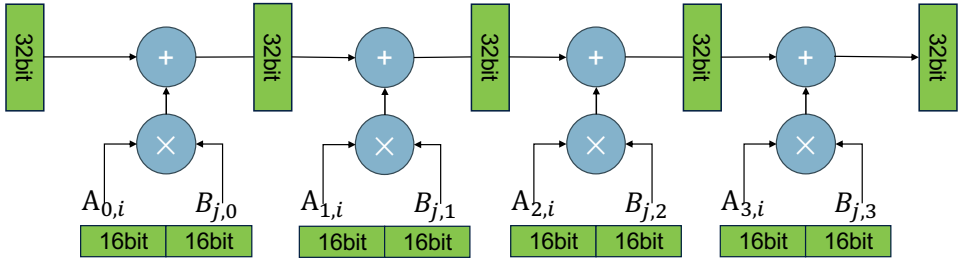
Matmul Operator

- **Special Hardware: using Tensor Core**
  - Why: CUDA Cores bottleneck at large-scale matrix ops in deep learning
  - **Tensor Core:**
    - NVIDIA's dedicated DL cores.
    - First introduced in Volta (2017)



The V100 GPU uses Tensor Cores achieves **2× speedup** over the P100[1]

  - Input: 16bit → Multiply-accumulate → Stored in 32-bit registers



[1] https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

Matmul Operator

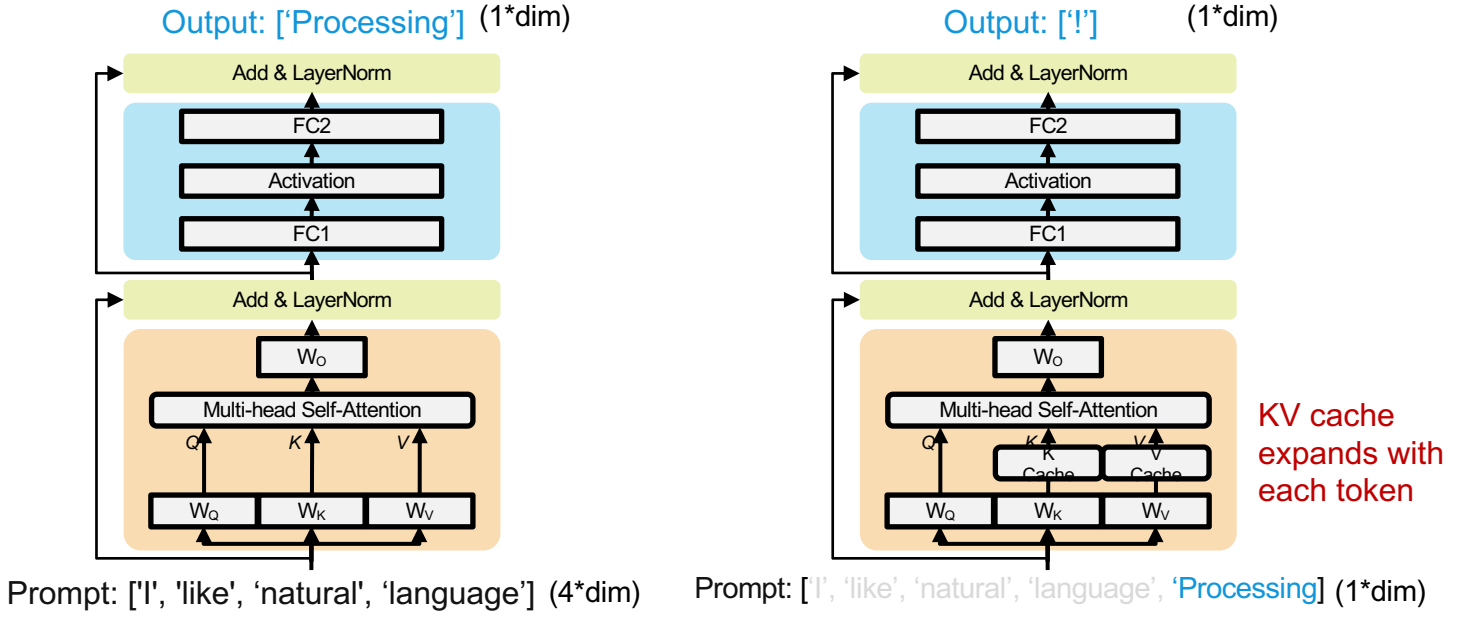
- **Special Hardware: using Tensor Core**
  - CUDA provides mma.h and wmma API for Tensor Core operations

Pesudo code：

```
#include <mma.h>
__global__ void MatMul_mma(half *a, half *b, float *c, int M, int N, int K) {
   // Declare fragments (16x16x16 tile size)
   wmma::fragment<wmma::matrix_a, 16, 16, 16, half, wmma::row_major> a_frag;
   wmma::fragment<wmma::matrix_b, 16, 16, 16, half, wmma::col_major> b_frag;
   wmma::fragment<wmma::accumulator, 16, 16, 16, float> c_frag;

   wmma::fill_fragment(c_frag, 0.0f); // Init accumulator
   wmma::load_matrix_sync(a_frag, a, K);
   wmma::load_matrix_sync(b_frag, b, N);
   wmma::mma_sync(c_frag, a_frag, b_frag, c_frag); // MM on Tensor Core

   wmma::store_matrix_sync(c, c_frag, N, wmma::mem_row_major); // Store result
}
```

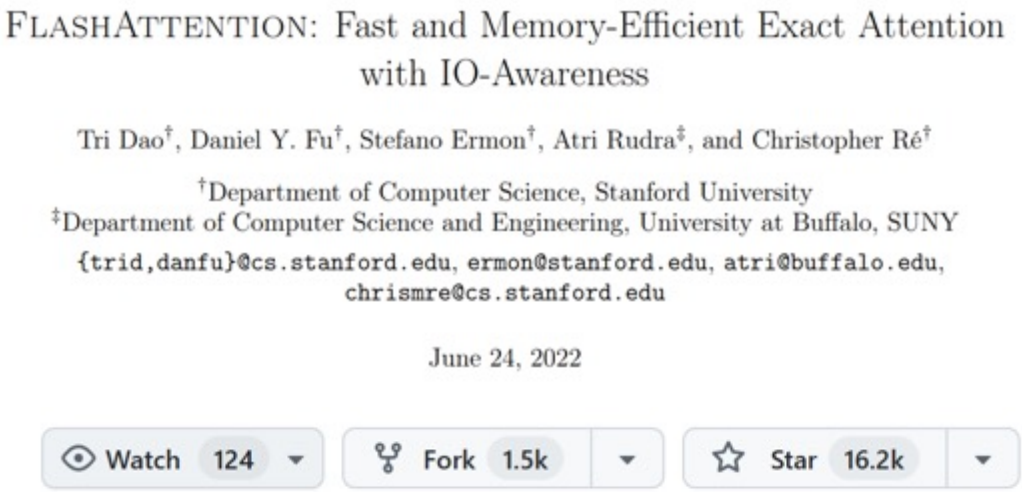
Decoder-only LLM Inference

- **Decoder-only LLM Inference：Two distinct phases**
  - Prefill phase: Handling input prompts, saving generated K and V in KV Cache
  - Decode phase: Update and use KV cache for computing Attention to generate a new token



Attention Operator

- **Prefill Optimization: FlashAttention[1,2,3]**
  - FlashAttention optimizes attention computation in prefill phase
  - One of the most widely adopted acceleration methods with 16.2k GitHub stars[4]



FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

[†]Department of Computer Science, Stanford University
[‡]Department of Computer Science and Engineering, University at Buffalo, SUNY

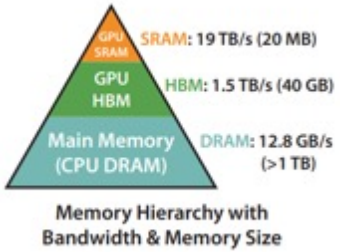
{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

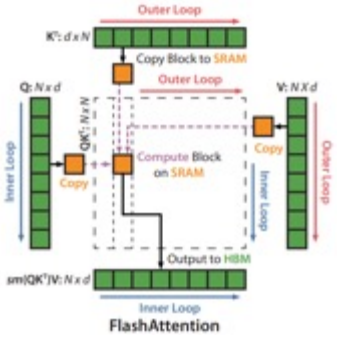
June 24, 2022

[1] Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." *Advances in neural information processing systems* 35 (2022): 16344-16359.
[2] Dao, Tri. "Flashattention-2: Faster attention with better parallelism and work partitioning." *arXiv preprint arXiv:2307.08691* (2023).
[3] Shah, Jay, et al. "Flashattention-3: Fast and accurate attention with asynchrony and low-precision." *Advances in Neural Information Processing Systems* 37 (2024): 68658-68685.
[4] https://github.com/Dao-AILab/flash-attention

Attention Operator

- **Prefill Optimization: FlashAttention[1,2,3]**
  - Why：Complex Attention I/O; Large activation memory
  - How：Operator fusion, including fwd and bwd
  - Results：2-4x speedup; memory: $O(N^2) \rightarrow O(N)$





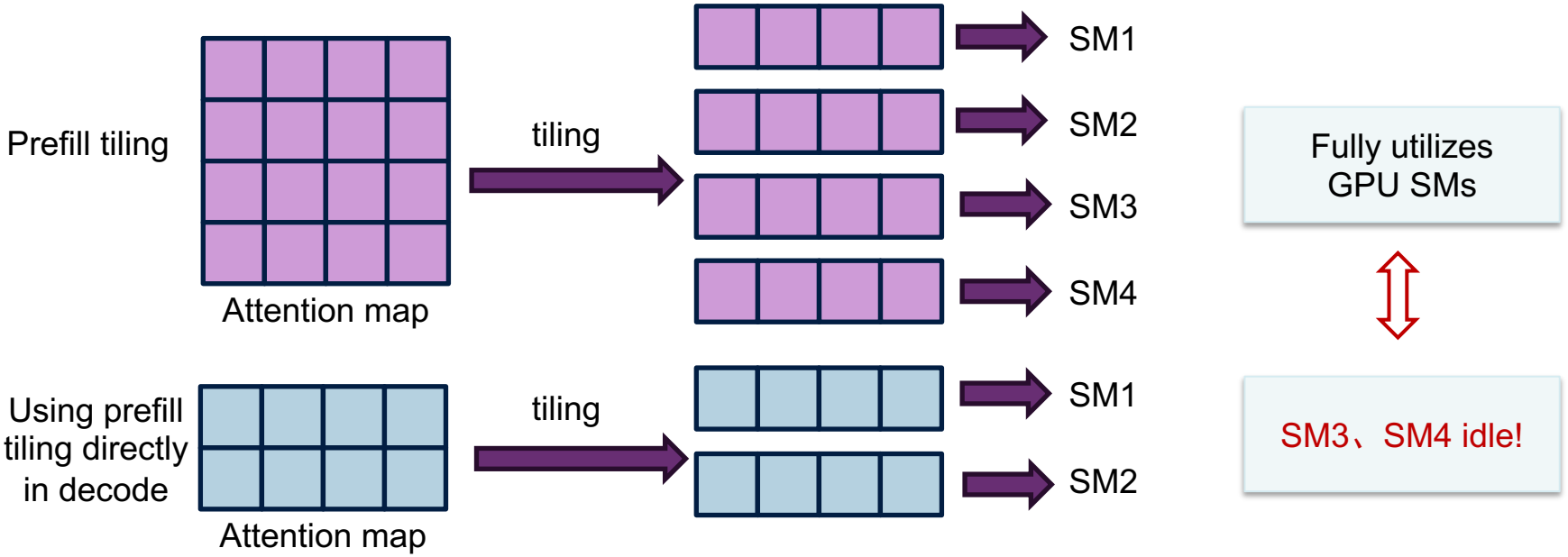
Saves I/O & memory via operator fusion

Tiling strategies differ between fwd/bwd passes

[1] Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness." *Advances in neural information processing systems* 35 (2022): 16344-16359.
[2] Dao, Tri. "Flashattention-2: Faster attention with better parallelism and work partitioning." *arXiv preprint arXiv:2307.08691* (2023).
[3] Shah, Jay, et al. "Flashattention-3: Fast and accurate attention with asynchrony and low-precision." *Advances in Neural Information Processing Systems* 37 (2024): 68658-68685.

Attention Operator

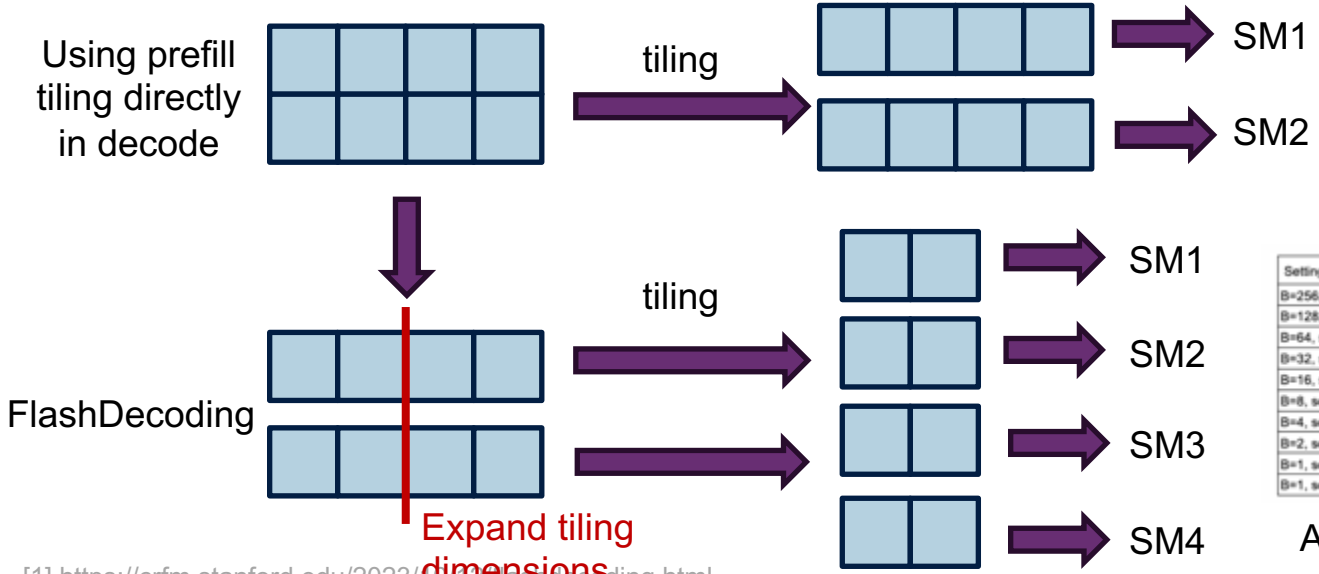
- **Decode Optimization: FlashDecoding[1]**
  - Why: only 1 token/step in decode → using prefill tiling strategy directly causes low GPU utilization



[1] https://crfm.stanford.edu/2023/10/12/flashdecoding.html

Attention Operator

- **Decode Optimization: FlashDecoding[1]**
  - Method: Increases matrix tile count to boost SM utilization, achieving higher GPU efficiency



Setting \ Algorithm	PyTorch Eager	Flash-Attention v2.0.9	Flash-Decoding
B=256, seqlen=256	3058.6	390.5	63.4
B=128, seqlen=512	3151.4	366.3	67.7
B=64, seqlen=1024	3160.4	364.8	77.7
B=32, seqlen=2048	3158.3	352	58.5
B=16, seqlen=4096	3157	401.7	57
B=8, seqlen=8192	3173.1	529.2	56.4
B=4, seqlen=16384	3223	582.7	58.2
B=2, seqlen=32768	3224.1	1156.1	60.3
B=1, seqlen=65536	1335.6	2300.6	64.4
B=1, seqlen=131072	2664	4592.2	106.6

Achieves 5-10x compute speedup

[1] https://crfm.stanford.edu/2023/10/12/flashdecoding.html

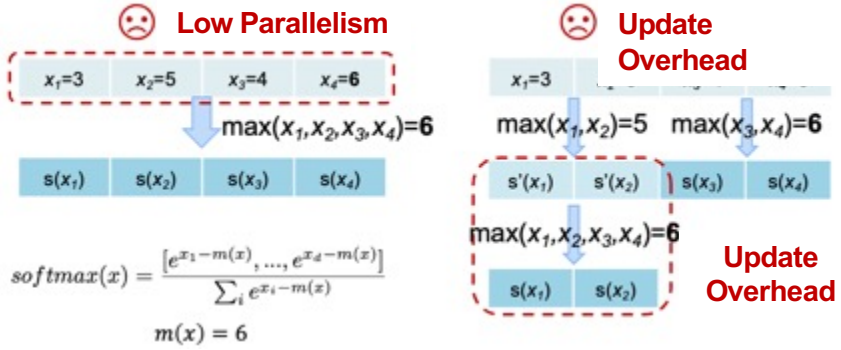
FlashDecoding++

Motivation

Attention & Matmul operators still optimizable in LLM inference

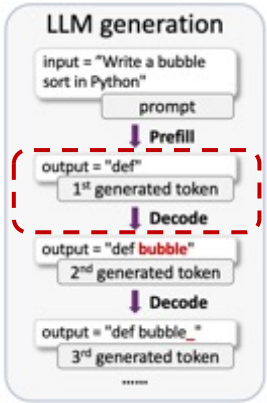
Attention（FlashAttention[1]）

Online Softmax → Global reduction causes significant update overhead

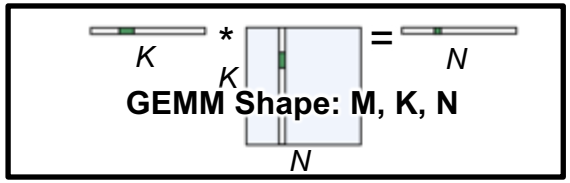


Matmul（cuBLAS[2], CUTLASS[3]）

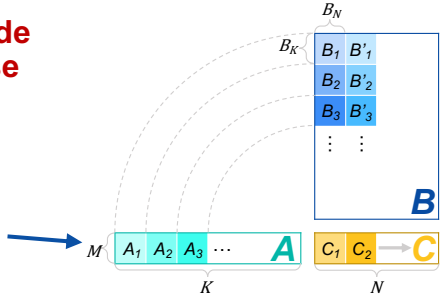
Inefficient tiling for “short-wide” matrices in decode phase



***M* dimension = batch size**



Decode phase



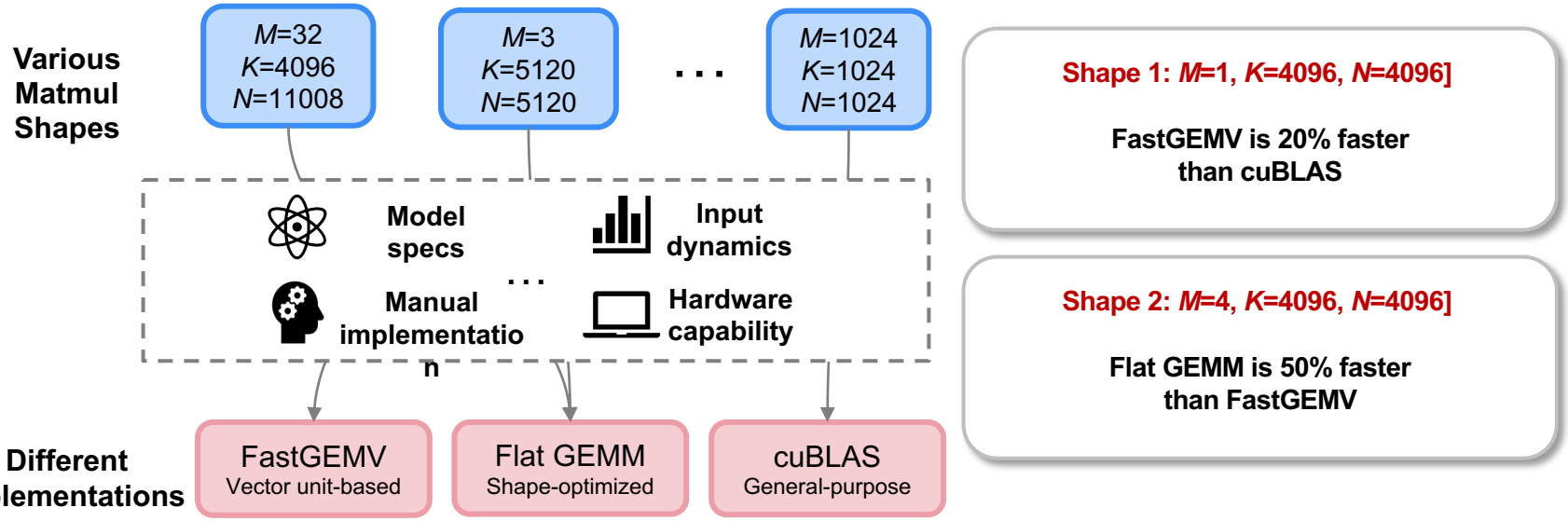
[1] Dao T, Fu D, Ermon S, et al. Flashattention: Fast and memory-efficient exact attention with io-awareness[J]. Advances in neural information processing systems, 2022, 35: 16344-16359.
[2] https://developer.nvidia.com/cublas
[3] https://github.com/NVIDIA/cutlass

FlashDecoding++

Motivation

Suboptimal implementation against various Matmul shapes

Wrong low-level implementation choice → Significant performance loss



[1] https://github.com/wangsiping97/FastGEMV
[2] https://developer.nvidia.com/cublas
[3] https://github.com/NVIDIA/cutlass

FlashDecoding++

Method

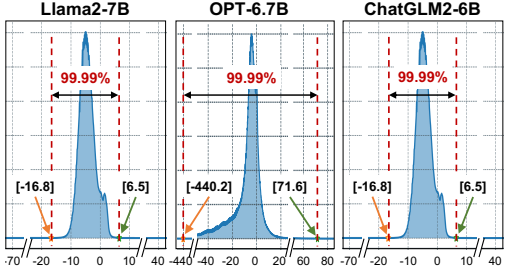
Leverage value distribution in LLM inference to optimize Attention operator[1]

Mathematical Equivalence & Numerical Feasibility

$$softmax(x) = \frac{[e^{x_1-m(x)}, ..., e^{x_d-m(x)}]}{\sum_i e^{x_i-m(x)}} = \frac{[e^{x_1-\phi}, ..., e^{x_d-\phi}]}{\sum_i e^{x_i-\phi}} \quad \forall \phi \in \mathbb{R}$$

Mathematical Equivalence

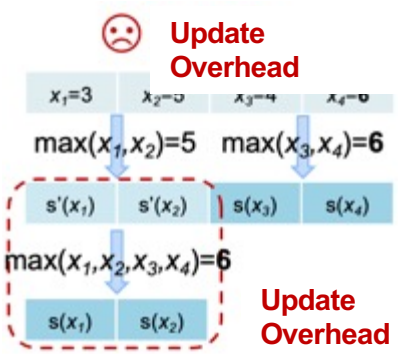
Arbitrary max value
Precision loss only from data type overflow



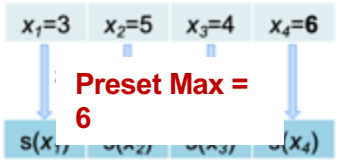
Numerical Feasibility

Data shows no FP32 overflow in 99.99% cases

Remove Update Overhead via Preset Max



Online softmax
Parallel but large update overhead



Async softmax
Parallel with zero update overhead

[1] Ke, Hong, et al. "FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics." Proceedings of Machine Learning and Systems. 2024.

