

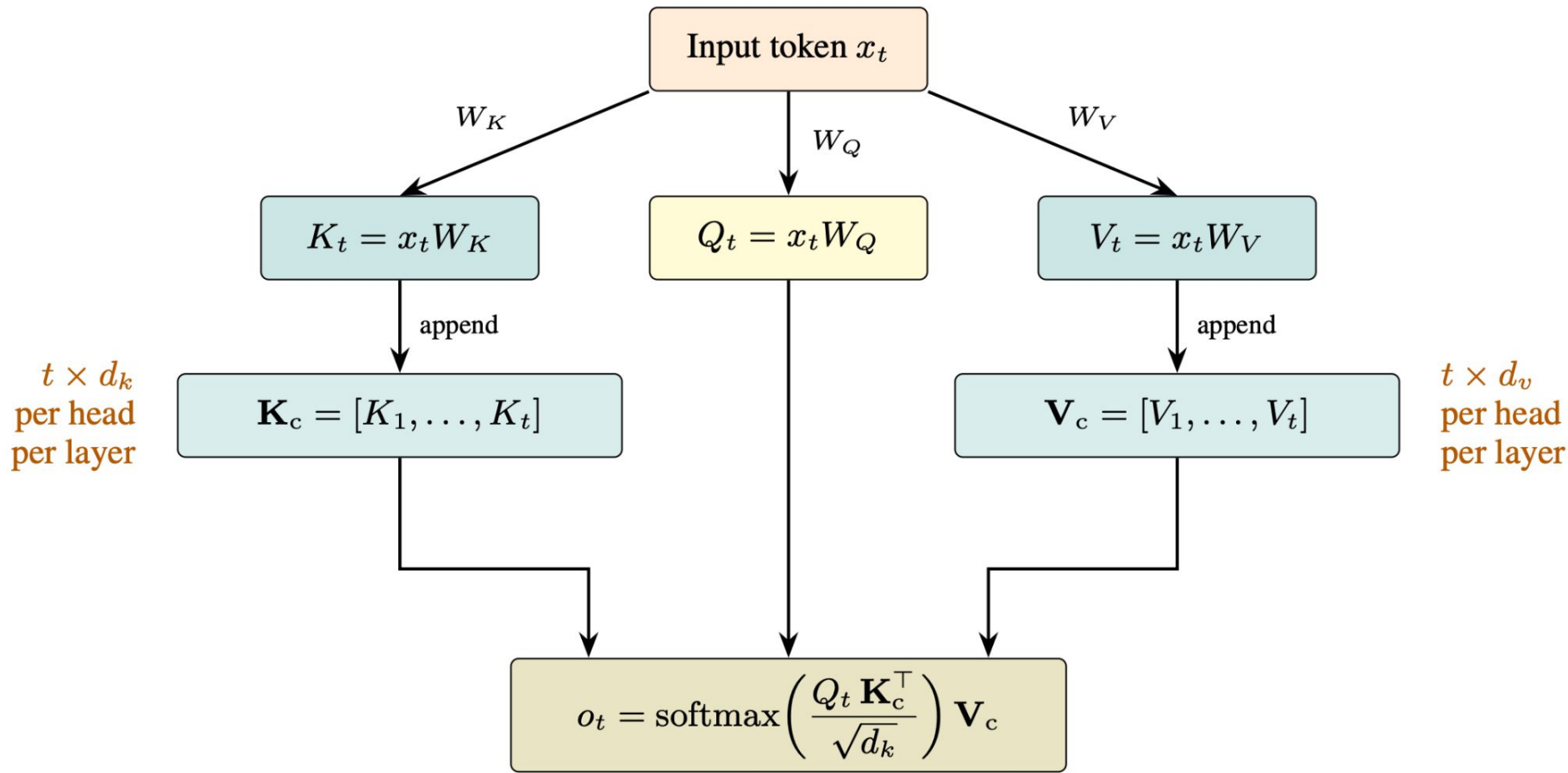
Kimi K3 in vLLM

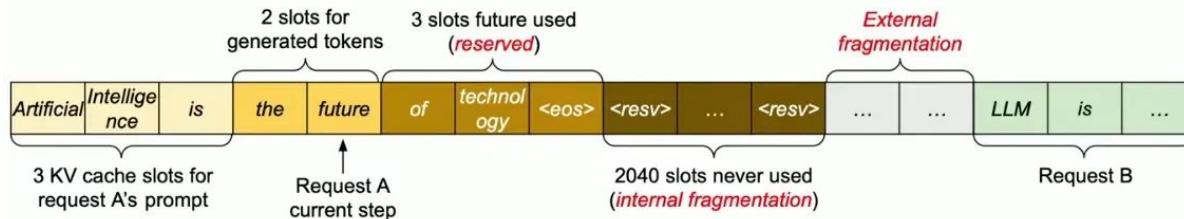


zhujiangyun@inferact.ai



**The High-Throughput and
Memory-Efficient open source
inference engine**

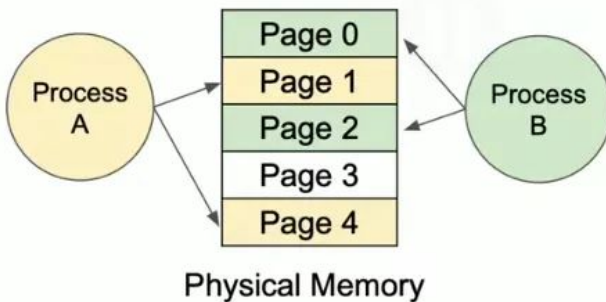




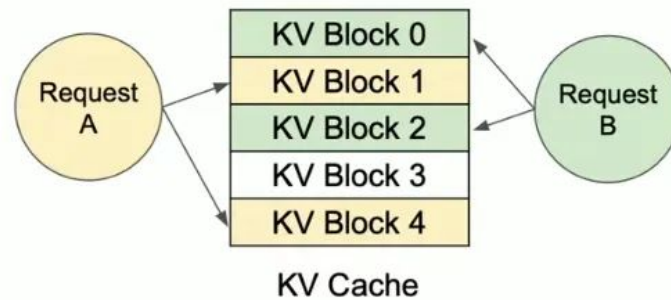
- **Internal fragmentation:** over-allocated due to the unknown output length.
- **Reservation:** not used at the current step, but used in the future
- **External fragmentation:** due to different sequence lengths.

Only **20–40%** of KV cache is utilized to store token states

Memory management in OS



Memory management in vLLM



Request
A

Prompt: "Alan Turing is a computer scientist"

Completion: "and mathematician renowned"