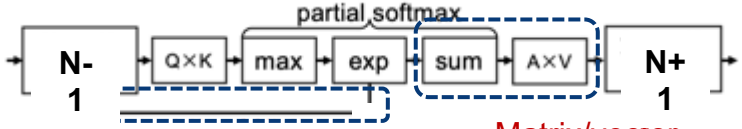
FlashDecoding++

Method

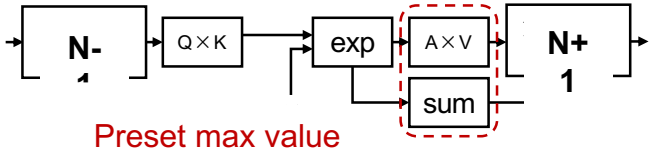
Leverage value distribution in LLM inference to optimize Attention operator[1]

Async Pipeline for Acceleration

Remove update overhead via **preset max value**



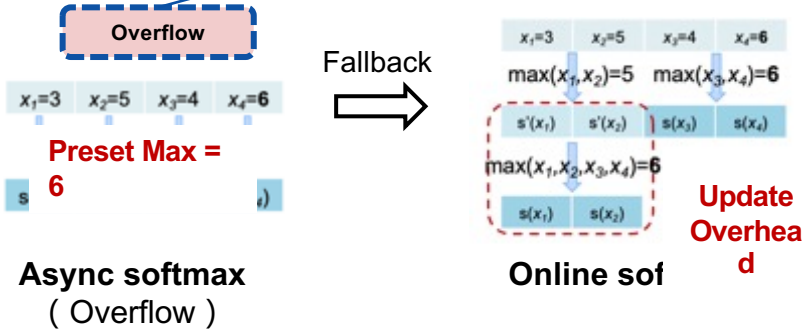
Async Pipeline: Overlap matrix & vector unit usage



Recomputation for Correctness

Fallback to online softmax on overflow detection

Overflow check: Set threshold via preset max → threshold detection



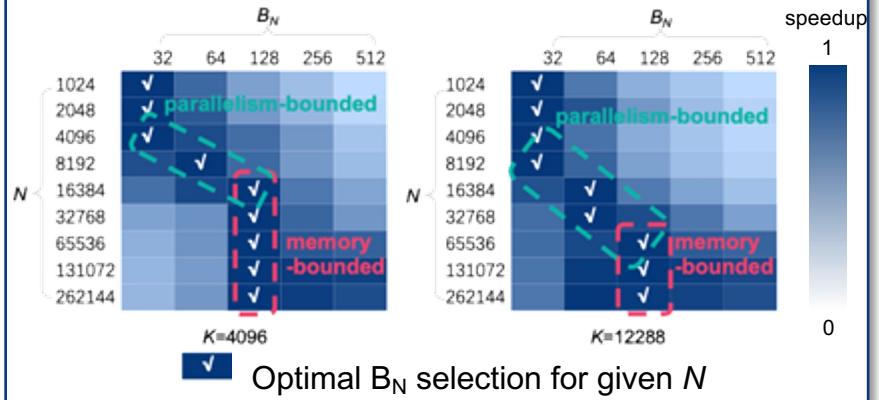
[1] Ke, Hong, et al. "FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics." Proceedings of Machine Learning and Systems. 2024.

FlashDecoding++

Method

Leverage typical shapes in LLM inference to optimize Matmul operator[1]

Short-wide Matmul Bottleneck Analysis

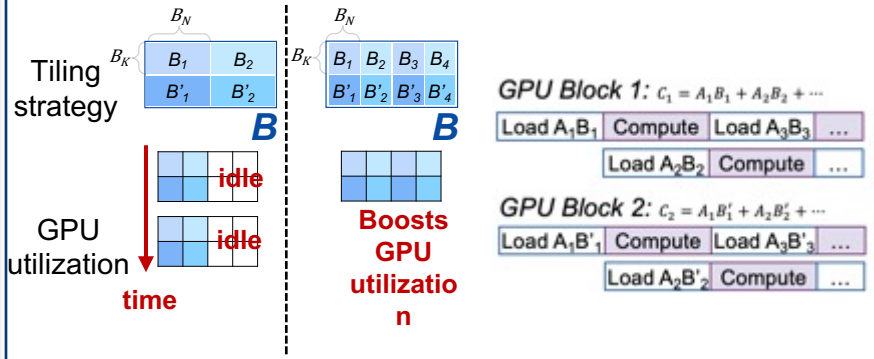


***N* is large: Low Parallelism**
Need more tiles

***N* is large: Memory-bound**
Low Comp-to-mem ratio

Adaptive Tiling + Double Buffering

Adaptive tiling & double buffering for different shapes



***N* is large: Low Parallelism**
Fine-grained tiling on *N*-dim for sufficient parallelism

***N* is large: Memory-bound**
Double buffering to hide access latency

[1] Ke, Hong, et al. "FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics." Proceedings of Machine Learning and Systems. 2024.

FlashDecoding++

Method

Leverage Matmul shape patterns in LLM inference for dynamic implementation selection[1]

Shape Variation Patterns

4 shape categories in inference with only M dimension varies

Observation 1:
Fixed [*N*, *K*] → only 4 combinations per model

Observation 2:
Only M dimension varies with input
Prefill: M=total token count
Decode: M=batch size

	Operation	M	N	K
Prefill phase	K, Q, V projection	SeqLen*B	HD*3	HD
	O projection	SeqLen*B	HD	HD
	FFN1	SeqLen*B	FD	HD
	FFN2	SeqLen*B	HD	FD
Decode phase	K, Q, V projection	B	HD*3	HD
	O projection	B	HD	HD
	FFN1	B	FD	HD
	FFN2	B	HD	FD

HD: Hidden dimension size
FD: Dimension size after the first FFN
B: Batch size
SeqLen: Input sequence length

Only 4 shape categories!

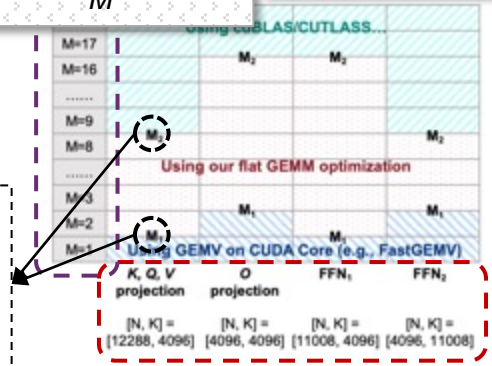
Offline Table building & Online heuristic selection

1:GEMV Vector-unit based

2: Flat GEMM Optimized for small *M*

3: cuBLAS General purpose

Rows: varying *M* dim



For each [*N*, *K*]
Offline: fine 2 critical points per [*N*, *K*]
Online: select based on *M* value

Columns: [*N*, *K*] combinations

[1] Ke, Hong, et al. "FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics." Proceedings of Machine Learning and Systems. 2024.

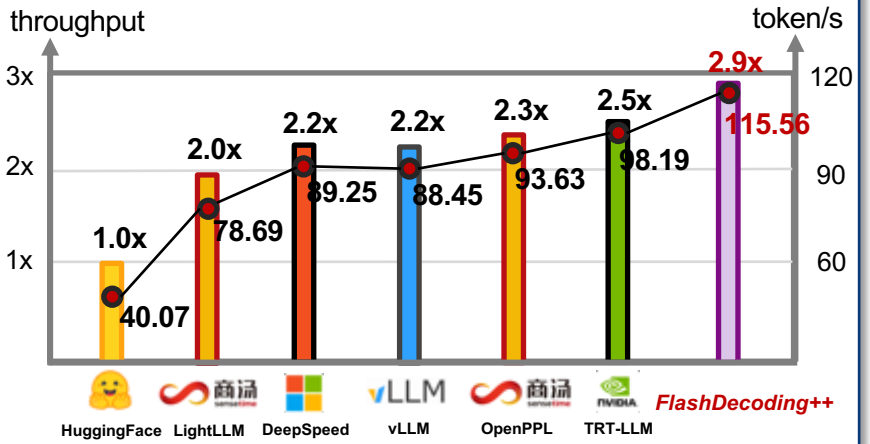
FlashDecoding++

Method

Throughput surpasses SOTA by **over 10%**
1.88x faster v.s. HuggingFace on average

Throughput Comparison on NV A100 GPU

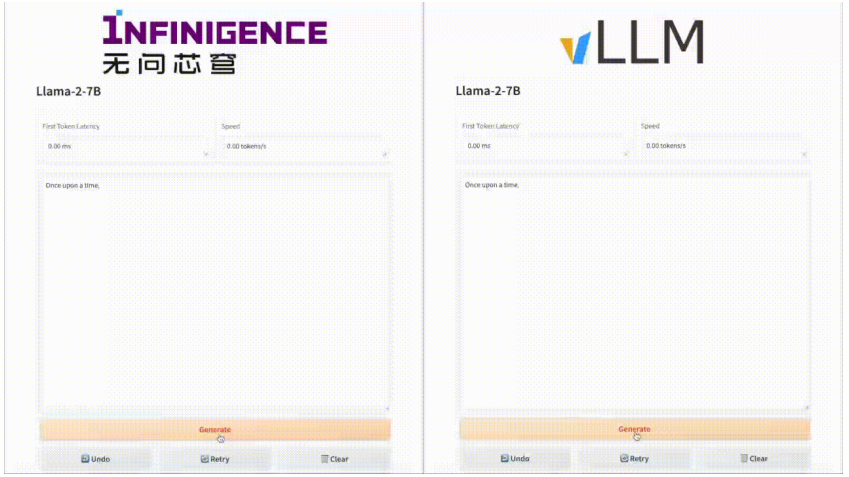
1.24x/1.13x faster v.s. vLLM/ TensorRT-LLM



*Test setup: L Llama2-7B, bs=1, 128 I/O tokens, single A100 GPU

Comparsion on AMD MI210 GPU

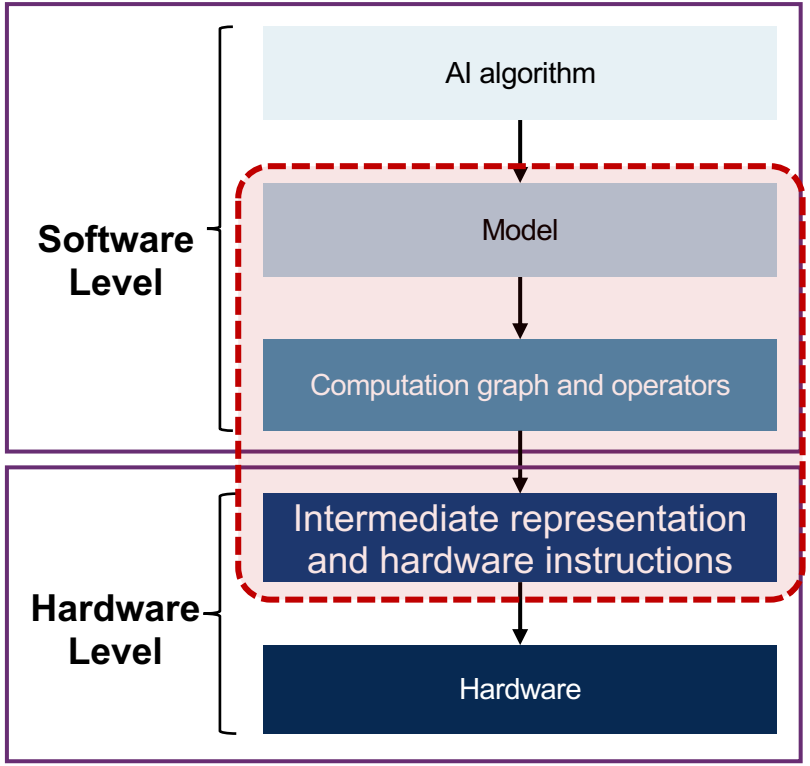
1.86x faster vs vLLM on average



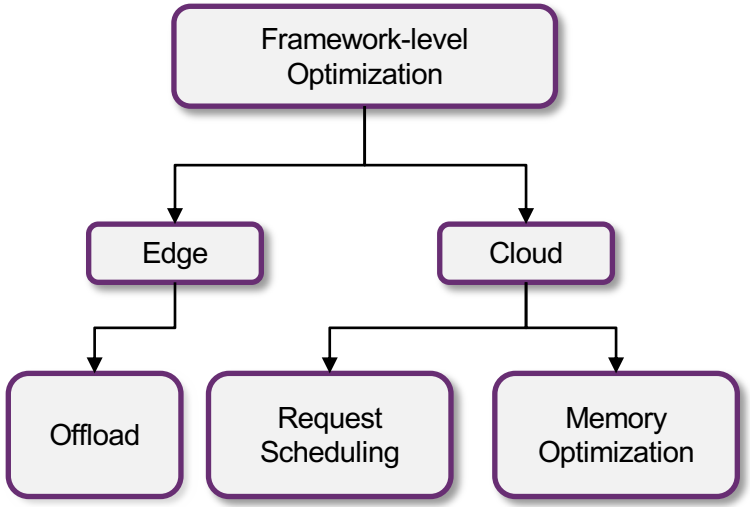
[1] Ke, Hong, et al. “FlashDecoding++: Faster Large Language Model Inference with Asynchronization, Flat GEMM Optimization, and Heuristics.” Proceedings of Machine Learning and Systems. 2024.

Design Space

- **System Design: Framework-level Optimization**



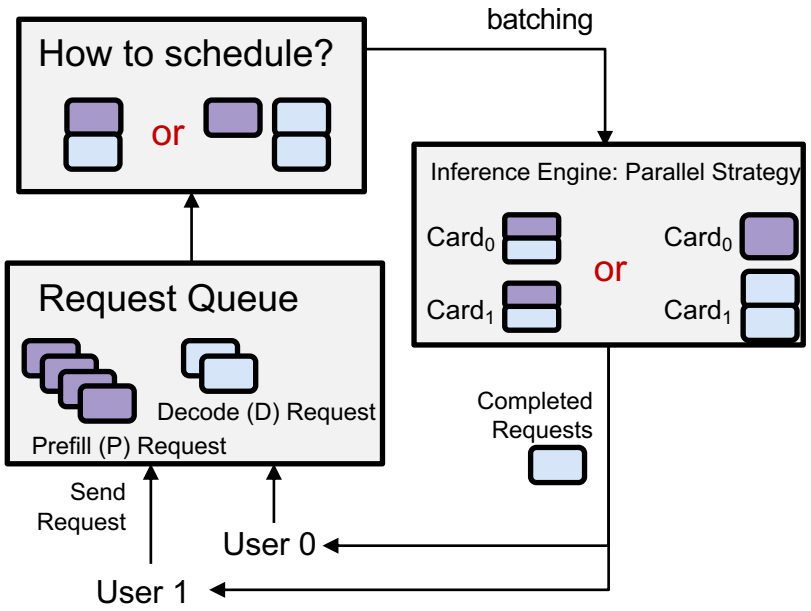
$$\max. \mathit{throughput},\\ \text{s.t.}\ \mathit{latency} \le \mathit{SLO}.$$



Framework-level Optimization

Method

Optimizing the system throughput
adhering to the service-level objective (SLO)



vLLM

(SOSP 23)

Paged KV cache
Basic memory management for LLM serving
2-4x throughput improvement

Orca

(OSDI 22)

Continuous batching
Basic scheduling method for LLM inference serving
~10x throughput improvement

Sarathi-Serve

(OSDI 24)

Mixed P/D request batching
Basic batching method for **co-located systems**
2x throughput improvement

DistServe

(OSDI 24)

P/D disaggregated system
Specialized optimization for P/D instances
Improves **effective throughput by 4.48x**

SGLang

(NeurIPS 24)

Prefix caching technique
Reuses repeated **KV cache** across requests
6x throughput improvement

Mooncake

(FAST 25 Best Paper)

KV cache-centric scheduling and storage strategy based on a **disaggregated** system design

Offloading Techniques

- **Offloading Technique: Definition and Motivation**
  - Definition: Offloading parts of the model (model weights, KV cache, etc.) from GPU to other devices (e.g., CPU) for storage or even computation, to save space and improve computational efficiency
  - Motivation: The large volume of model parameters and KV cache data exceeds the storage capacity of GPU memory

Table 1: LLaMA2-13B, KV Cache size with context length

Context length	10k	100k	500k	1000k
KV Cache size	8.19GB	81.9GB	409.6GB	819.2GB
Misc size	26GB	26GB	26GB	26GB

LLaMA2-13B: when the context length reaches 100k, the required KV cache reaches nearly 82GB, exceeding the memory capacity of a single GPU

Table 3: Model parameter counting at layer-level (dtype: BF/FP16)

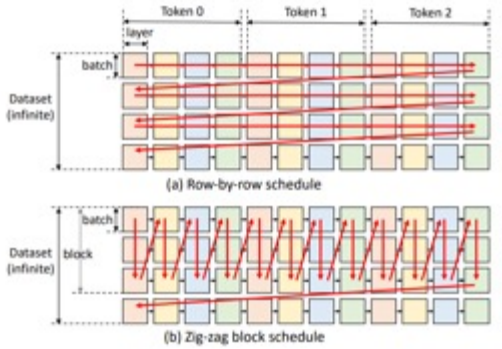
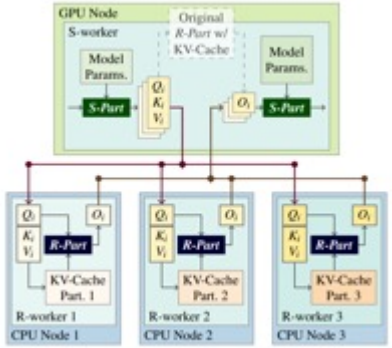
Layers	Modules	Shapes	No. Parameters	Per Layer	MB	GB
Layer 0	Embedding	[129280, 7168]	926,679,040	1.5 B	2880	2.8
	MLA	-	187,107,328			
	MLP	3 * [7168, 18432]	396,361,728			
	LN	2*7168+1536+512	16,384			
Layers 1 - 2	MLA	-	187,107,328	0.58 B	1112	1.1
	MLP	3 * [7168, 18432]	396,361,728			
	LN	2*7168+1536+512	16,384			
Layers 3 - 59	MLA	-	187,107,328	11.5 B	21950	21.44
	Gate	[256, 7168]	1,835,008			
	MoE	3 * [7168, 2048] * 257	11,318,329,344			
	LN	2*7168+1536+512	16,384			
Layer 60	MLA	-	187,107,328	12.4 B	23712	23.16
	Gate	[256, 7168]	1,835,008			
	MoE	3 * [7168, 2048] * 257	11,318,329,344			
	LN	2*7168+1536+512	16,384			
	Head	[7168, 129280]	926,679,040			
Total				671 B	1,280,000	1250

DeepSeek-V3: parameter size reaches 1250GB

[1] Memory Analysis on the Training Course of DeepSeek Models, Zhang et al. Arxiv Preprint 2502.07486.

Offloading Techniques

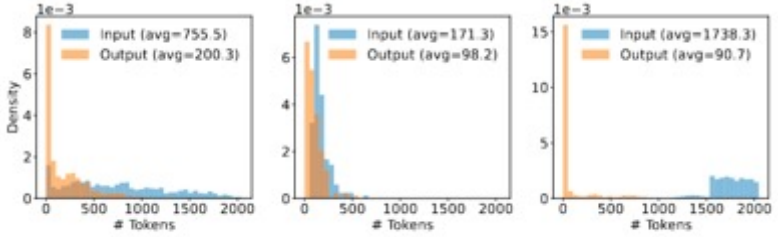
- **Offloading Techniques: Categories**

Offloading Model Weights	Offloading KV Cache	Offloading Experts
Representative Work: FlexGen[1]	Representative Work: FastDecode[2]	Representative Work: KTransformers[3]
		
Adopt the Zig-zag computation order to hide the communication time of weight transmission	Offload KV Cache to CPUs to relieve the storage pressure on GPUs during the decode phase	KTransformers offloads experts to CPUs (detailed in the next page)

[1] FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU, Sheng et al. Arxiv Preprint 2303.06865.
[2] FastDecode: High-Throughput GPU-Efficient LLM Serving using Heterogeneous Pipelines, He et al. Arxiv Preprint 2403.11421.
[3] https://github.com/kvcache-ai/ktransformers

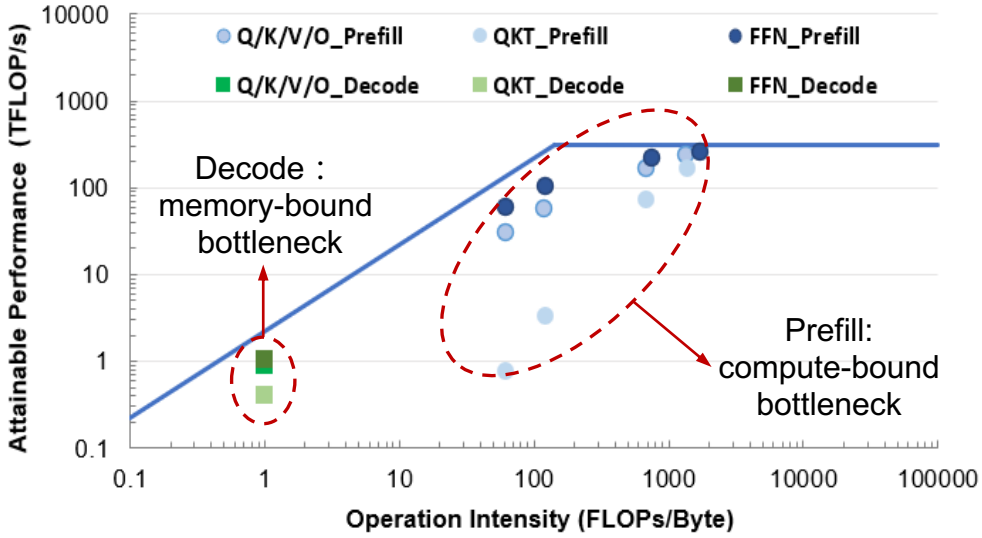
Request Scheduling

- **Request Scheduling**
  - Definition: Scheduling requests in different phases to GPU instances for batched computation
  - Motivation: Requests for LLM inference have varying lengths and distinct phases (Prefill and Decode phases), and schedu



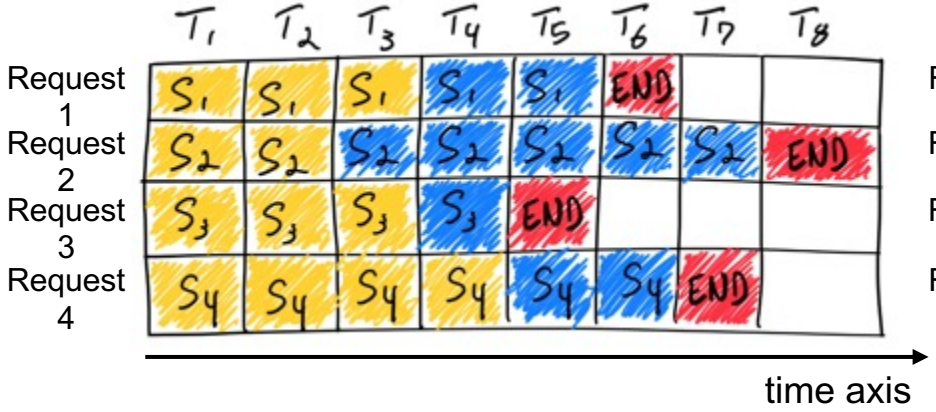
Statistical distribution of request lengths across different datasets

How to perform batch processing for requests with varying lengths?



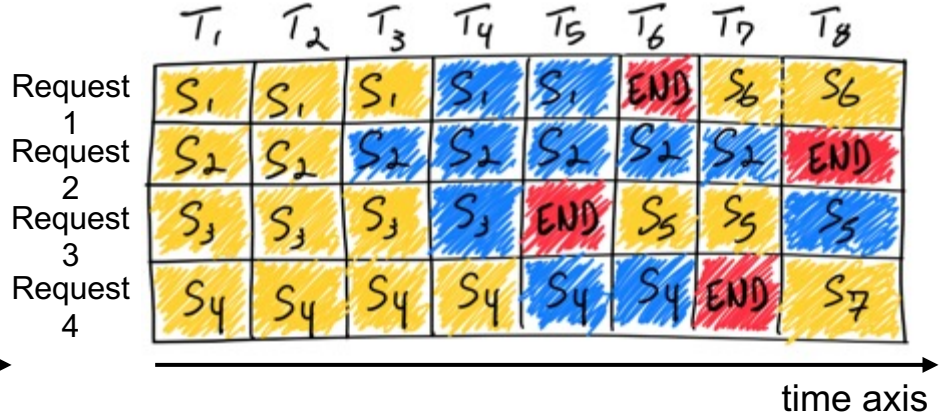
Request Scheduling

- **Request Scheduling: Batching**
  - Orca (G. Yu et al, OSDI'22) proposes the continuous batching technique, which batches requests with varying lengths at the **TOKEN** granularity. Compared to request-level batching, it improves throughput by 36.9x



Request-level batching

The processing time depends on the longest request, leading to **low utilization**



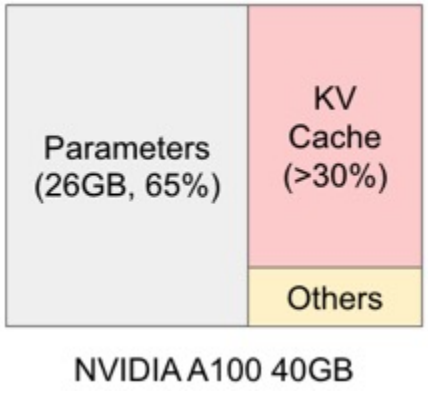
Token-level batching (Continuous batching)

Batching requests of different lengths by concatenating them in the Token dimension

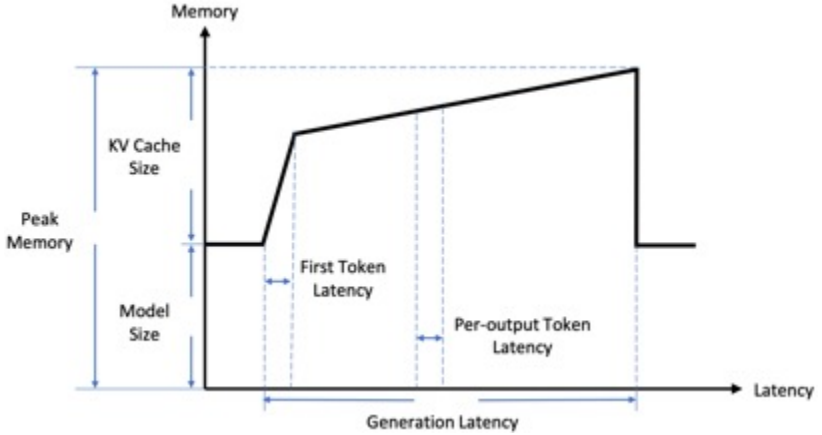
[1] G. Yu, et al. "ORCA: A Distributed Serving System for Transformer-Based Generative Models.", OSDI, 2022.

Memory Optimization

- **Memory Optimization: PagedAttention**
  - The growth of KV cache and memory fragmentation limit concurrency, resulting suboptimal system throughput.
  - vLLM (W. Kwon et al, SOSP'23) proposes PagedAttention,which stores KV cache in a paged manner. This approach effectively eliminates memory fragmentation, and improves throughput by 2–4 times.



KV cache accounts for a large proportion in distributed systems.

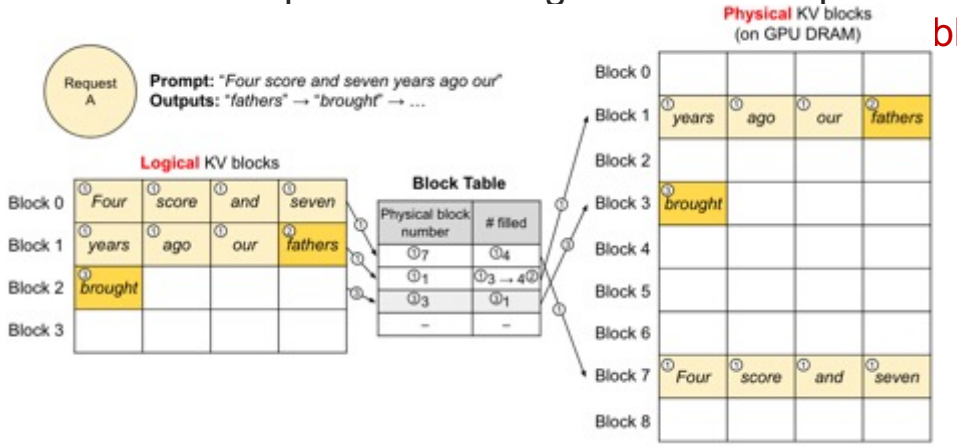


KV cache in LLM inference grows with the generation process

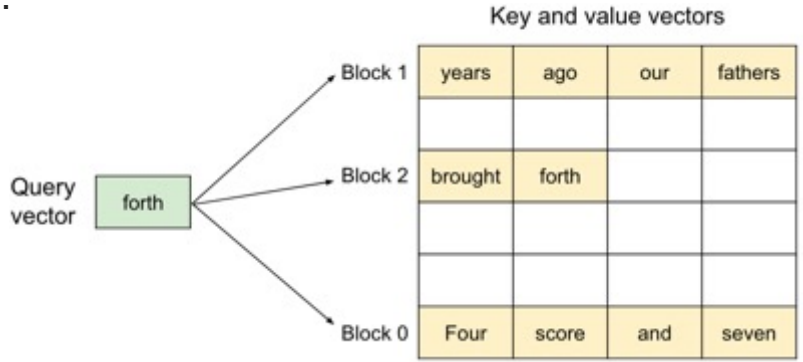
[1] W. Kwon, et al. " Efficient Memory Management for Large Language Model Serving with PagedAttention.", SOSP, 2023.

Memory Optimization

- **Memory Optimization: PagedAttention**
  - Memory: KV cache is stored in blocks along the sequence dimension at different physical addresses.
  - Computation: During attention computation, the physical addresses of the corresponding ble.



Memory: Logically continuous sequences are stored in blocks at actual physical addresses.



Computation: Index the KV cache addresses required for computation by looking up a table.

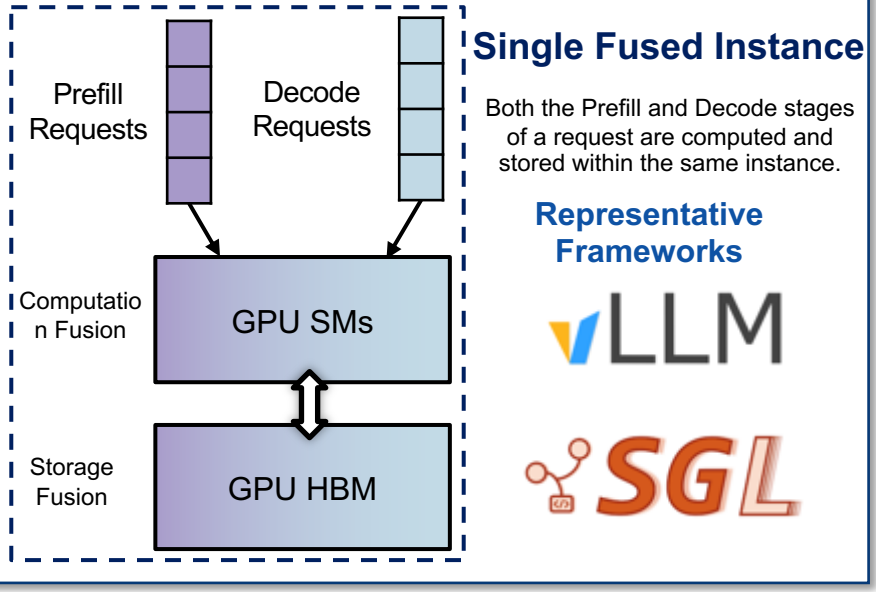
[1] W. Kwon, et al. " Efficient Memory Management for Large Language Model Serving with PagedAttention.", SOSP, 2023.

semi-PD

Key Problem

Co-located and disaggregated systems for Prefill/Decode have their own strengths and weaknesses in computation and storage.

Co-located Systems（vLLM, SGLang）



Single Fused Instance

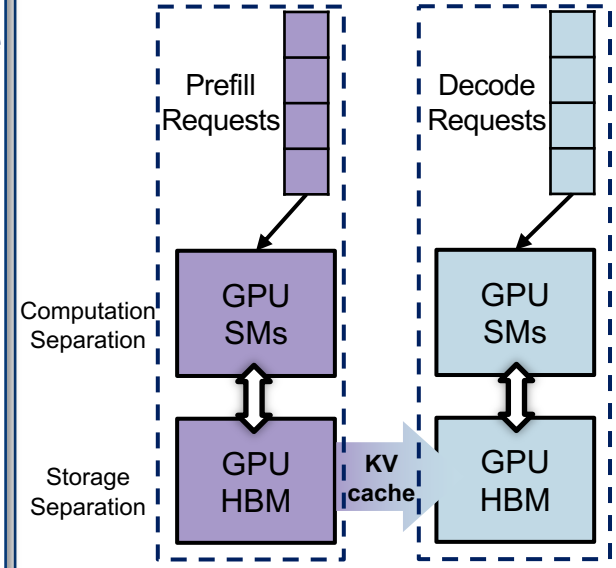
Both the Prefill and Decode stages of a request are computed and stored within the same instance.

Representative Frameworks

vLLM

SGL

Disaggregated Systems（Mooncake, DistServe）



Prefill Instance +Decode Instance

After a request is completed in the Prefill instance, the KV cache is transmitted to the Decode instance for computation.

Representative Frameworks

Mooncake

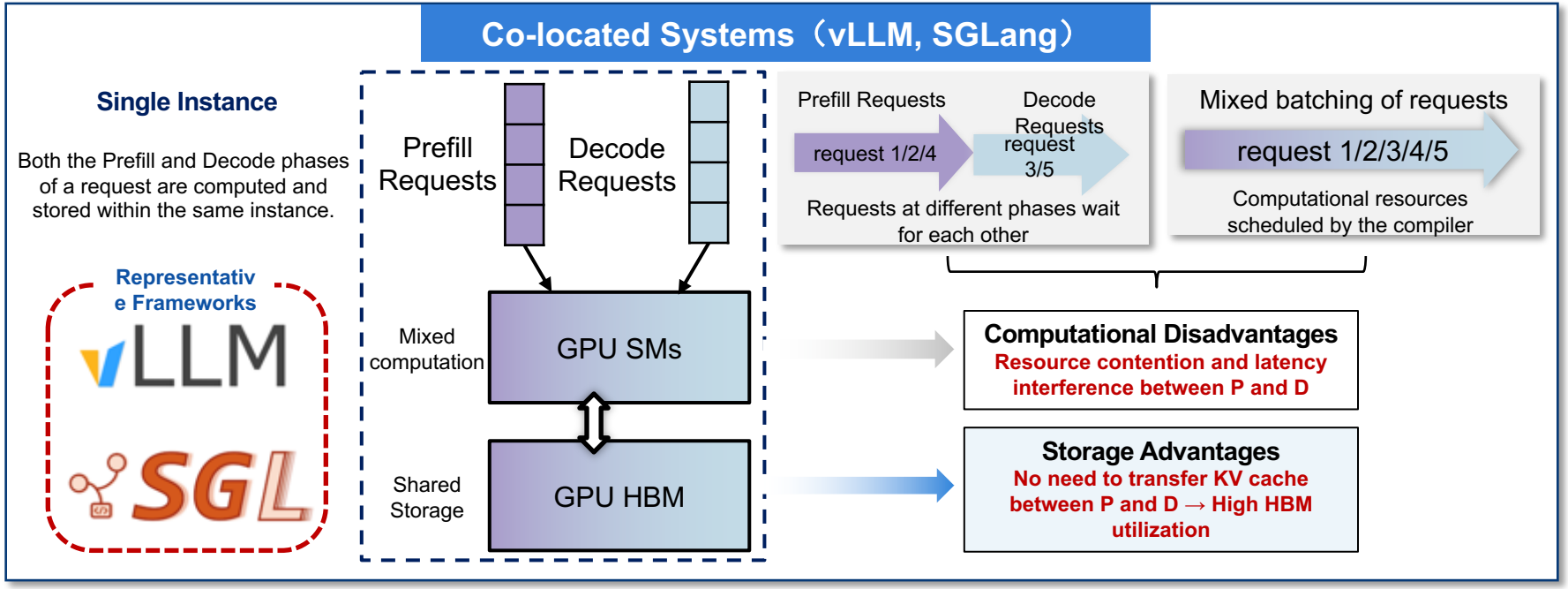
deepseek

[1] W. Kwon, et al. ” Efficient Memory Management for Large Language Model Serving with PagedAttention.”, SOSP, 2023.
[2] L. Zheng, et al. “SGLang: Efficient Execution of Structured Language Model Programs”, NeurIPS, 2024.
[3] R. Qin, et al. “Mooncake: Trading More Storage for Less Computation”, FAST, 2025.
[4] DeepSeek Team. “DeepSeek-V3 Technical Report”. arXiv, 2024.

semi-PD

Key Problem

Co-located and disaggregated systems for Prefill/Decode have their own strengths and weaknesses in computation and storage.

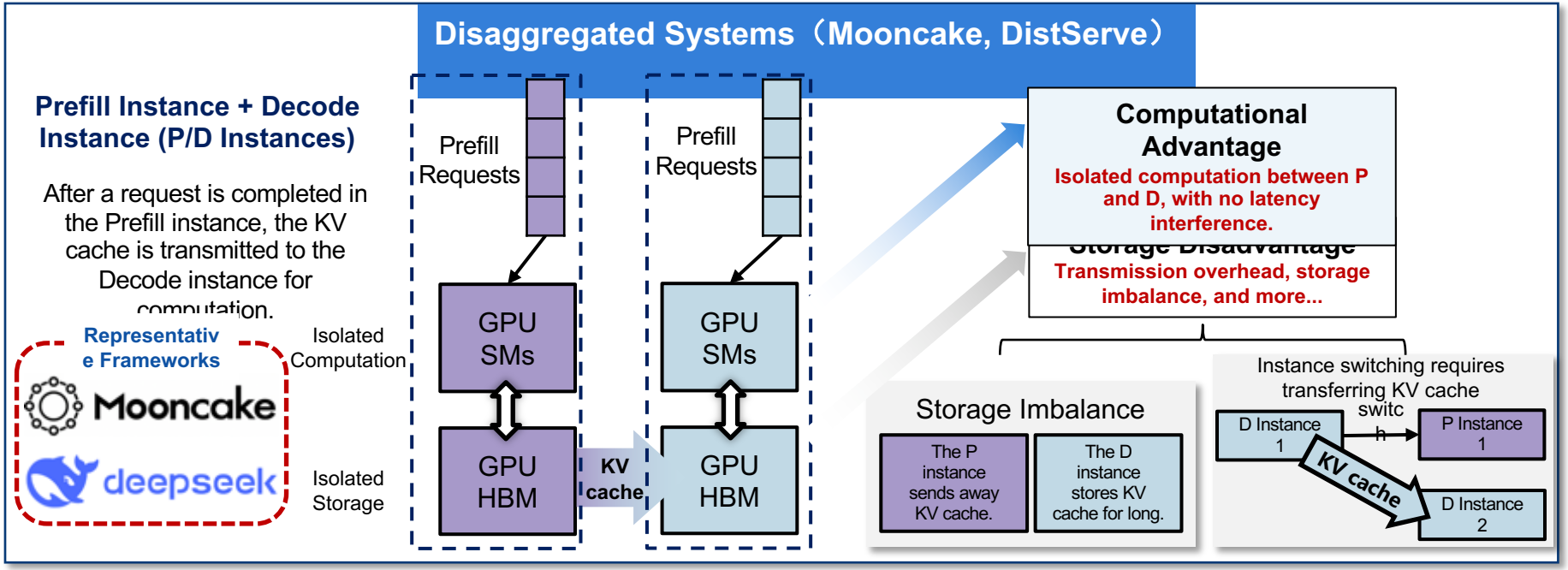


[1] W. Kwon, et al. ” Efficient Memory Management for Large Language Model Serving with PagedAttention.”, SOSP, 2023.
[2] L. Zheng, et al. “SGLang: Efficient Execution of Structured Language Model Programs”, NeurIPS, 2024.

semi-PD

Key Problem

Co-located and disaggregated systems for Prefill/Decode have their own strengths and weaknesses in computation and storage.