Logical KV blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	mathem tician
block 2	renowned			
block 3				

Block table

Physical block number	# Filled
7	4
1	4
5	1
-	-

Physical KV blocks

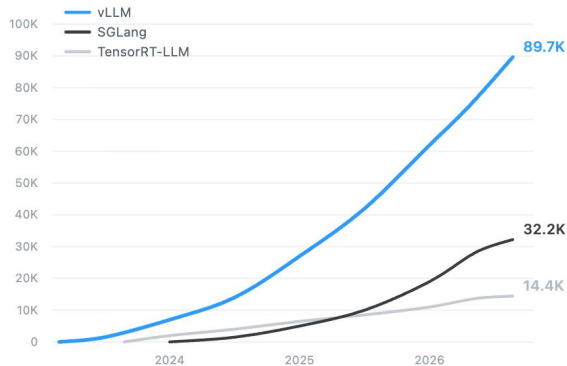
block 0				
block 1	computer	scientist	and	mathem atician
block 2				
block 3				
block 4				
block 5	renowned			
block 6				
block 7	Alan	Turing	is	a

Allocated on demand

The world's leading open inference engine

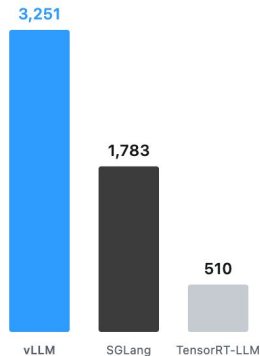
COMMUNITY

GitHub stars over time



Source: GitHub, August 2026

Contributors



ECOSYSTEM

APPLICATION LAYER



FRONTIER LABS



INFERENCE CLOUD



ECOSYSTEM PARTNERS



vLLM is the most used inference engine.

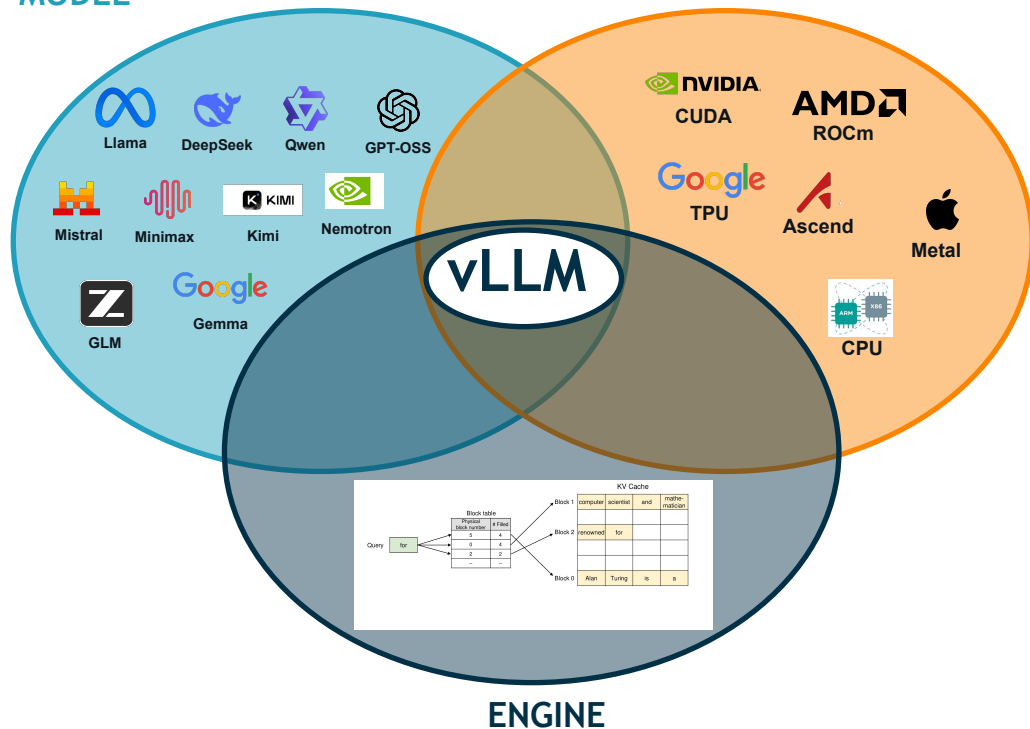
vLLM is the AI ecosystem default.