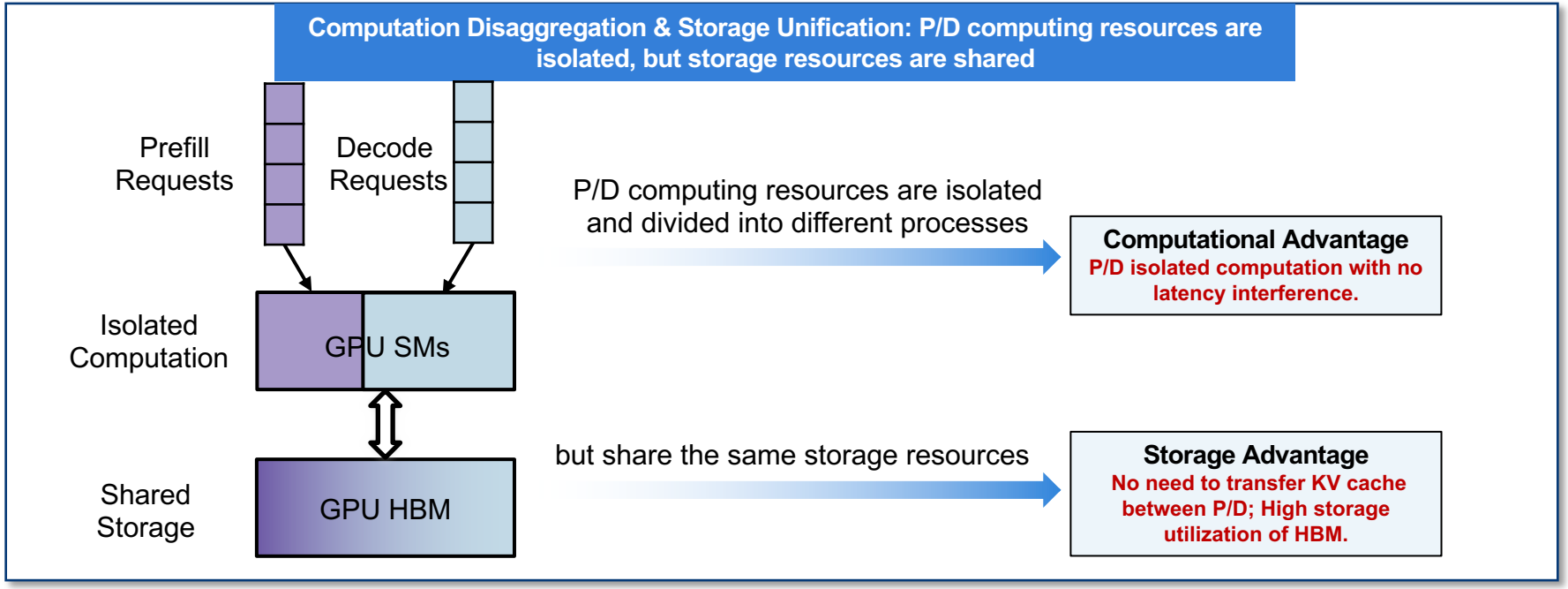


[1] R. Qin, et al. “Mooncake: Trading More Storage for Less Computation”, FAST, 2025.
[2] DeepSeek Team. “DeepSeek-V3 Technical Report”. arXiv, 2024.

semi-PD

Method

Combine the computational advantage of disaggregated systems and the storage advantage of co-located systems[1]



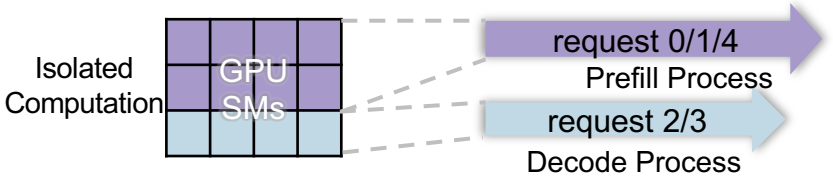
[1] Ke, Hong, et al. "semi-PD: Towards Efficient LLM Serving via Phase-wise Computation Disaggregation and Unified Storage." *arXiv preprint arXiv:2504.19867*. 2025.

semi-PD

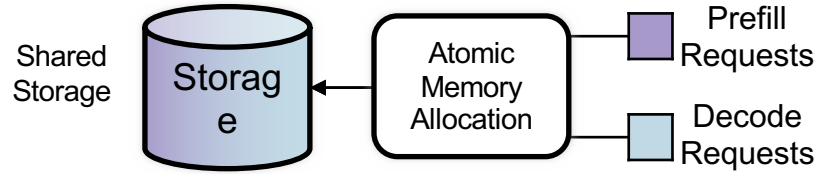
Method

Combine the computational advantage of disaggregated systems and the storage advantage of co-located systems[1]

Isolated Computation and Shared Storage via IPC



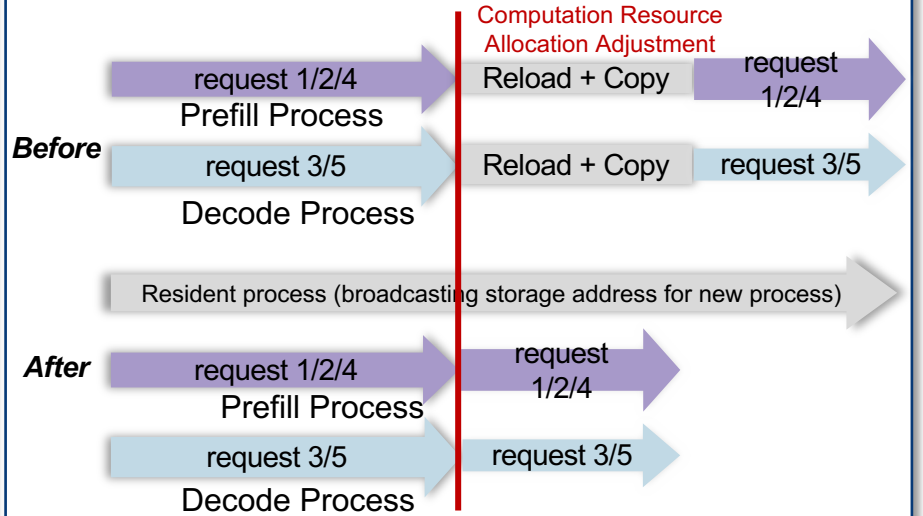
P/D uses asynchronous processes to achieve computation resource isolation at the SM level.



Atomic memory allocation is used to avoid Write-After-Read (WAR) conflicts.

Low-Overhead Resource Adjustment Mechanism

Resident processes manage weights and KV cache storage, eliminating the need for reloading and copying.



[1] Ke, Hong, et al. "semi-PD: Towards Efficient LLM Serving via Phase-wise Computation Disaggregation and Unified Storage." *arXiv preprint arXiv:2504.19867*. 2025.

semi-PD

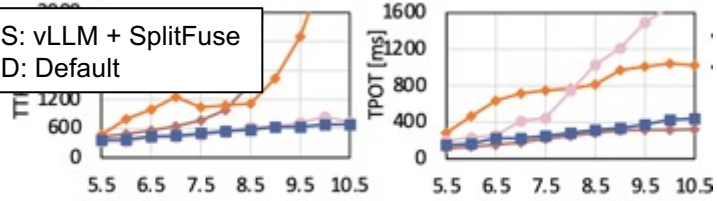
Result

Llama3 series models: 1.55-1.72x improvement in request service rate under given SLOs DeepSeek-V3 model: 1.49-2.58x reduction in latency

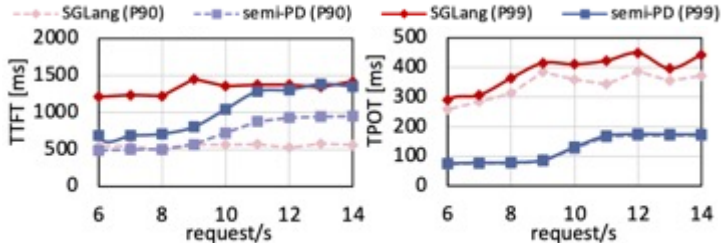
Instance-Level Inference

Lower Time To First Token (TTFT) and Time Per Output Token (TPOT).

vLLM-S: vLLM + SplitFuse
vLLM-D: Default



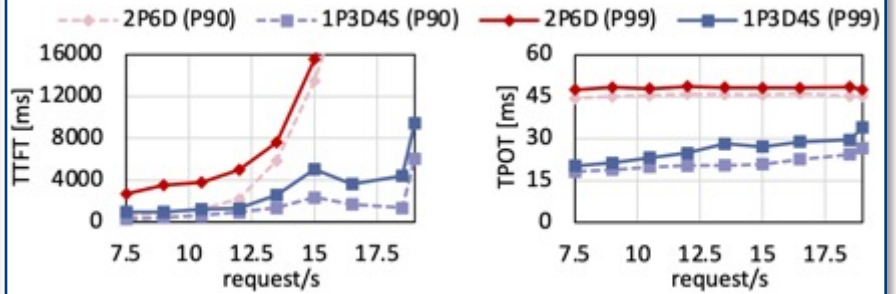
Llama3-70B, ShareGPT, FP16, 4xA100 (vLLM, semi-PD), 8xA100 (DistServe)



DeepSeek-V3, FP16, MATH-500, 8xH200

Cluster-Level Inference

semi-PD instances collectively participate in request routing.



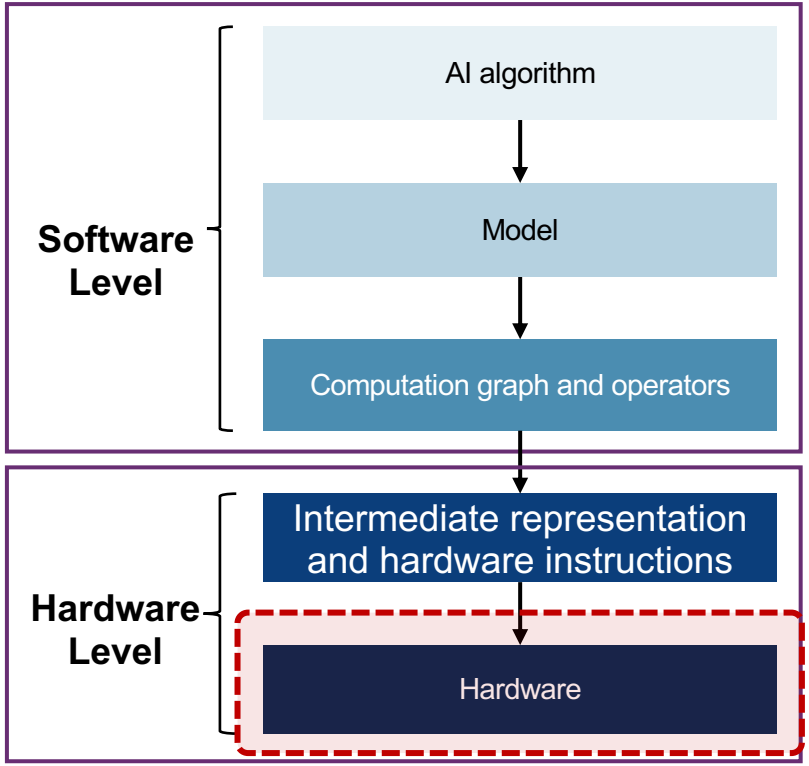
*x*P*y*D*z*S: *x* Prefill instances, *y* Decode instances, ***z* semi-PD instances**, implemented based on NVIDIA Dynamo（baseline）, DeepSeek-V2-Lite model, single A100 per instance

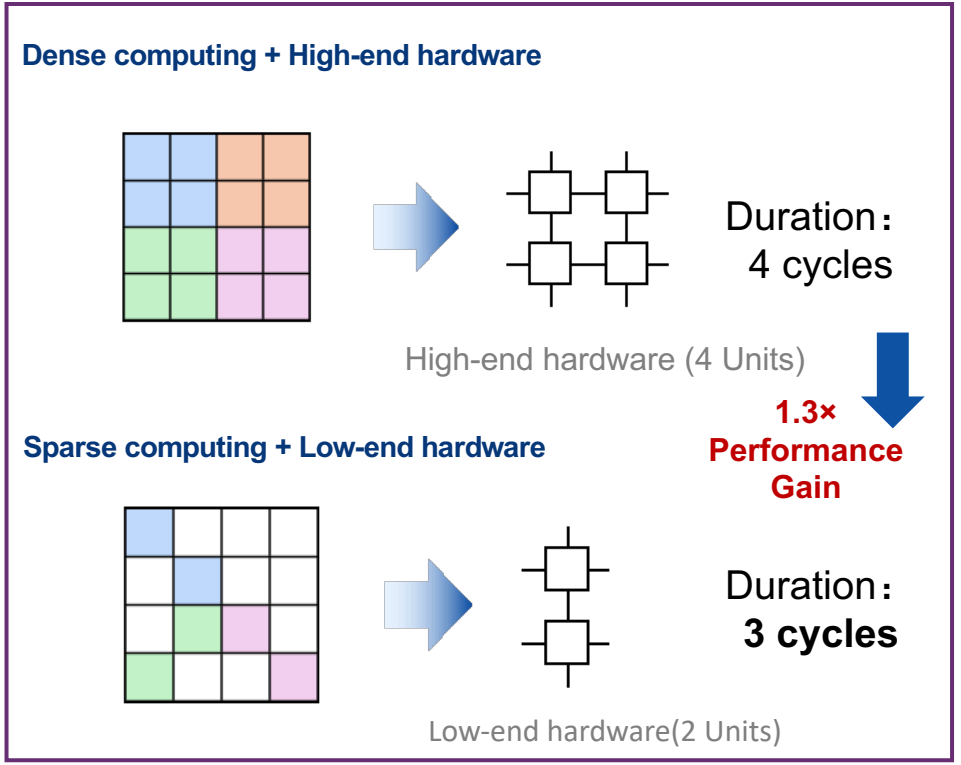
Significant reduction in TTFT/TPOT achieved after replacing 1 Prefill instance and 3 Decode instances with 4 semi-PD instances.

[1] Ke, Hong, et al. "semi-PD: Towards Efficient LLM Serving via Phase-wise Computation Disaggregation and Unified Storage." *arXiv preprint arXiv:2504.19867*. 2025.

Design Space

- **System Design: Hardware-level Optimization**





Hardware for Video Generation Models

Video generation is one of the important modalities, and it is a promising path towards as the physical world's simulators

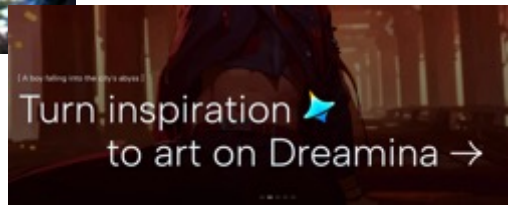
OpenAI's Sora model

Videos generated by Sora are almost indistinguishable from real ones.[1]

Follow-ups by major technology firms

Kuaishou

Google



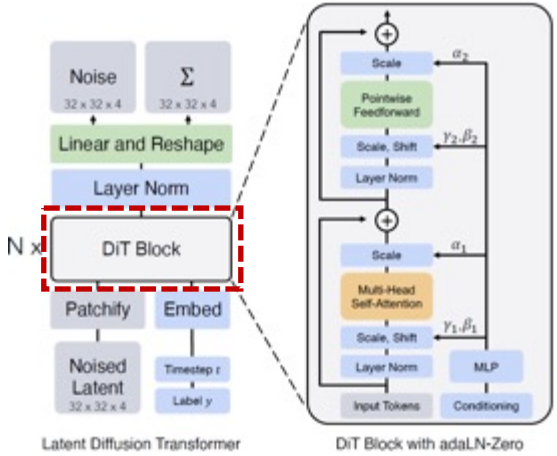
ByteDance

Source：[1] OpenAI， https://openai.com/index/sora/

Hardware for Video Generation Models

VGMs are mainly based on diffusion transformers, composed of noising and denoising process

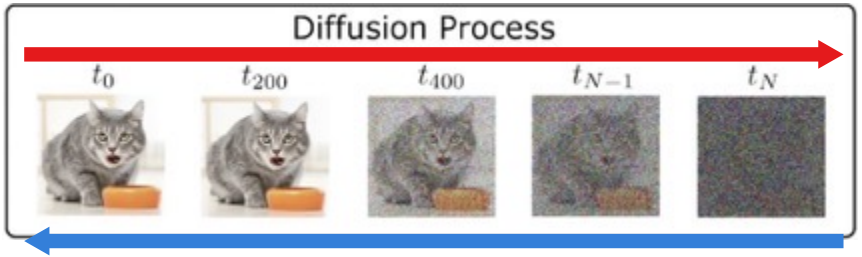
The structure of VGMs



At present, most of the mainstream video generation models adopt the Diffusion Transformer (DiT) architecture[1]

Diffusion

Forward diffusion (backward propagation): Gradually add Gaussian noise of different amplitudes



Reverse diffusion (inference): Denoise gradually

The principle is to endow the model with the ability to generate videos through multiple rounds of noise superposition and denoising training.

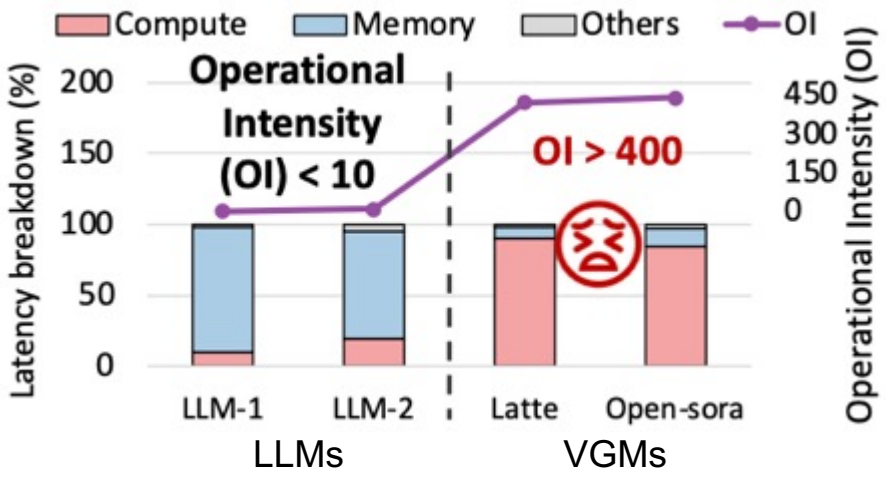
Source：[1] Scalable diffusion models with transformers，ICCV 2023.

Hardware for Video Generation Models

Different from LLMs, the inference bottleneck of video generation models (VGMs) has shifted from memory-bound to compute-bound

	LLMs (mainly considering the decode stage)	VGMs (Temporal, spatial, and FFN structure)
Computation*	$\approx 12d^2 + 2Nd$	$\approx 16FNd^2 + 2FN(F+N)d$
Memory access*	$\approx 12d^2 + 2Nd$	$\approx 16d^2 + 15FNd$
Operational intensity (OI)	$\approx \mathbf{1}$	$\approx \boldsymbol{FN}$

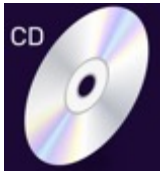
F: Video frames. N: Tokens. d: Hidden dimension.
*Calculation of a single block/layer.



Hardware for Video Generation Models

H.261 [1988]
First modern and useful codec, **max resolution 352*288**

MPEG-1 [1993]
Compressed video **making video CDs possible**



H.262/MPEG-2 [1995]
Developed for TV broadcasts and DVD-Video playback, **max resolution 1920*1080**

H.263 [1996]
Low-bandwidth standard **used for the internet**

H.264/MPEG-4 [1996]
The **most used video codec today**, max resolution

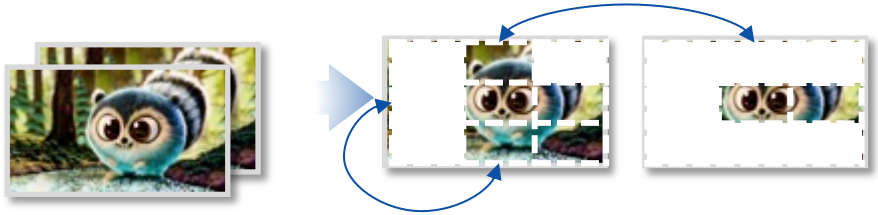
H.265/HEVC [2013]
Further reduce the video size, **enables technologies such as AR and VR** [1]

1000x size compression and overhead reduction!

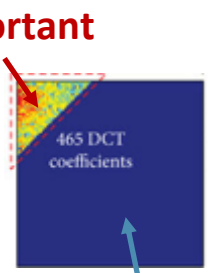
Source：[1] https://medium.com/videocoin/the-history-of-video-codecs-infographic-432e5be1154f.

Hardware for Video Generation Models

Video Compression

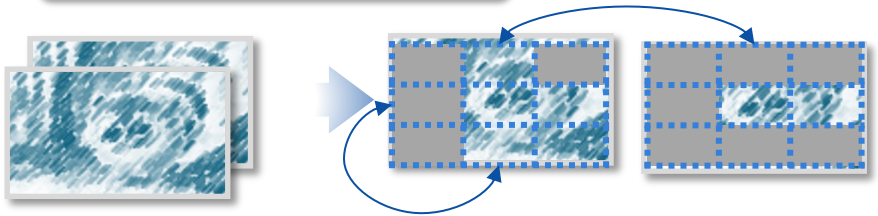


DCT
(Discrete Cosine Transform)



Video with small size

Video Generation



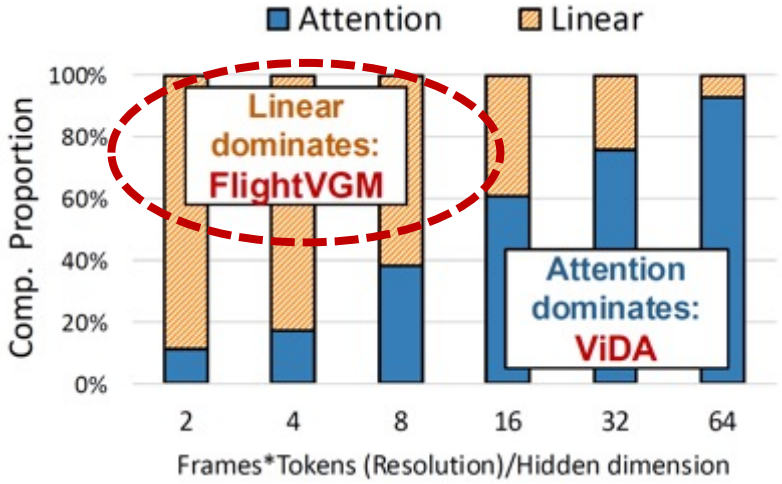
Generated video

1. Activation sparsification

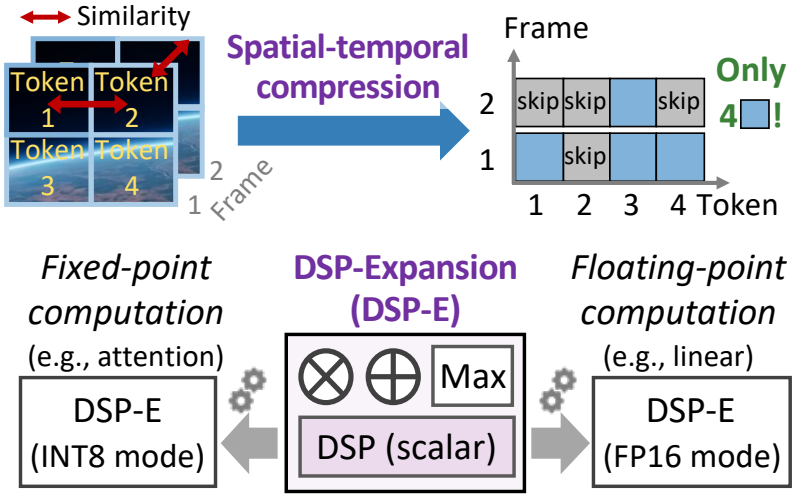
2. Hybrid precision quantization

Hardware for Video Generation Models

For linear-heavy video generation models, we propose FlightVGM, a HW-SW co-design with temporal-spatial & floating-fixed strategies



- Attention dominated： resolution ↑ or dim.↓
- **Linear dominated： resolution↓ or dim.↑**



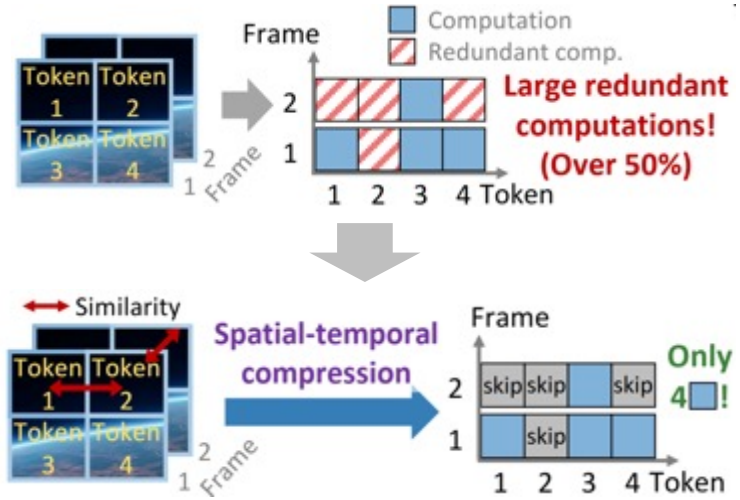
Sparse Computing + Configurable Design → Higher Performance

Hardware for Video Generation Models

Main Method

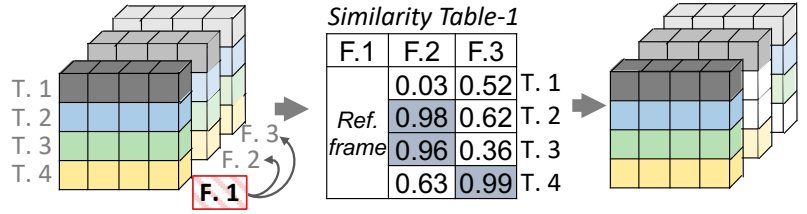
To address the computational redundancy problem in VGMs, a temporal-spatial joint sparse method is proposed

Utilize Inter/Intra-frame Similarity

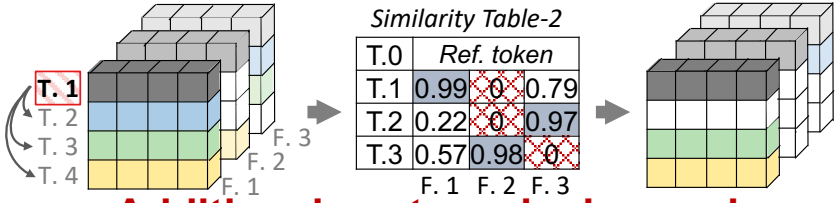


Mining Similarities by Grouping

Step-1: Inter-frame sparsification



Step-2: Intra-frame sparsification



Additional cost can be ignored

($6TNd$ vs. TNd^2), d=1152

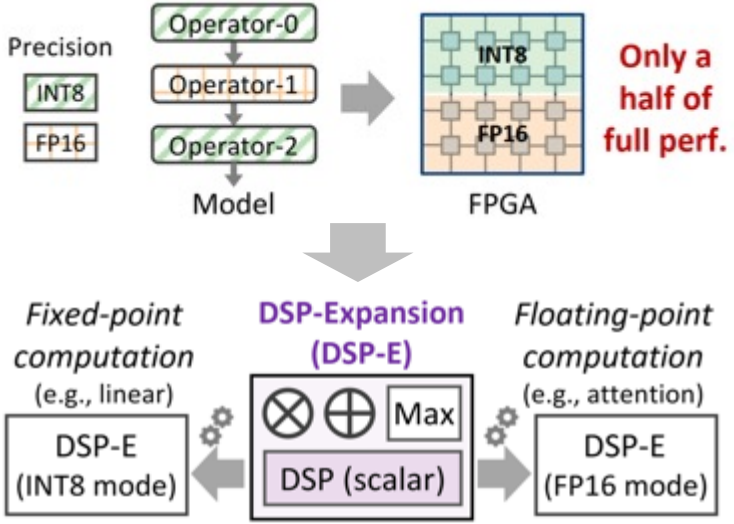
[1] Liu J, Zeng S, Ding L, et al. Flightvgm: Efficient video generation model inference with online sparsification and hybrid precision on fpgas[C]//Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2025: 2-13.

Hardware for Video Generation Models

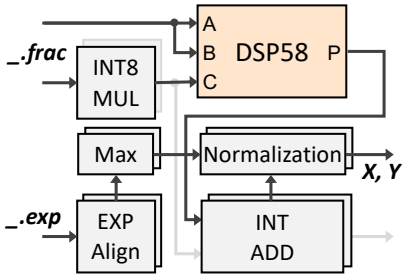
Main Method

A DSP58 extension architecture is proposed to enhance utilization under mixed-precision computing

Floating and Fixed Hybrid Architecture

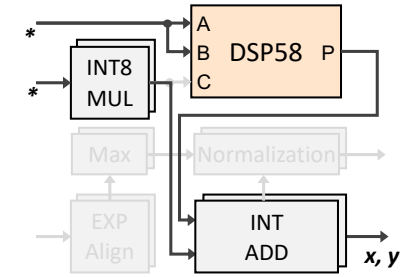


Expanded Architecture based on DSP IP



FP16 mode

$X = A \times B_0 + C_0, Y = A \times B_1 + C_1$



INT8 mode

$[x, y] = [a_0, a_1] \times \begin{bmatrix} b_{00} & b_{01} \\ b_{10} & b_{11} \end{bmatrix} + [c_0, c_1]$

FP16 mode
(2 floating-point MACs)

INT8 mode
(4 floating-point MACs)

[1] Liu J, Zeng S, Ding L, et al. Flightvgm: Efficient video generation model inference with online sparsification and hybrid precision on fpgas[C]//Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2025: 2-13.

Hardware for Video Generation Models

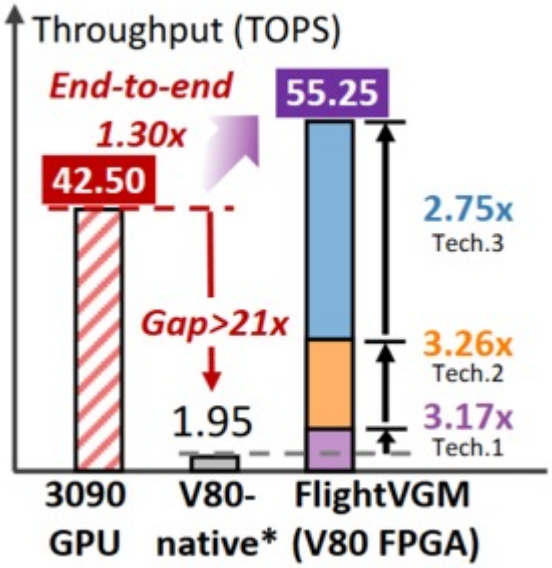
- Models and datasets
  - Models：Latte-1 and Open-Sora 1
  - Datasets：UCF-101
- Metrics
  - CLIPSIM: Text-Video Alignment
  - VBench: Video quality
- Baseline
  - Generic hardware: NVIDIA 3090 GPU
  - FPGA-based accelerator for Transformer: HiSpMV[FPGA'24] and FlightLLM[FPGA'24]
  - ASIC-based accelerator for DiT: InterArch[DAC'24] and CMC[ASPLOS'24]

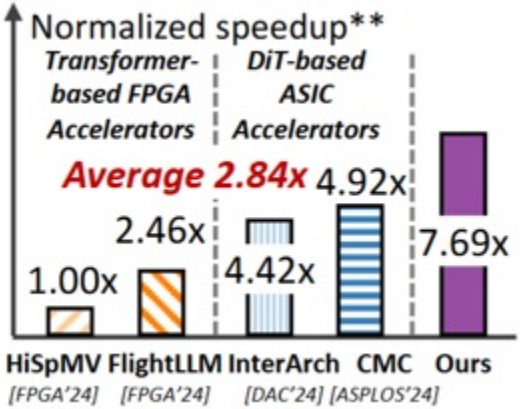
Table 1: Hardware parameters of GPU and FPGA platforms.

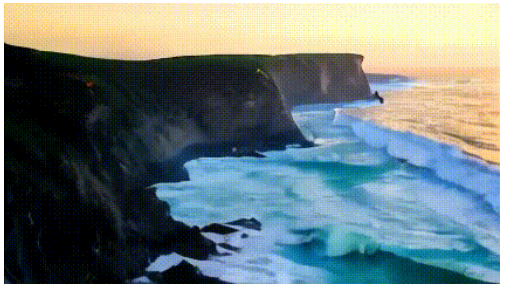
Platform	NVIDIA 3090 GPU	AMD U280 FPGA	AMD V80 FPGA
Compute units	328 Tensor Cores	9024 DSP48s	10848 DSP58s
Frequency (MHz)	1695	225	300
FP16 PCP (TOPS)	142	5.4	6.5
Memory (GB)	24	8 & 32	32 & 32
BW (GB/s)	936	38 & 460	51 & 819

Hardware for Video Generation Models

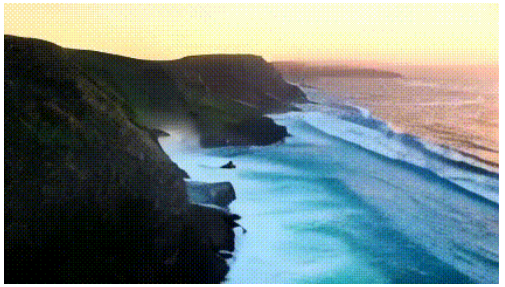
With 21× lower peak compute than the NVIDIA 3090, FlightVGM achieves 1.3× higher speedup and 4.5× better energy efficiency







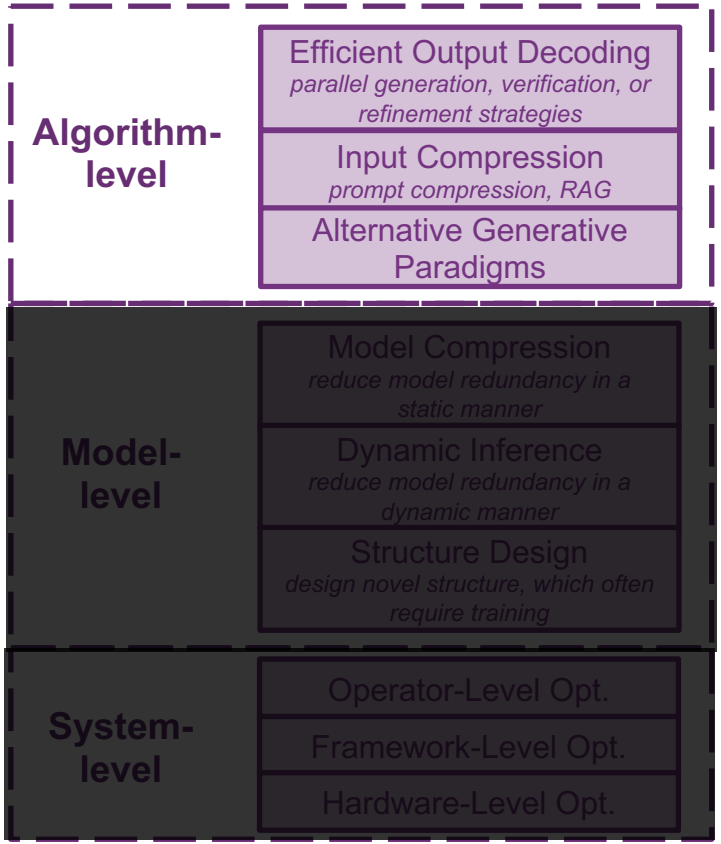
Video generated by original model



Video generated by efficient model

[1] Liu J, Zeng S, Ding L, et al. Flightvgm: Efficient video generation model inference with online sparsification and hybrid precision on fpgas[C]//Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays. 2025: 2-13.

Menu of Techniques



Efficient Output Decoding

- Speculative Decoding
- Jacobi Decoding
- Agentic Generation

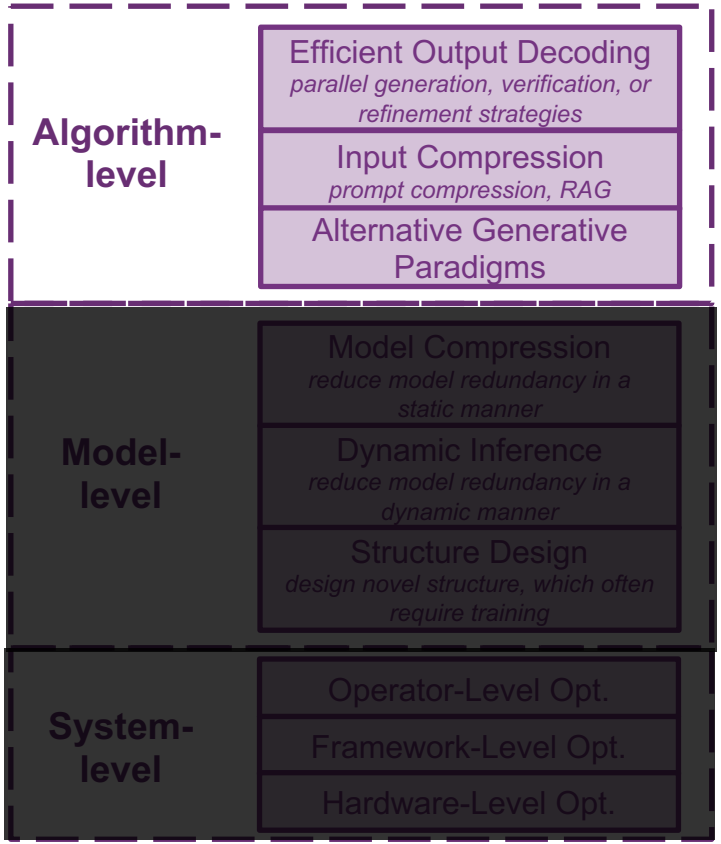
Input Compression

- Input Compression

Alternative Generative Paradigms

- Diffusion for Text

Menu of Techniques



Efficient Output Decoding

- **Speculative Decoding**
- Jacobi Decoding
- Agentic Generation

Input Compression

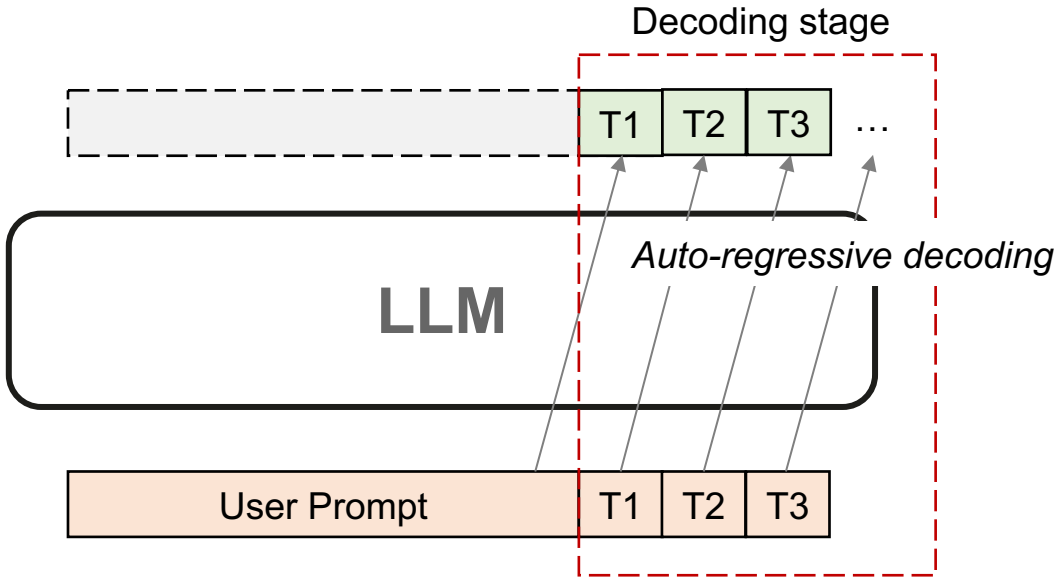
- Input Compression

Alternative Generative Paradigms

- Diffusion for Text

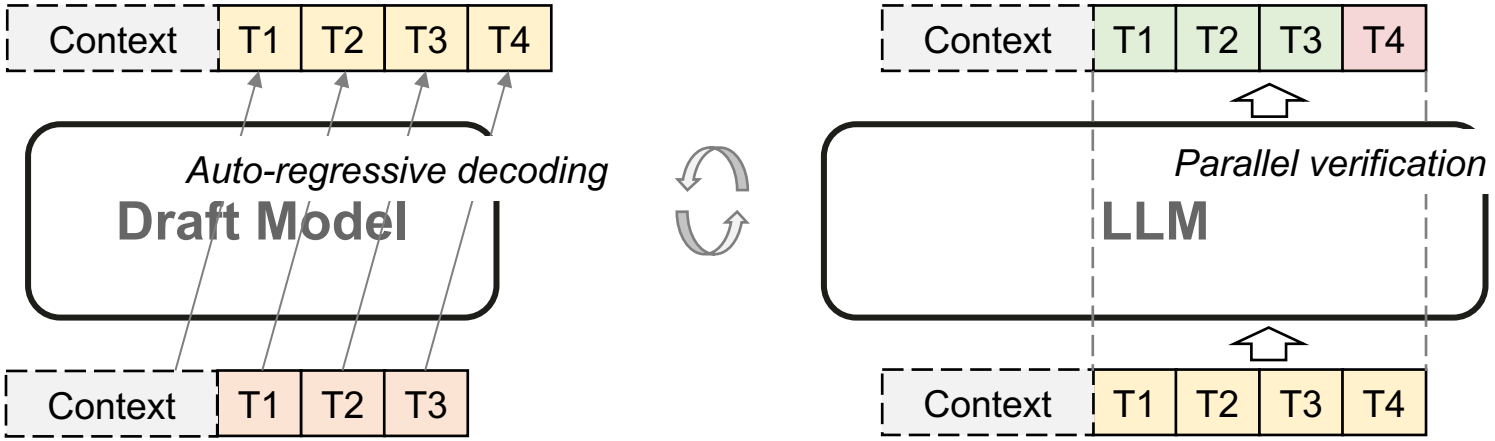
Speculative Decoding: Concepts

- Recall: autoregressive decoding of LLMs
  - Decoding stage: memory bound (data transfer of model weights & KV cache)
  - Redundant computation is left unused!