Model × Hardware × Engine = Intelligent Tokens

Each axis evolves rapidly. vLLM is the open intersection.

MODEL

HARDWARE



THE TAKEAWAY

Open-source coordination is the only sustainable way to keep pace with three rapidly evolving axes.

That's the gravity holding model vendors, hardware vendors, and engine developers in the same room.

Kimi K3 Architecture Overview

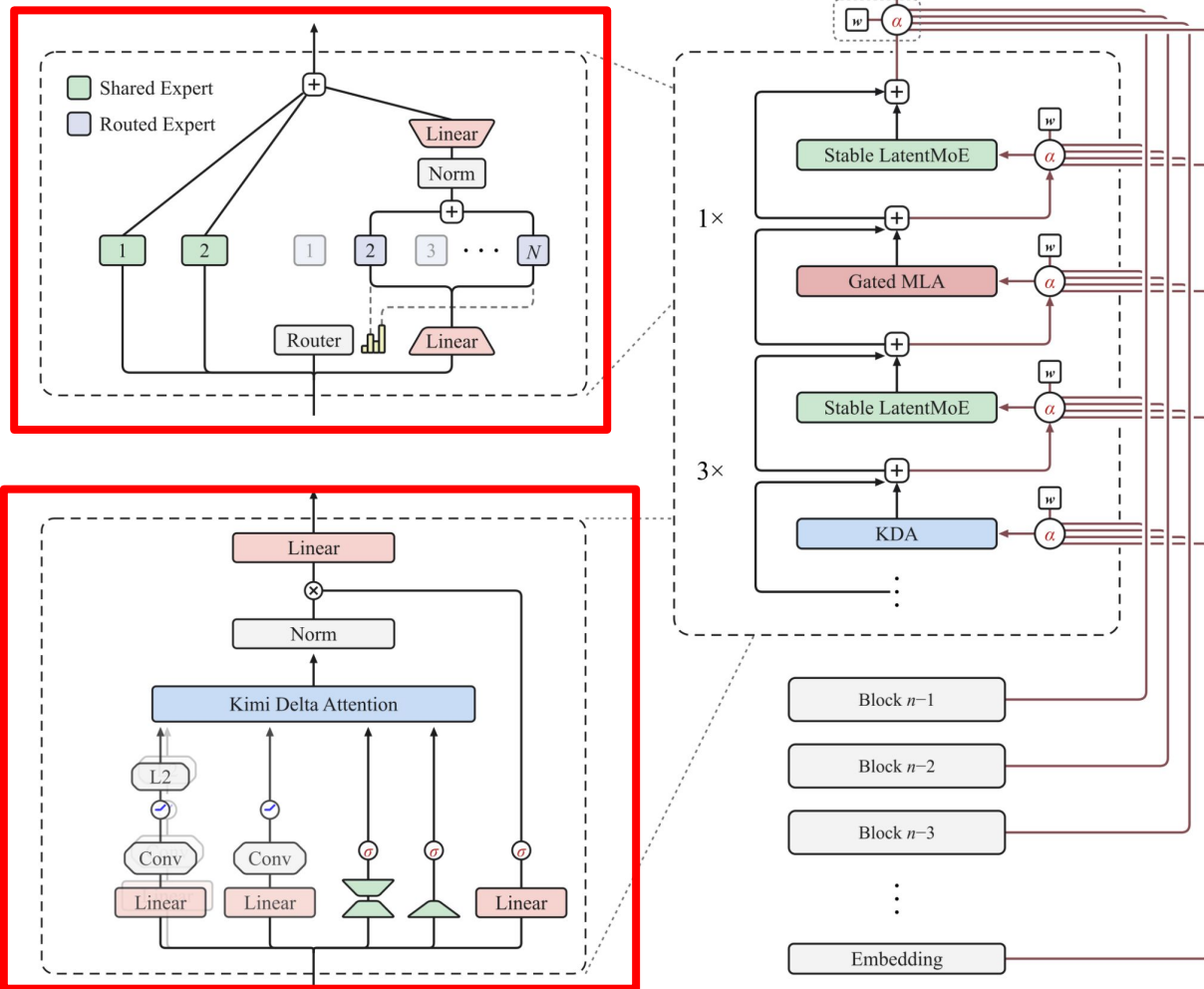
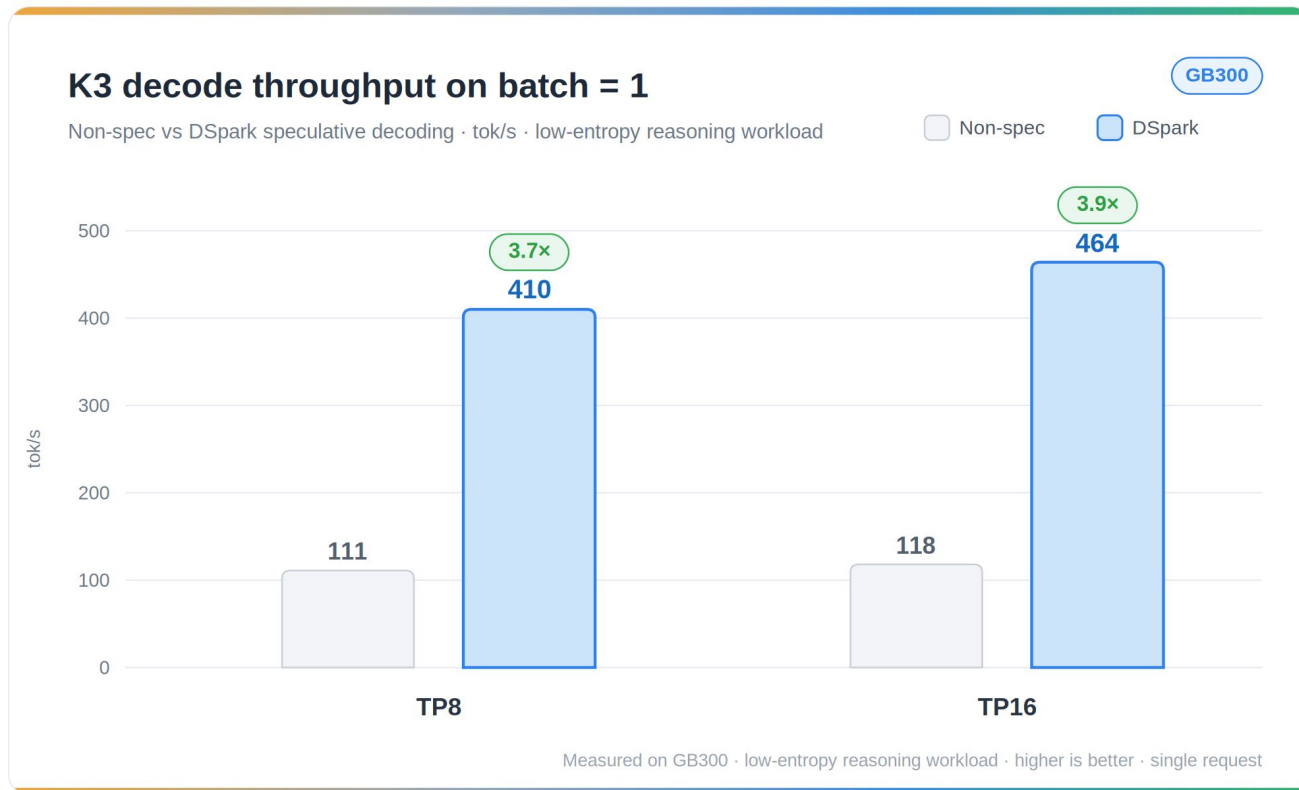