Speculative Decoding: Concepts

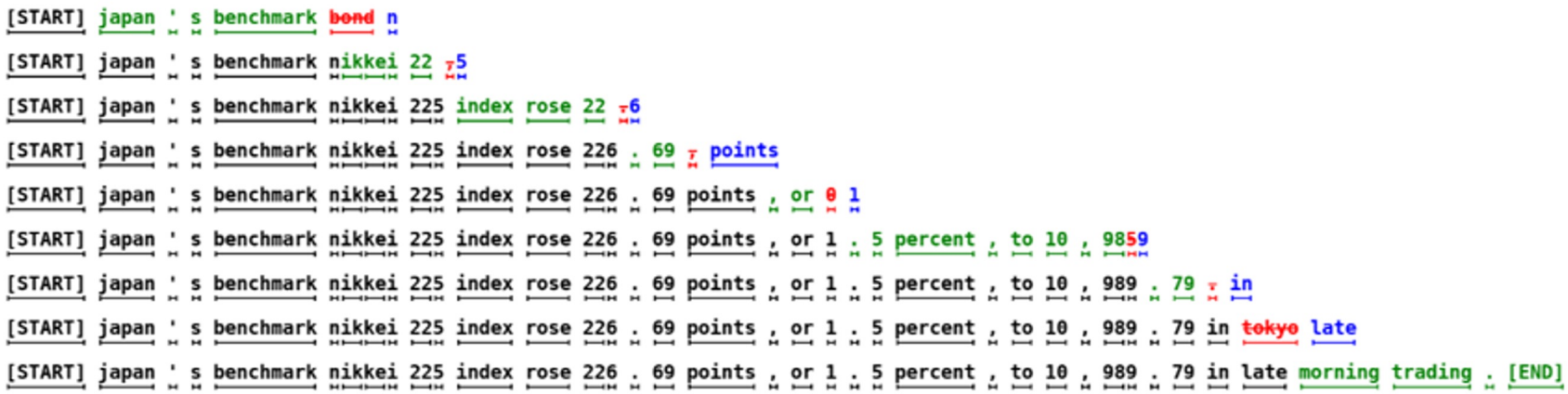
- Core idea of speculative decoding
  - Use a small draft model to generate multiple token for verification
  - The LLM conducts parallel verification (memory bound allows more computation)
  - Key elements: 1) the acceptance rate of generated tokens; 2) the cost of draft model



[1] Leviathan, Yaniv, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. ICML 2023.

Speculative Decoding: Demo

- Demo from [1]
  - Green: accepted tokens
  - Red: rejected tokens
  - Blue: corrected tokens



[1] Leviathan, Yaniv, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. ICML 2023.

Speculative Decoding: Speed-up Estimation

- The speed-up rate of speculative decoding (SD) can be estimated!
  - ***S***: the total number of tokens
  - ***R***: the number of SD rounds
  - ***γ***: the number of generated tokens in each SD round

$$Speedup = \frac{T_{AR}}{T_{SD}} = \frac{S \cdot T_T(B,1)}{R \cdot \left(\gamma \cdot T_D(B,1) + T_T(B,\gamma) + T_{reject}\right)} = \frac{S}{R} \cdot \frac{1}{\gamma \cdot \frac{T_D(B,1)}{T_T(B,1)} + \frac{T_T(B,\gamma)}{T_T(B,1)} + \frac{T_{reject}}{T_T(B,1)}}$$

① ② ③

① $\frac{T_D(B,1)}{T_T(B,1)}$ *The relative latency of draft model to the target model*

② $\frac{T_T(B,1)}{T_T(B,\gamma)}$ *The cost of multi-token verification. A large batch size B is harmful to speed-up rate*

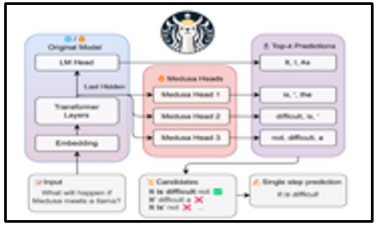
③ $\frac{T_{reject}}{T_T(B,1)}$ *The negligible cost of token sampling for the rejected tokens*

[1] Sadhukhan, Ranajoy, et al. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. ICLR 2025.
[2] Huang, Zongle, et al. MoESD: Unveil Speculative Decoding's Potential for Accelerating Sparse MoE. arXiv preprint arXiv:2505.19645 (2025).

Speculative Decoding: Representative Works

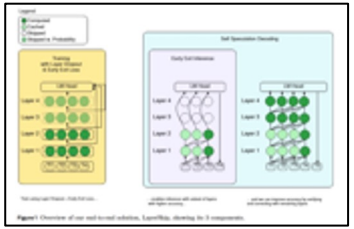
- How to find an good draft model?
  - consistent with the target model
  - efficient in decoding

23-9: Medusa [1] | **24-1: Eagle [2]** | **24-10: LayerSkip [3]** | **24-12: DeepSeek-V3 [4]**



[Draft model] Independently trained multi-layer decoders

[Draft model] A single transformer layer taking the output from LLM

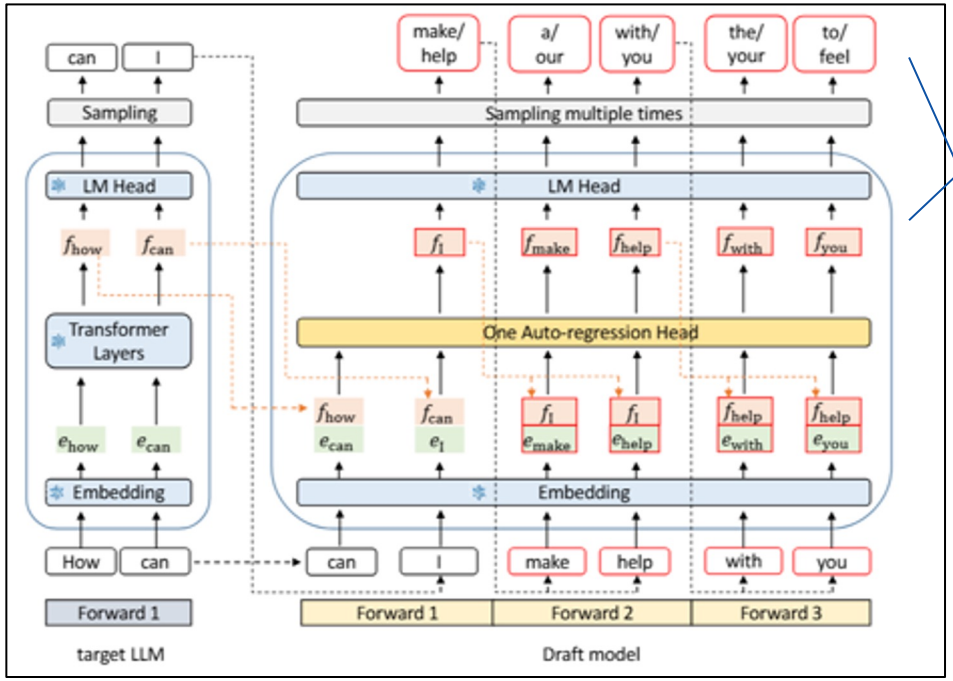


[Draft model] The first a few layers of the LLM itself

[Draft model] Independent multi-layer perception pre-trained together with DS-V3

[1] Tianle Cai, et. al. Medusa: Simple llm inference acceleration framework with multiple decoding heads. ICML 2024.
[2] Yuhui Li, et. al, EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty, ICML 2024
[3] Mostafa Elhoushi, et. al. LayerSkip: Enabling Early Exit Inference and Self-Speculative Decoding, arXiv preprint, 2024
[4] Deepseek Team. DeepSeek-V3 Technical Report.

Speculative Decoding: Eagle

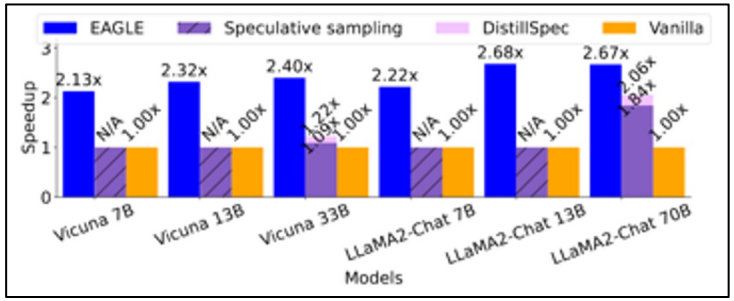


Drat model: a single transformer layer

- # params: 0.25B ~ 1B
- # training data: 1B tokens
- Acceptance rate: 75%

Tree attention: more tokens per pass

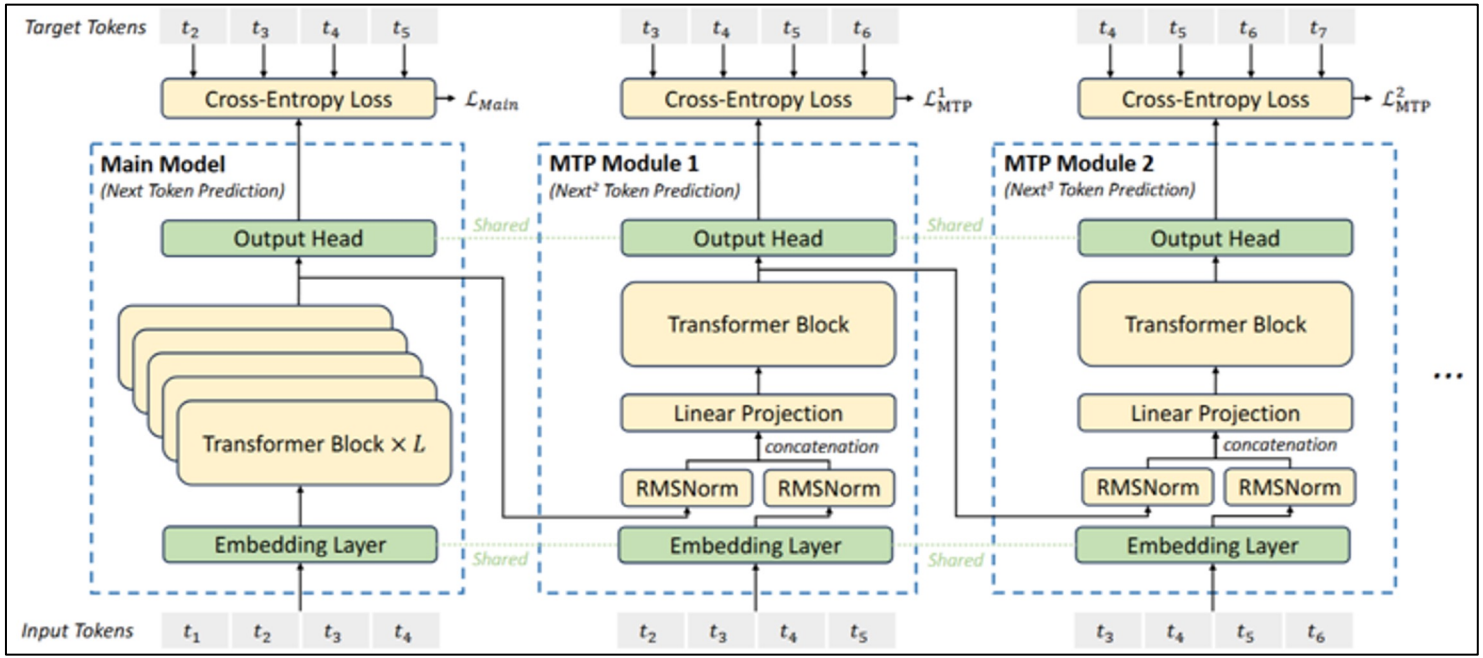
- Verify more tokens per pass
- E.g., 12 tokens (4 paths)



[1] Yuhui Li, et. al, EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty, ICML 2024
[2] Yuhui Li, et, al, Eagle-2: Faster inference of language models with dynamic draft trees. EMNLP 2024
[3] Yuhui Li, et. al, EAGLE-3: Scaling up Inference Acceleration of Large Language Models via Training-Time Test, arXiv preprint 2503.01840.

Speculative Decoding: Multi-Token Prediction

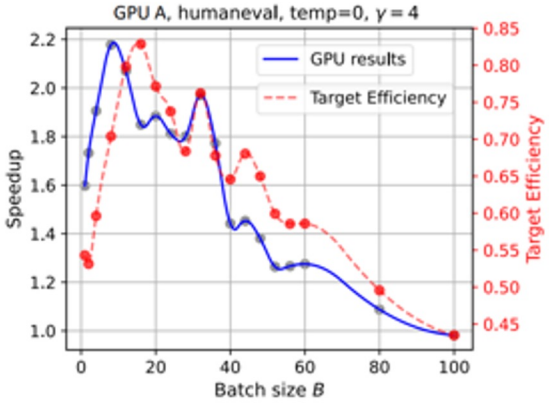
- Multi-Token Prediction (MTP): trained from scratch with the LLM backbone

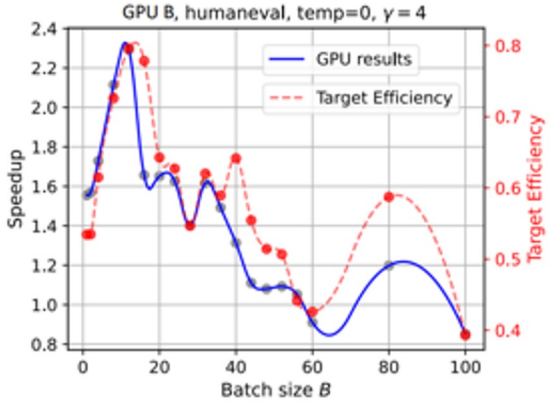


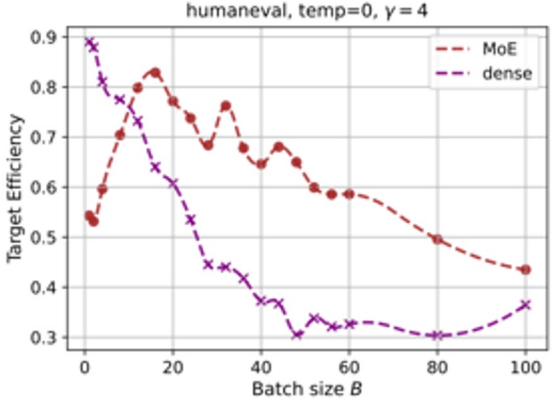
[1] Deepseek Team. DeepSeek-V3 Technical Report.

Speculative Decoding for MoE Architectures

- SD is hard to tackle MoE:
  - **MoE does not favor small batch:** Additional memory costs to load experts
  - **SD does not favor large batch:** SD is beneficial for memory-bound systems
- SD + MoE is helpful under medium batch size
  - The number of activated experts saturates, but does not reach compute-bound

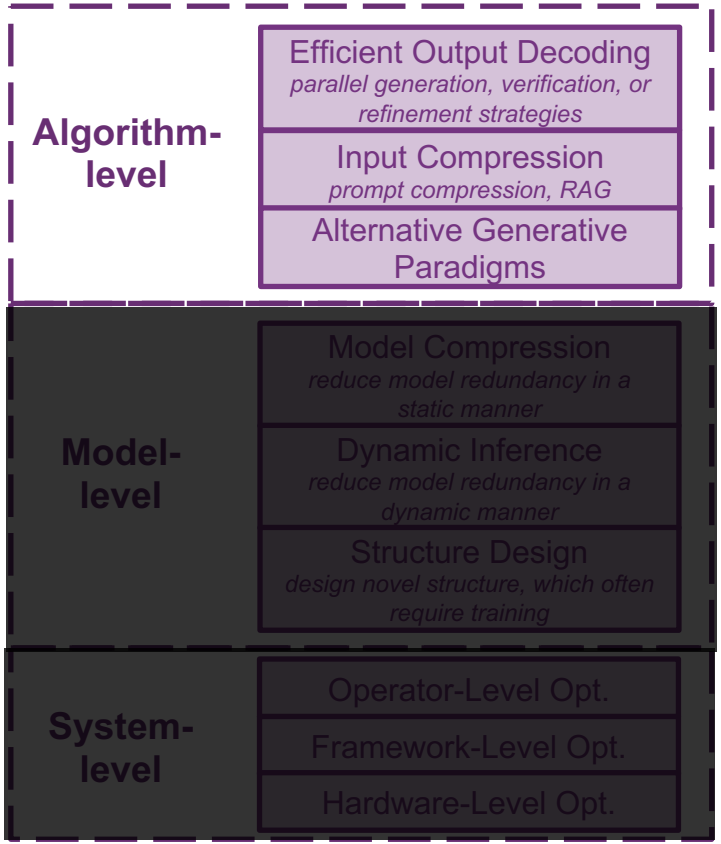






[1] Huang, Zongle, et al. MoESD: Unveil Speculative Decoding's Potential for Accelerating Sparse MoE. NeurIPS 2025 spotlight.

Menu of Techniques



Efficient Output Decoding

- Speculative Decoding
- **Jacobi Decoding**
- Agentic Generation

Input Compression

- Input Compression

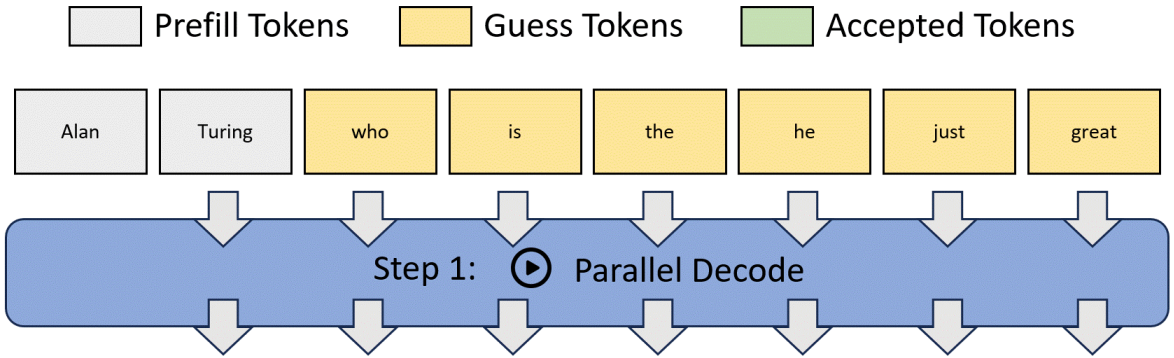
Alternative Generative Paradigms

- Diffusion for Text

Jacobian Decoding

Drawbacks of speculative decoding

- Low acceptance rate, decoding time etc.
- A draft model needs to be separately trained and paired with the target LLM



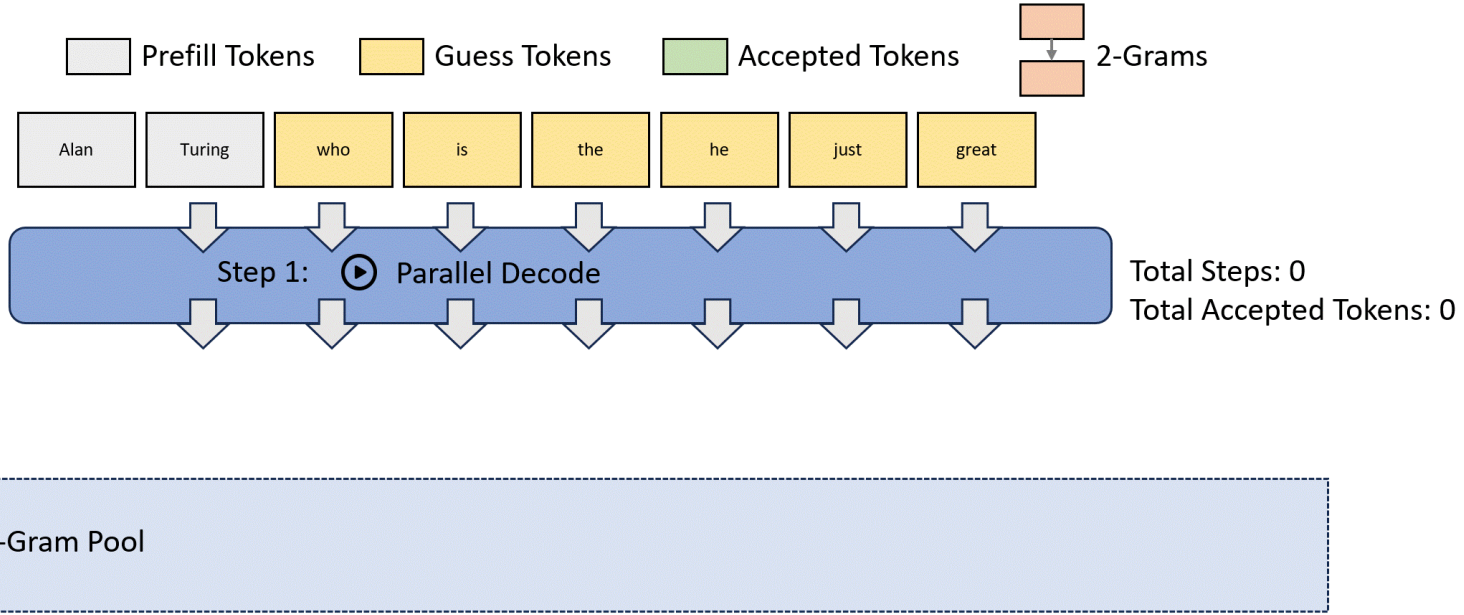
Jacobian decoding:
A single LLM without the draft model

Total Steps: 0
Total Accepted Tokens: 0

[1] https://

Lookahead Decoding

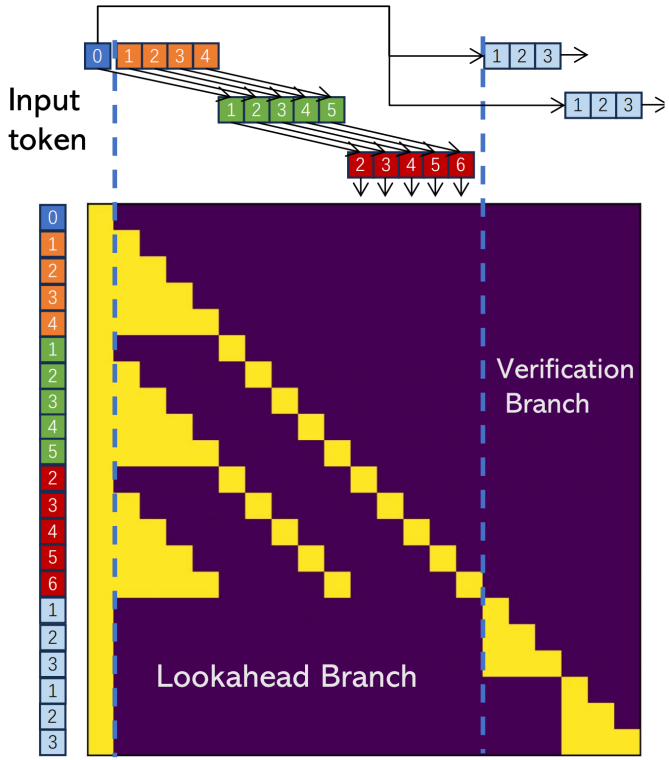
“Lookahead”: reuse the promising draft from past N-gram trajectories



[1] https://lmsys.org/blog/2023-11-21-lookahead-decoding/

Lookahead Decoding

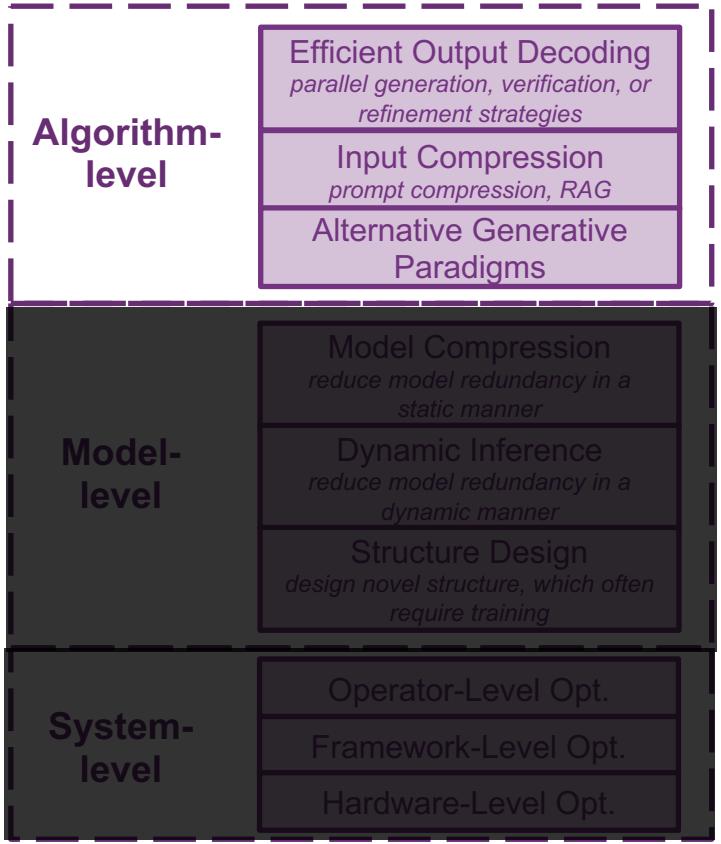
Lookahead branch maintains a fixed-sized, 2D window to generate n-grams from the Jacobi iteration trajectory.