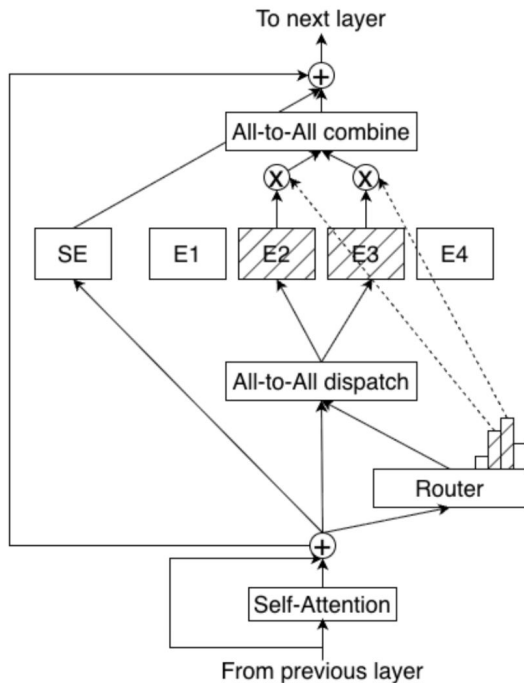
Table 1: Architectural comparison between Kimi K2 and Kimi K3.

	Kimi K2	Kimi K3	Δ
Architecture	MoE	MoE	–
#Layers	61	93	$\uparrow$ 52%
Total Parameters	1.04T	2.78T	$\uparrow$ 167%
Activated Parameters	32.6B	104.2B	$\uparrow$ 220%
Hidden Dimension	7,168	7,168	=
Latent MoE Dimension	–	3584 (0.5 $\times$)	–
MoE Hidden Dimension per Expert	2,048	3,072	$\uparrow$ 50%
Routed Experts	384	896	$\uparrow$ 133%
Experts Active per Token	8	16	$\uparrow$ 100%
Shared Experts	1	2	$\uparrow$ 100%
Attention Heads	64	96	$\uparrow$ 50%
Number of Dense Layers	1	1	=
Vocabulary Size	160K	160K	=
Training Context Length	128K	1M	8 $\times$
Attention Mechanism	MLA	Hybrid KDA–MLA	–
Activation Function	SwiGLU	SITU-GLU	–
Attention-Layer Composition	61 MLA	69 KDA + 24 MLA	–
Number of MTP Layers	1 layer	1 layer	=
Total Parameters of ViT	–	401M	–
#ViT Layers	–	27 layers	–
Patch Size of ViT	–	14	–
#Attention Heads of ViT	–	12	–

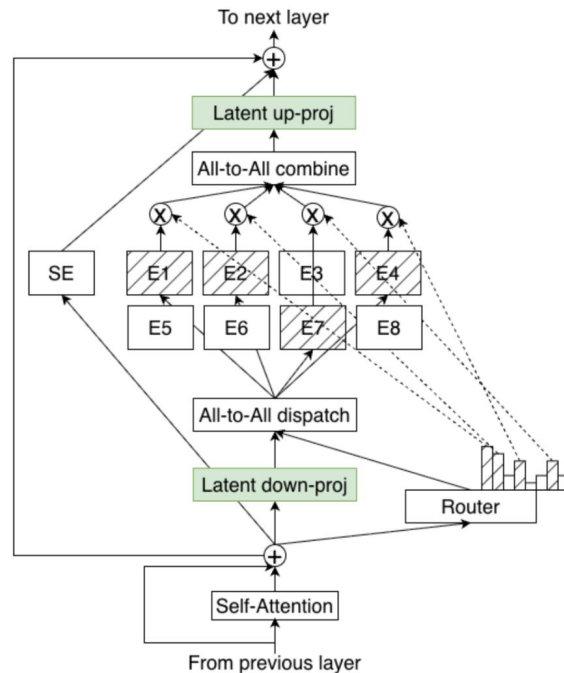
Performance



LatentMoE: reduce communication



(a) Standard MoE architecture.



(b) LatentMoE architecture.

Kimi Delta Attention

3 Kimi Delta Attention: Improving Delta Rule with Fine-grained Gating

We propose Kimi Delta Attention (KDA), a new gated linear attention variant that refines GDN's scalar decay by introducing a fine-grained diagonalized gate $\text{Diag}(\alpha_t)$ that enables fine-grained control over memory decay and positional awareness (as discussed in §6.1). We begin by introducing the chunkwise parallelization of KDA, showing how a series of rank-1 matrix transformations can be compressed into a dense representation while maintaining stability under diagonal gating. We then highlight the efficiency gains of KDA over the standard DPLR (*Diagonal-Plus-Low-Rank*) formulation [30, 71].

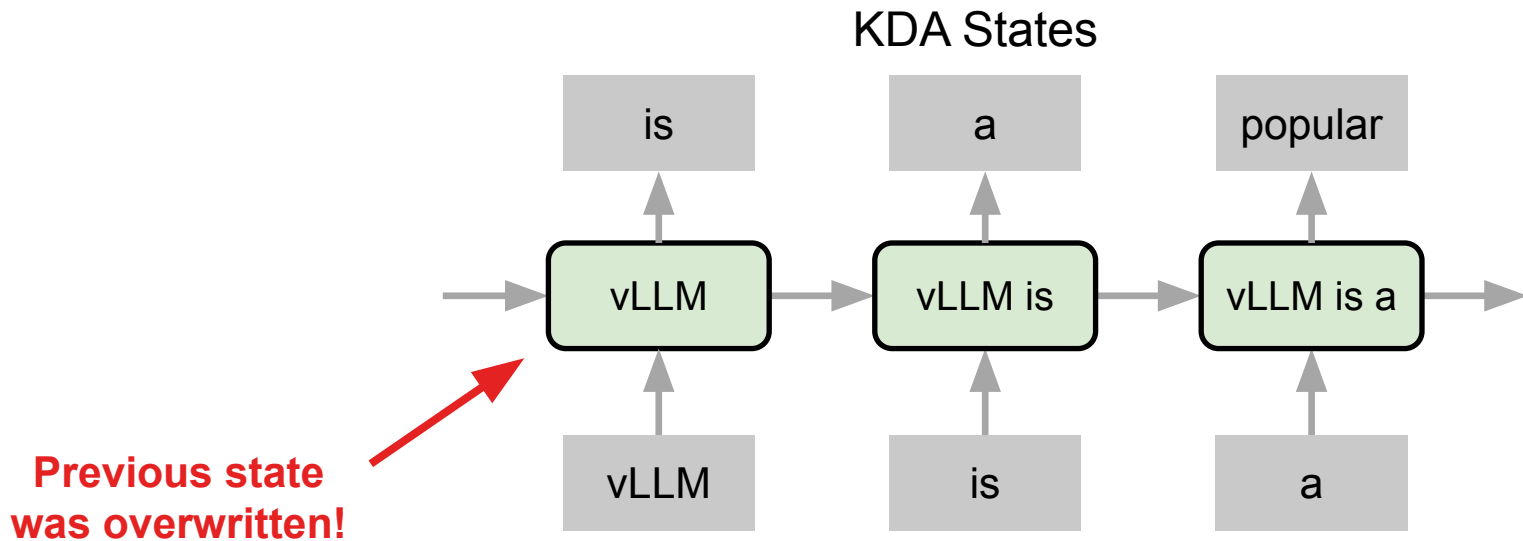
$$\mathbf{S}_t = (\mathbf{I} - \beta_t \mathbf{k}_t \mathbf{k}_t^\top) \text{Diag}(\alpha_t) \mathbf{S}_{t-1} + \beta_t \mathbf{k}_t \mathbf{v}_t^\top \in \mathbb{R}^{d_k \times d_v}; \quad \mathbf{o}_t = \mathbf{S}_t^\top \mathbf{q}_t \in \mathbb{R}^{d_v} \quad (1)$$

The diagram illustrates the matrix operations in the KDA equation. It shows the decomposition of the update term into a difference of two rank-1 matrices and a new rank-1 matrix, all multiplied by a diagonal gate matrix. The diagram uses colored squares to represent matrix elements and arrows to show the flow of operations.