Verification branch selects and verifies promising n-gram candidates.

[1] https://lmsys.org/blog/2023-11-21-lookahead-decoding/

Menu of Techniques



Efficient Output Decoding

- Speculative Decoding
- Jacobi Decoding
- **Agentic Generation**

Input Compression

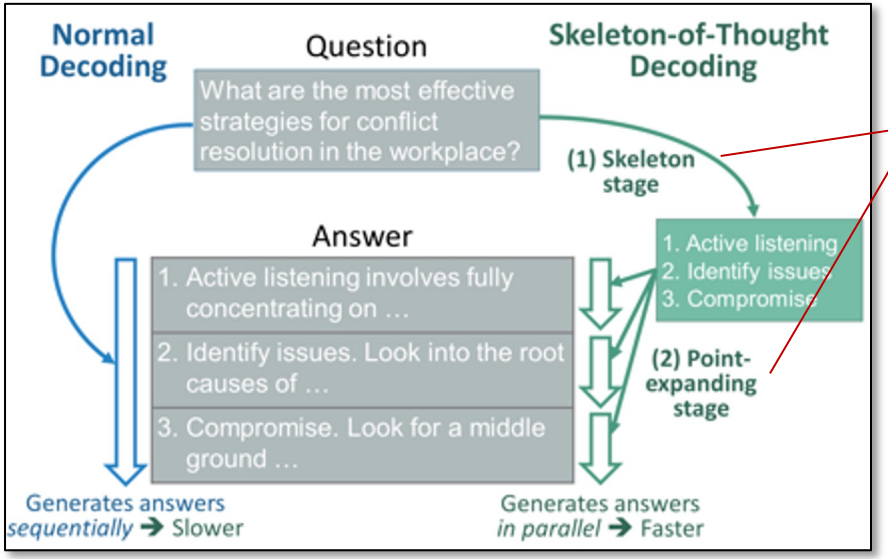
- Input Compression

Alternative Generative Paradigms

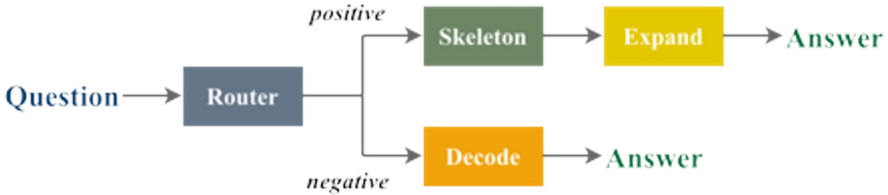
- Diffusion for Text

Skeleton-of-Thought (SoT)

SoT: LLM generates the skeleton autoregressively, and then each points in parallel
(an attempt in **agentic generation** for efficiency)



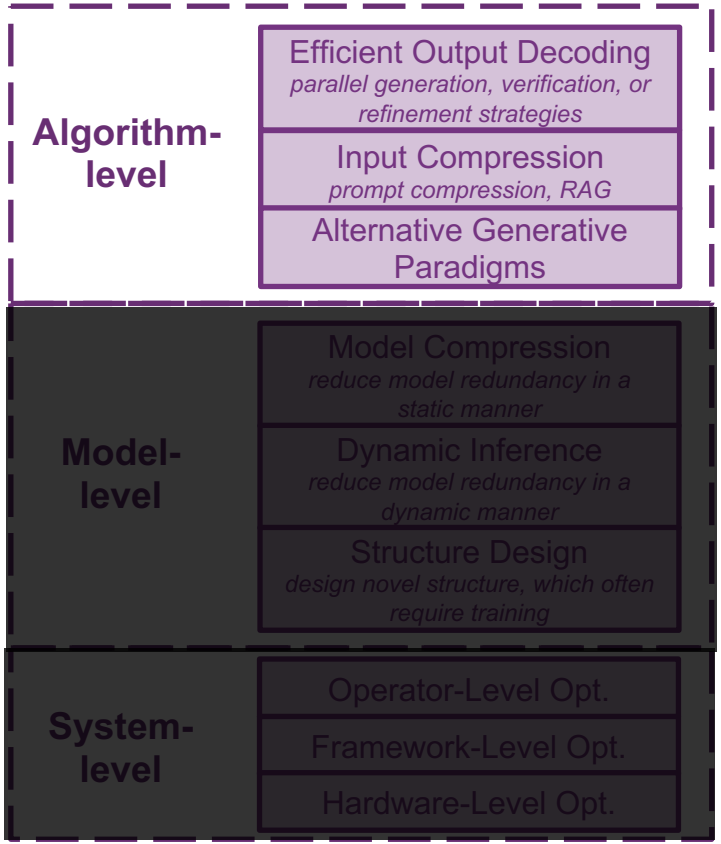
- **Skeleton Stage**: Guide the LLM to output a concise skeleton of the answer
- **Point-expanding Stage**: Guide the LLM to expand on each point from the skeleton in parallel
- Achieve up to **2.39x** end-to-end speed-up



SoT in practice: A router to classify queries (SoT-R)

[1] Ning, Xuefei*, Zinan Lin*, et. al., "Skeleton-of-Thought: Prompting LLMs for Efficient Parallel Generation." ICLR 2024.

Menu of Techniques



Efficient Output Decoding

- Speculative Decoding
- Jacobi Decoding
- Agentic Generation

Input Compression

- **Input Compression**

Alternative Generative Paradigms

- Diffusion for Text

Prompt Compression

- **Prompt compression**: eliminate redundant tokens in the prompt

Selective Context: filter out redundant tokens to shorten the input prompt

Original: INTRODUCTION Continual Learning (~~CL~~) ; ~~also known as Lifelong Learning ; is~~ a promising learning paradigm to design models ~~that~~ have to ~~learn~~ how ~~to perform multiple tasks~~ across ~~different environments~~ over ~~their lifetime .~~ To uniform the language and enhance ~~the readability of the paper we~~ adopt the unique term continual learning ~~(CL) . ~~ Ideal CL models in ~~the real world~~ should ~~be~~ deal ~~with~~ domain shifts ; researchers ~~have~~ recently started ~~to~~ sample tasks ~~from~~ two different datasets . For instance , proposed to train and evaluate ~~a model~~ on Imagenet first ~~and then~~ challenge ~~its performance on the~~ Places365 ~~dataset .~~ considers more scenarios , starting ~~with~~ Imagenet or Places365 , ~~and then moving on to~~ the VOC/CUB/Scenes datasets . Few works propose more advanced scenarios built ~~on~~ top ~~of~~ more than two datasets .

Filtered: INTRODUCTION Continual Learning (a promising learning paradigm to design models have to how across over To uniform the language and enhance adopt the unique term continual learning Ideal CL models in should deal domain shifts researchers recently started sample tasks two different datasets For instance proposed to train and evaluate on Imagenet first challenge Places365 considers more scenarios starting Imagenet or Places365 the VOC/CUB/Scenes datasets Few works propose more advanced scenarios built top more than two datasets

1. Identify the token importance

$$I(x) = -\log_2 P(x_t | x_0, x_1, ..., x_{t-1})$$

2. Group tokens to units

$$I(u) = \sum_{i=t}^{\alpha} I(x_i)$$

3. Sort units in the descending order

$$I_p = \texttt{np.percentile}([I(u_0), .., I(u_k)], p)$$

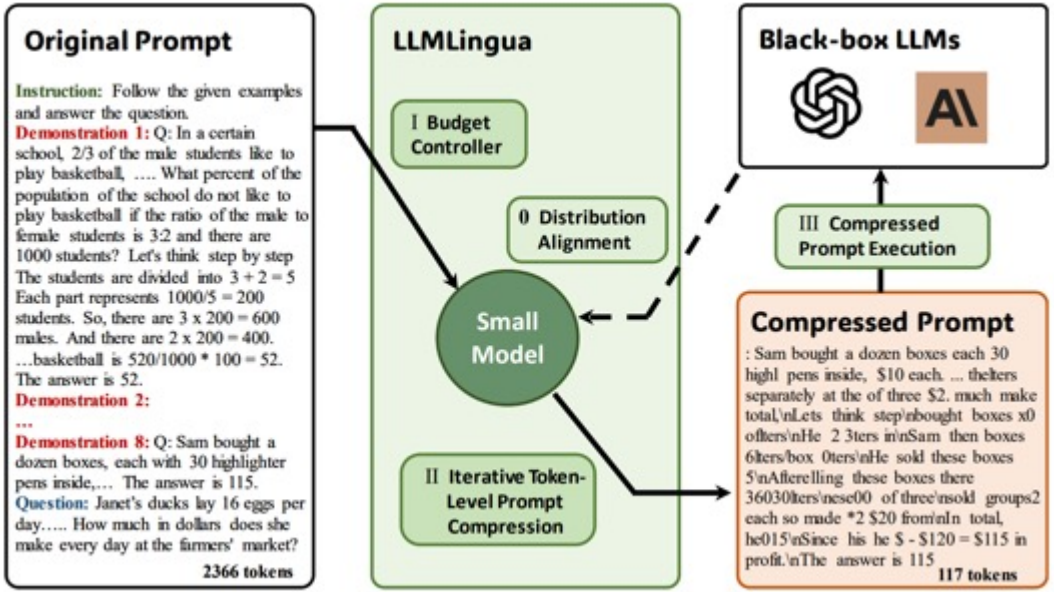
4. Keep units above the threshold

$$C' = U_i \mid I(U_i) \geq I_p, 1 \leq i \leq n$$

[1] Li Y, Dong B, Guerin F, et al. Compressing Context to Enhance Inference Efficiency of Large Language Models. EMNLP, 2023.

LLMLingua

Compress the prompt with an small language model, with an reduction rate up to 20x.



0. Distribution alignment

Instruction tuning of small LLM

1. Budget controller

Calculate the token-wise perplexity and sort in the descending order

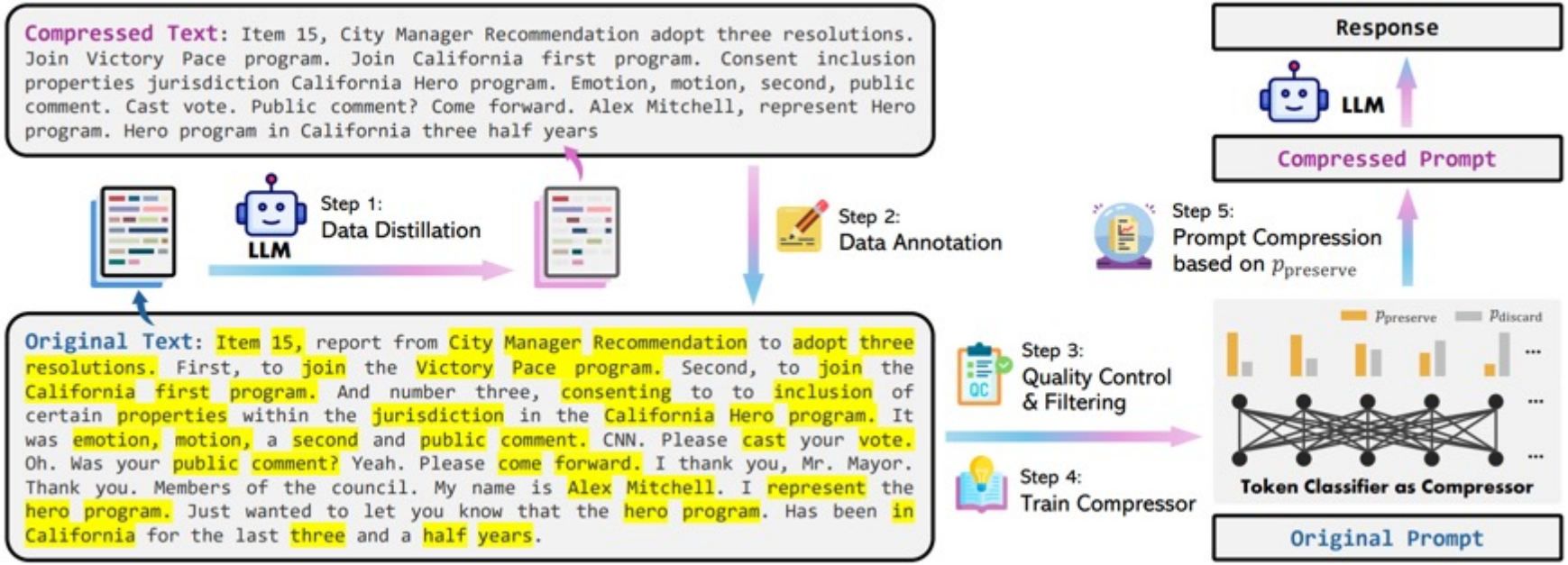
2. Iterative prompt compression

Group tokens into segments to ensure their dependency, and compute segment-wise perplexities

[1] Huiqiang Jiang, et al. RECOMP: Improving Retrieval-Augmented LMs with Compression and Selective Augmentation. EMNLP, 2023.

LLMLingua-2

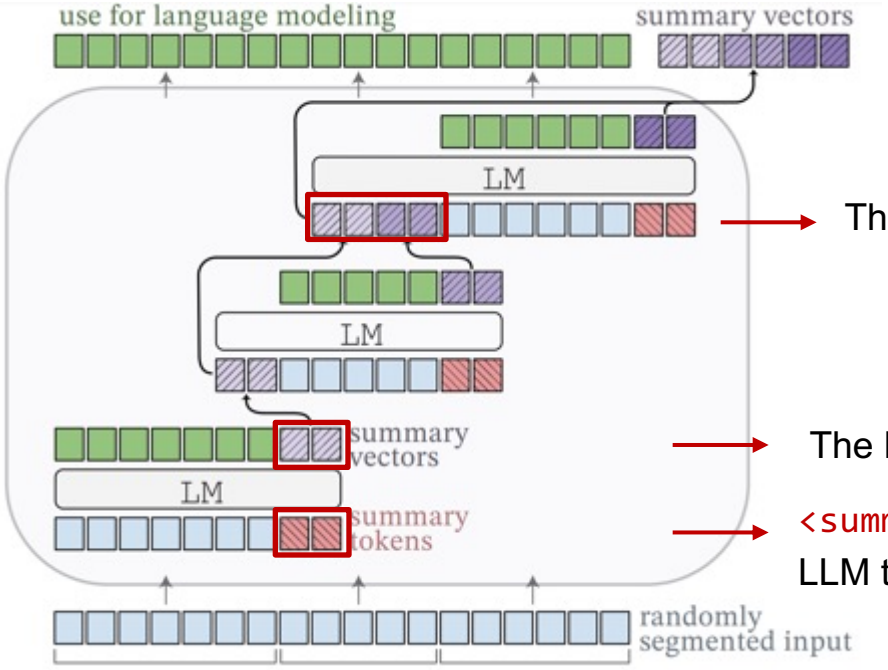
A better way to construct training data for the compressor: instructed by GPT-4



[1] Pan Z, Wu Q, Jiang H, et al. Llmlingua-2: Data distillation for efficient and faithful task-agnostic prompt compression[J]. arXiv preprint arXiv:2403.12968, 2024.

AutoCompressor

The LLM learns to summarize history context given the instruction `<summary_token>`



→ The context only consists of summary tokens

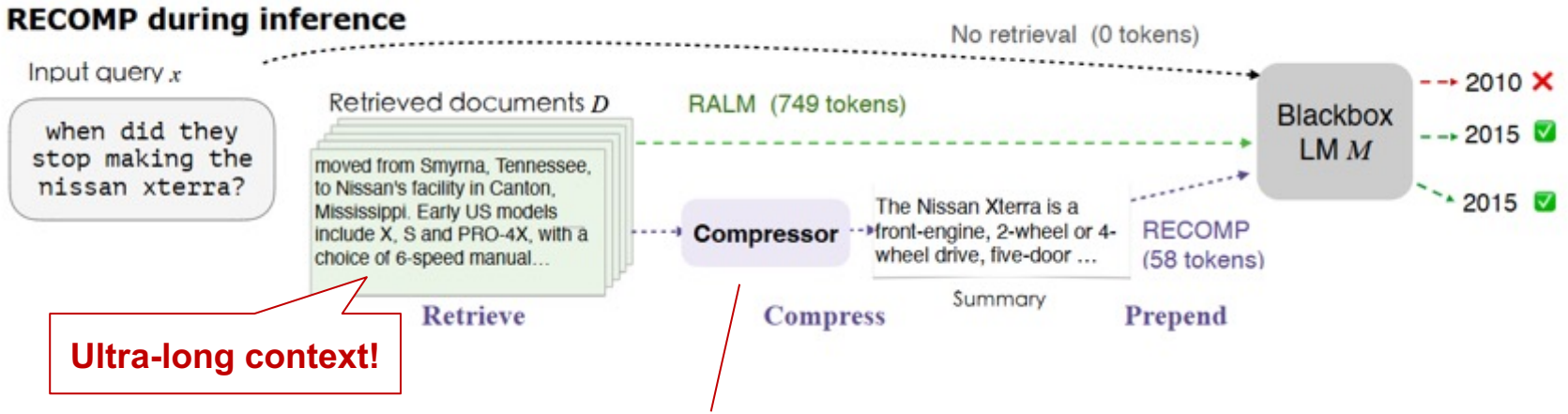
→ The LLM will generate summarized history

→ `<summary_token>`: A new token to tell the LLM to summarize the context

[1] Chevalier A, Wettig A, Ajith A, et al. Adapting Language Models to Compress Contexts. EMNLP, 2023.