<https://zhiyuan1i.github.io/posts/kda-mathematics>

Challenging

- states can't be rolled back to represent a prefix



SSM's modeling win complicates their systems win!

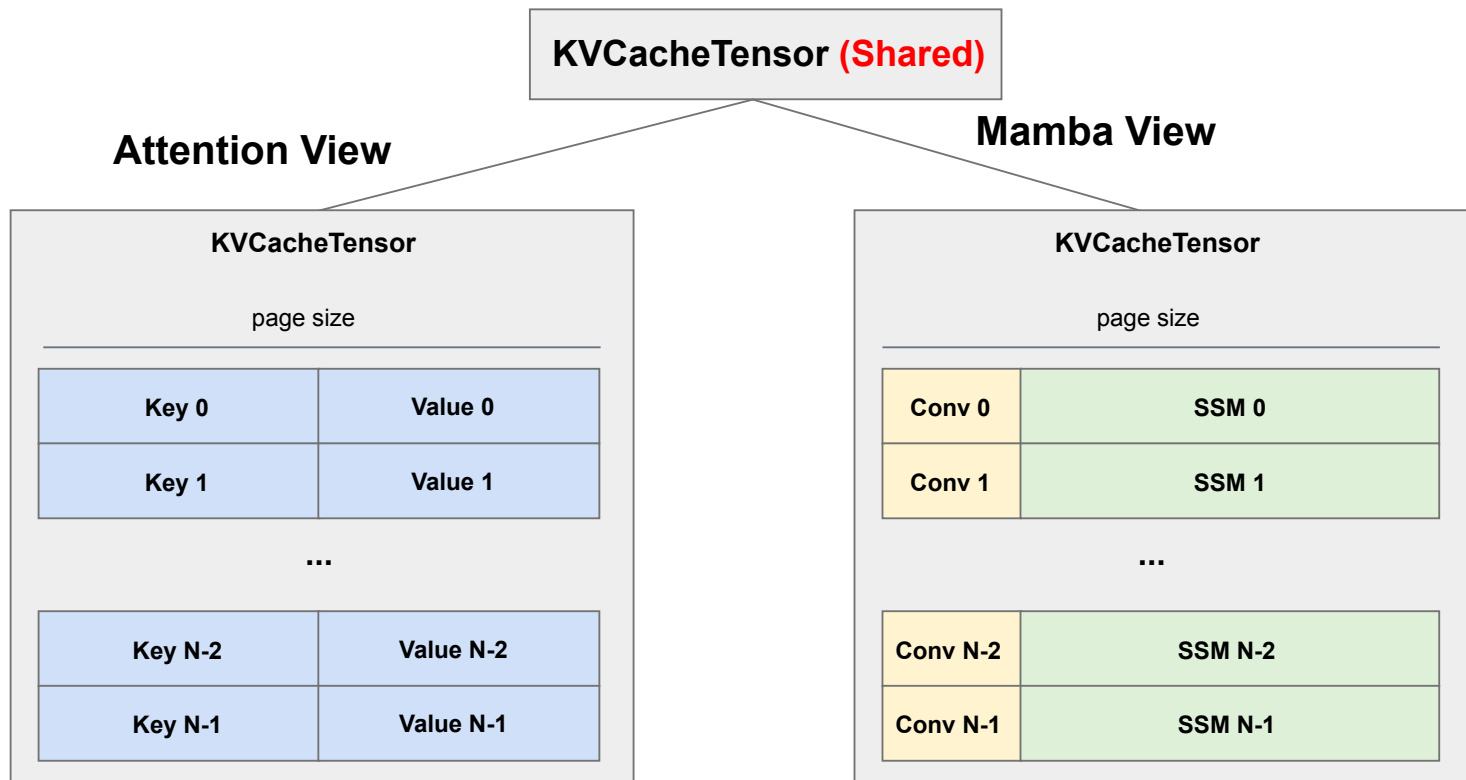
Challenging

- This breaks the assumption of vLLM: the kv cache grows continuously and is append-only.
- How to do:
 - **memory management**
 - **prefix caching**

KV Cache

vLLM	is	a	popular
------	----	---	---------

How to manage: Hybrid Memory Allocator



Hybrid Memory Allocator

- Split the memory into pages whose size is the **Least Common Multiple (LCM)** of all possible allocation sizes
- For each type, further split one LCM page into multiple small pages of specific size with **no fragmentation**

Data Type 1 (full attn)

Size X = 1 KB

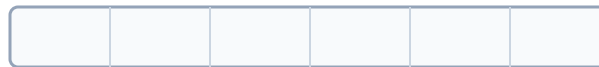
Data Type 2 (sliding window)

Size Y = 2 KB

Data Type 3 (mamba)

Size Z = 3 KB

LCM Page (Size = $\text{LCM}(X, Y, Z) = 6 \text{ KB}$)