RECOMP

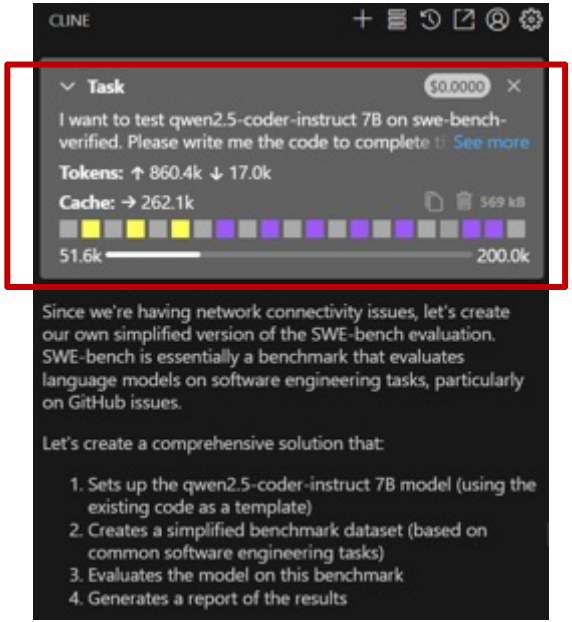
- Prompt compression for RAG systems
  - Otherwise, the retrieved documents can be extremely long



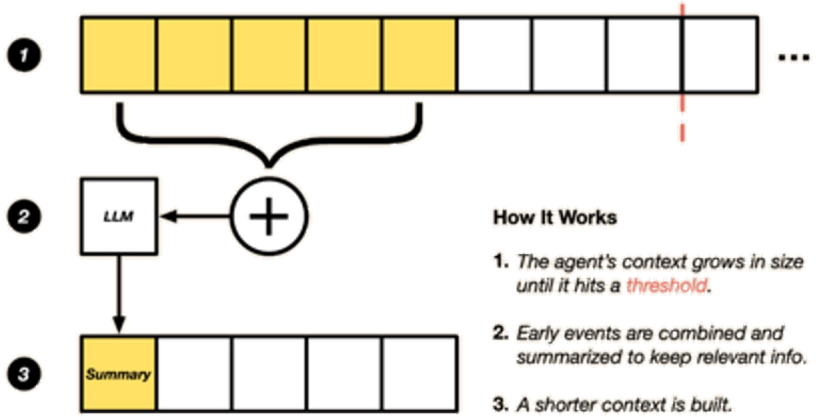
Abstractive Compressor: summarize text from the documents
Extractive Compressor: extract text from the documents

Context Folding for Agentic AI

- **Context Folding:** prompt compression for Agentic AI
  - Deep research, coding agent, etc.



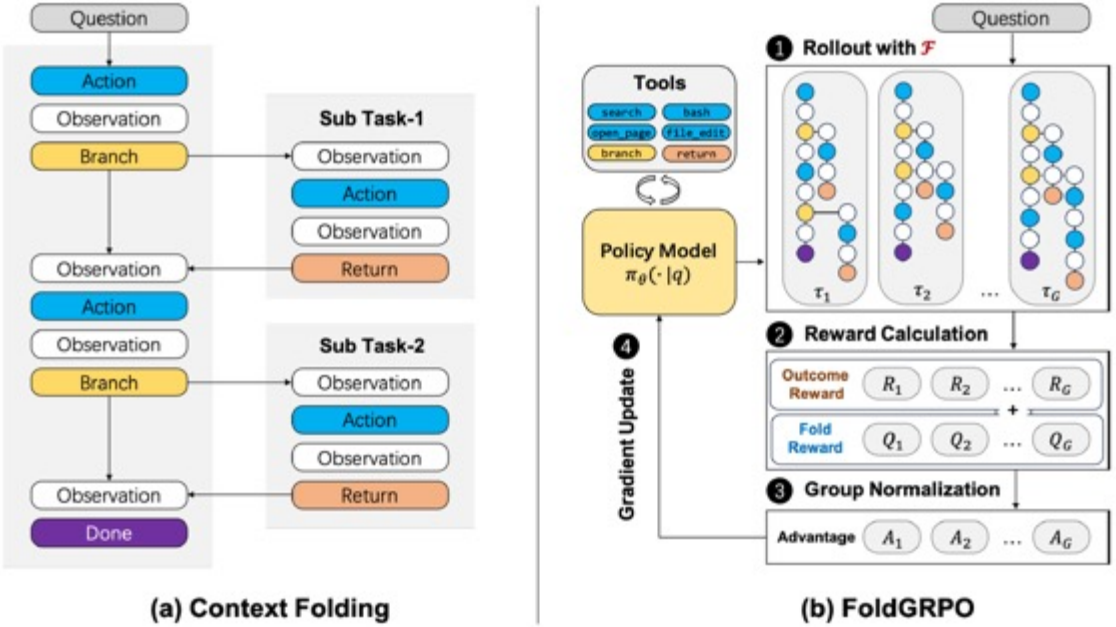
Key idea: A LLM to summarize the history whenever the context reaches some threshold



[1] https://openhands.dev/blog/openhands-context-condensation-for-more-efficient-ai-agents

Context Folding for Agentic AI

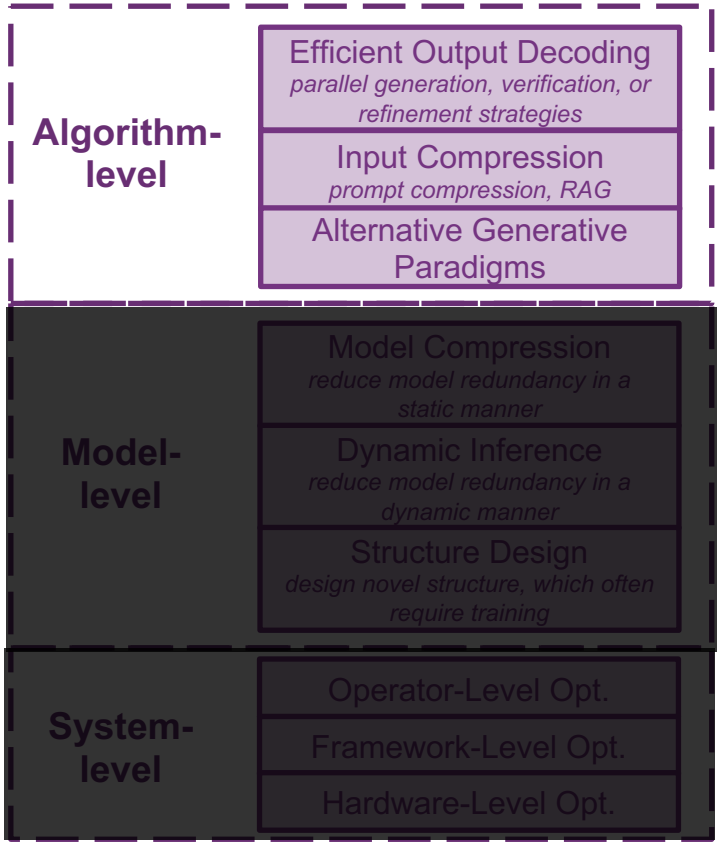
- **Context Folding:** prompt compression for Agentic AI



1. Learn to create, solve and summarize sub-tasks
2. Reduce the output context from sub-task result
3. SFT/RL training to enable the ability to fold context
4. Context reduction 10x without accuracy drop on SWE & BrowseComp

[1] Weiwei Sun, et al. Scaling Long-Horizon LLM Agent via Context-Folding. arXiv,2510.11967.

Menu of Techniques



Efficient Output Decoding

- Speculative Decoding
- Jacobi Decoding
- Agentic Generation

Input Compression

- Input Compression

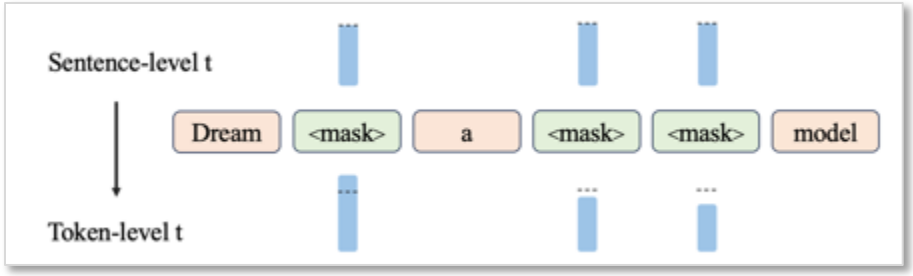
Alternative Generative Paradigms

- **Diffusion for Text**

Diffusion Language Models

Any other solutions? —— Don't use the autoregressive model?

—— To fully parallelized input & output, diffusion language models




```
Janet's ducks lay 16 eggs per day. She eats three for breakfast
every morning and bakes muffins for her friends every day with
four. She sells the remainder at the farmers' market daily for $2
per fresh duck egg. How much in dollars does she make every day at
the farmers' market?
```

Completion

```
Write a story that ends with "Finally, Joey and Rachel get
married."
```

```
Finally, Joey and Rachel get married.
```

Infiliing

[1] Jiacheng Ye, et al. Dream 7B: Diffusion Large Language Models. arXiv preprint, 2508.15487.

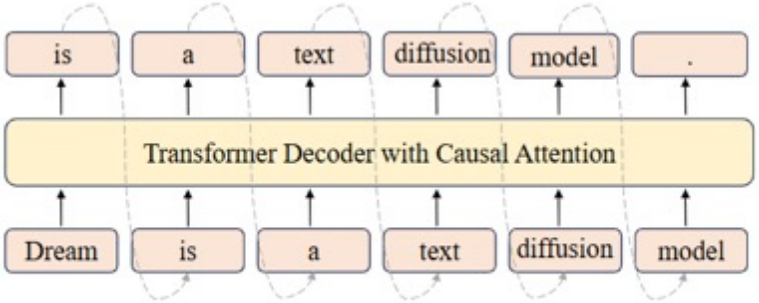
Diffusion Language Models

- Autoregressive modeling v.s. Diffusion modeling

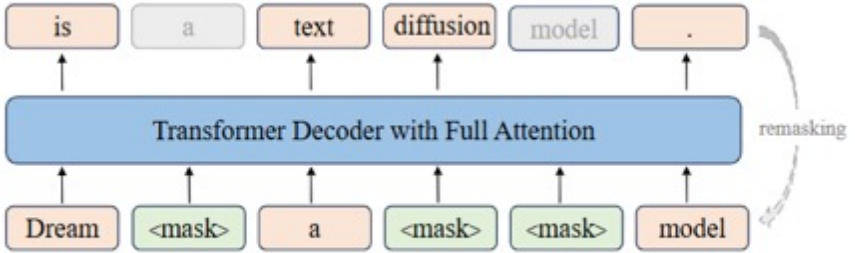
$$\underbrace{\max_{\theta} \mathbb{E}_{p_{\text{data}}(x)} \log p_\theta(x) \Leftrightarrow \min_{\theta} \mathrm{KL}(p_{\text{data}}(x) || p_\theta(x))}_{\text{Generative modeling principles}}$$

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = p_{\boldsymbol{\theta}}(\boldsymbol{x}^1) \underbrace{\prod_{n=2}^{N} p_{\boldsymbol{\theta}}(\boldsymbol{x}^n \mid \boldsymbol{x}^{1:n-1})}_{\text{progressive left-context prediction}},$$

$$p_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{\mathbf{x}_{1:T} \sim q} p(\mathbf{x}_T) \underbrace{\prod_{t=1}^{T} p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1} | \mathbf{x}_t)}_{\text{progressive full-context prediction}}.$$



(a) Autoregressive Modeling

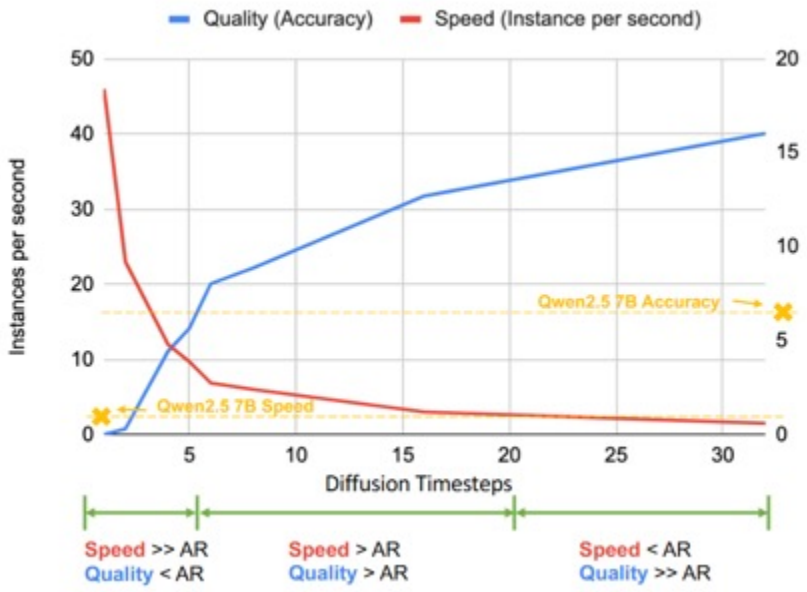


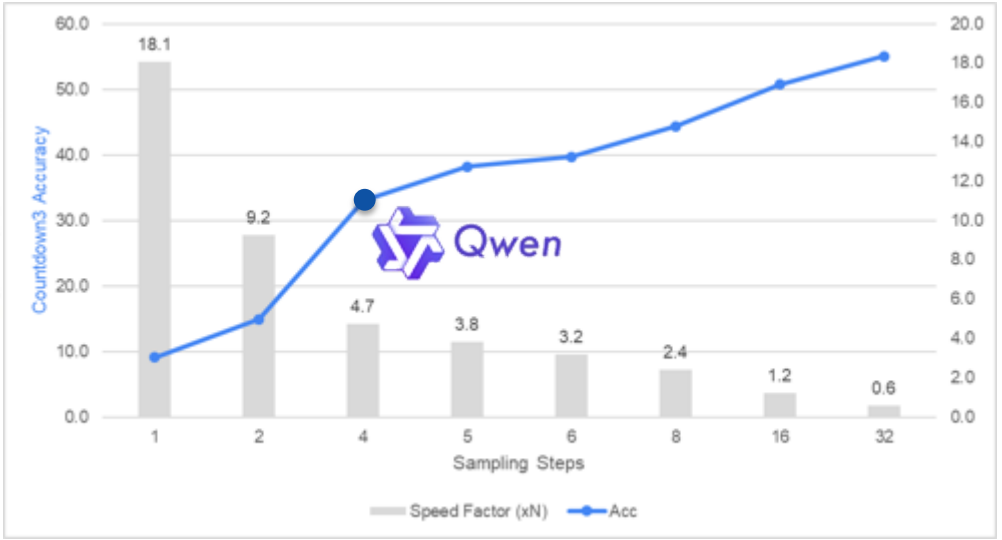
(b) Diffusion Modeling in Dream

[1] Jiacheng Ye, et al. Dream 7B: Diffusion Large Language Models. arXiv preprint, 2508.15487.
[2] Shen Nie, et al. Large Language Diffusion Models. NeurIPS 2025.

Diffusion Language Models

- Inference latency of diffusion models





Contents

Tutorial Review

- **Motivation**: Scaling up model / data / computation based on Transformer is the mainstream and effective pathway for stronger generative AI till now. The scaling up of the model size and input & output cause efficiency issues.

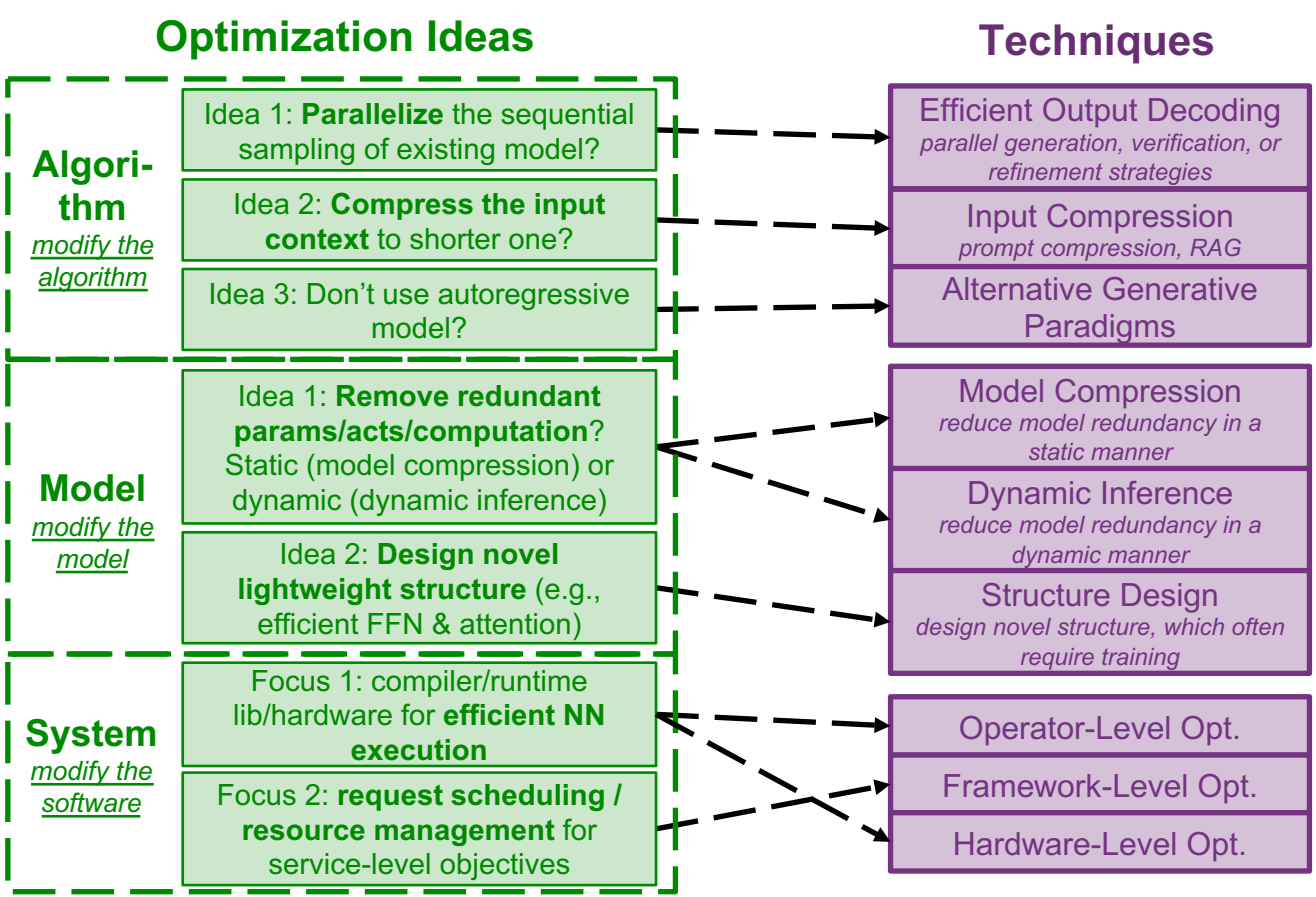
- **Preliminary**:
  - Most LLMs use autoregressive model as the generative modeling method, the transformer architecture, in which the attention operation is a core mechanism.
  - We introduce basic concepts of software, hardware system, device, chip, microarchitecture, and the interface between software & hardware – instruction.
  - AI inference is seen as forwarding data on a computational graph, where each node represents a single operator, edge represents dependency. Operators are translated to instructions. Hardware execute instructions.

Tutorial Review

- **Problem Definition**: Usually, latency, memory, energy consumption and throughput will be the ultimate objective or constraint on “efficiency”. In the meantime, the intelligence level of AI needs to be retained.
  - **Measured metrics** are actually tested on platform (thus is platform-related), and directly correspond to the objectives / constraints.
  - **Proxy metrics** are estimated with only model specification. In practice, they are useful in diagnosis of the bottleneck and estimation of measured metrics.

- **Practical Pipeline**: We can estimate bottleneck modules and overall objectives, whether each module is compute or memory bound using some simple method (e.g., roofline model), then we can actually profile them (NVIDIA GPU: Nsight system & compute). Finally, we design method accordingly.

Tutorial Review



Future Directions

- **Application Requirements**
  - Long context (e.g., for complex reasoning)
  - Multi-modality input/output
  - Multi-model agentic application

- **Recent Active Directions**
  - Algorithm-level
    - Agentic generation (multi-agent collaboration)
    - Agent context engineering
  - Model-level
    - Latent reasoning
    - Efficient architecture design
  - System-level
    - Agent infrastructure

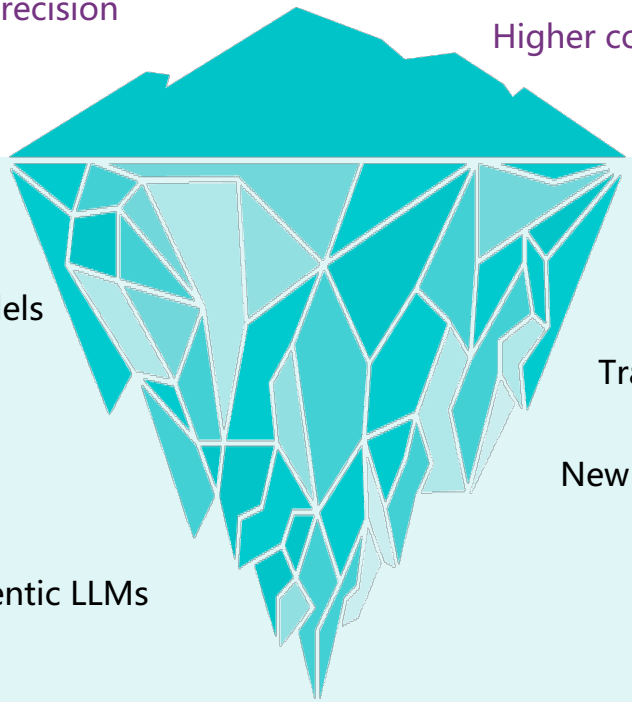
Future Directions

Higher precision

Higher compression rate: lower bits

Hardware-friendly algorithms

...



Compression of reasoning models

Data-mixture of QAT

RL with quantization

Training objectives v.s. LLM origins

The off-policy effect by quantization

New format: MXFP4 or NVFP4?

Efficient agentic LLMs

Efficient context engineering

Thank You !

Xuefei Ning[1], Guohao Dai[2,4], Haoli Bai[3], Lu Hou[3], Yu Wang[1], Qun Liu[3]

[1]Tsinghua University [2]Shanghai Jiao Tong University [3]Huawei [4]Infinigence-AI

Tutorial Website

https://haolibai.github.io/emnlp-2025-tutorial-efficiency/