LCM Page



6 × 1 KB



3 × 2 KB



2 × 3 KB

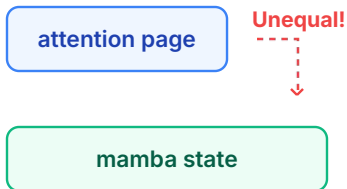
Page Size Alignment

Aligning page sizes between Attention and Mamba models to enable hybrid KV caching

01

Calculate Original Sizes

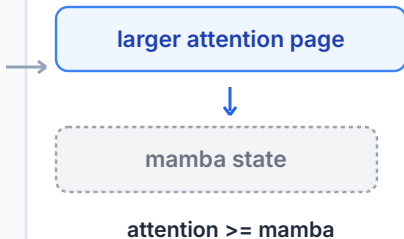
Calculate original size of both page types, which are usually unequal



02

Adjust block_size

Make an attention page large enough to accommodate Mamba



03

Padding Mamba

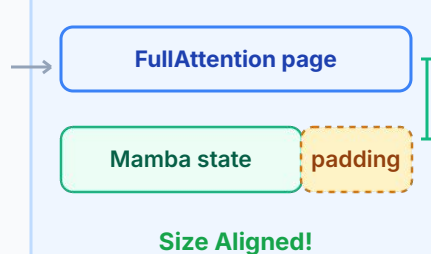
If still unequal, pad the Mamba page to match attention size



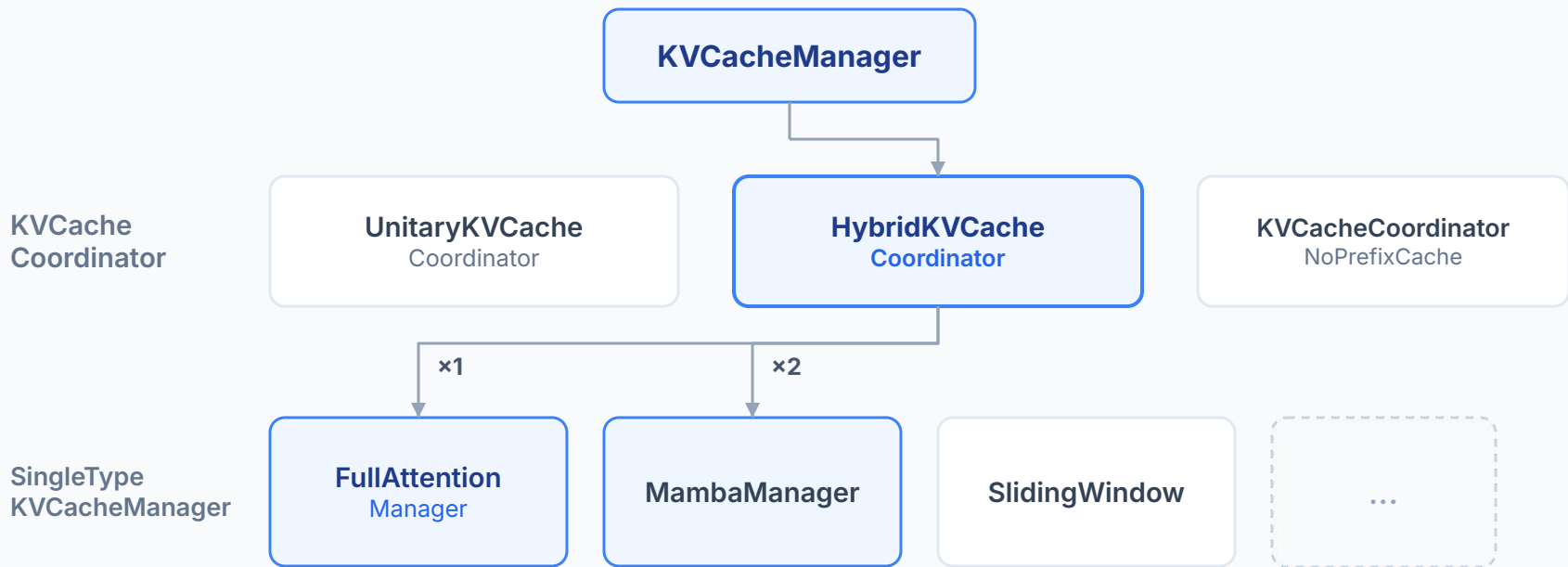
04

Aligned Result

Ensure identical page_size_bytes for each KV cache group



Hybrid KV Cache Manager



Block-aligned Scheduling

Prefix caching for Mamba / linear-attention models in vLLM

1 Two prerequisites

States must be bound to a **Block Hash** to be reused on a cache hit

Linear attention only emits the **final state** — it cannot be rolled back per token

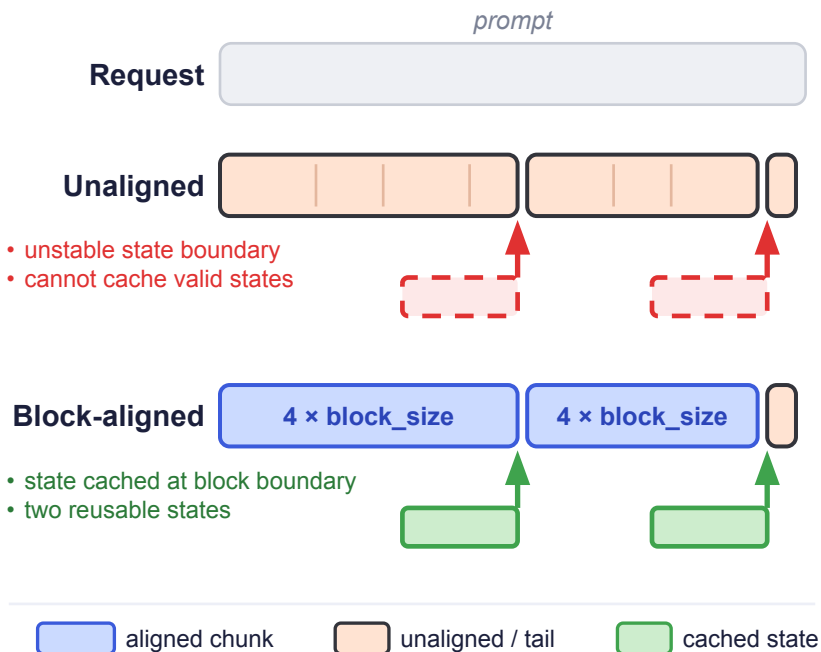
2 Solution

Align every prefill chunk to a **multiple of block_size**, so each state lands on a block boundary — only the **last chunk** is unconstrained

3 Key advantages

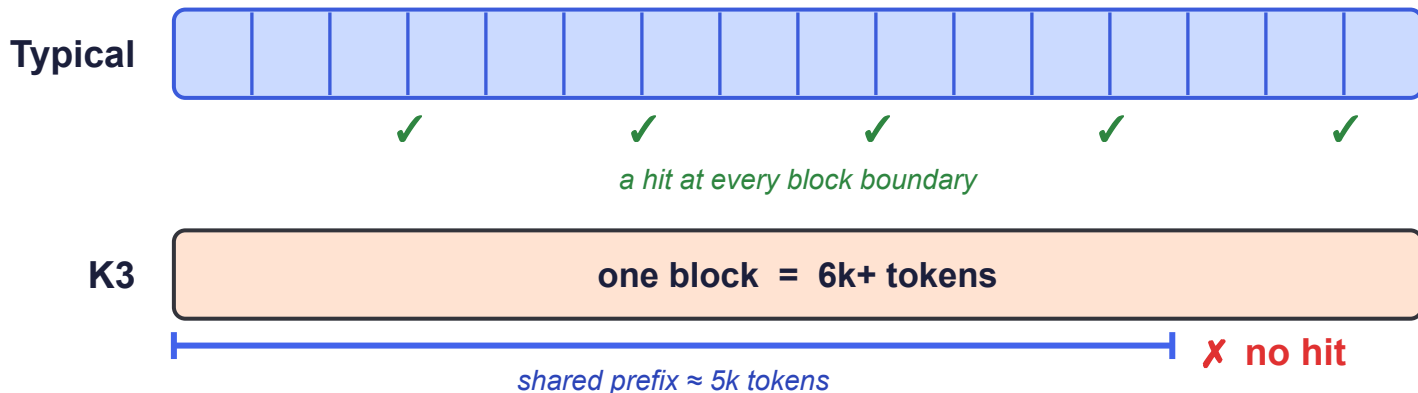
Scheduler-driven — **zero model-layer changes**

Balances **GPU memory** footprint against **cache-hit rate**

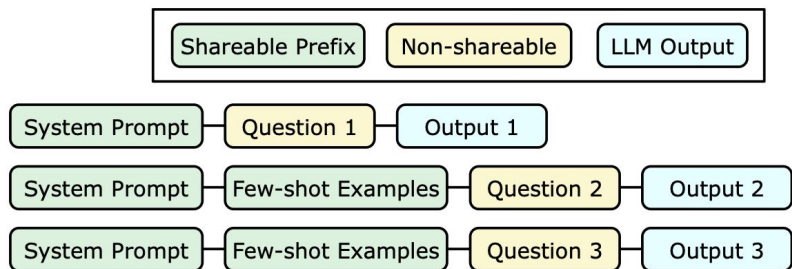


Weakness of existing design

- We need to align the page sizes across different KV cache groups.
- As a result, if one cache is much larger than another, the smaller cache may require a very large block size to match it.
- K3: KDA state is about 600x larger than MLA cache, the block size of K3 is 6k+ in DP case, this makes prefix cache very inefficient

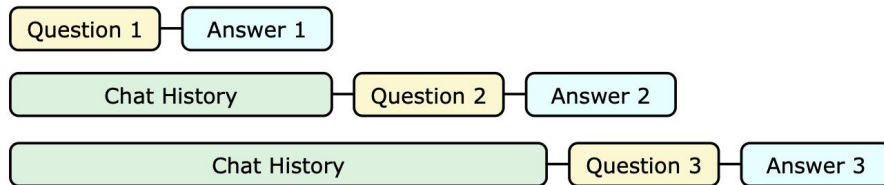


What's the most valuable cache



(a) System prompt and few-shot prompting

System Prompt



(b) Multi-turn conversation (e.g., ChatGPT)

The boundary of each turn

Partial Cache Hit: Decouple Cache-Hit Granularity from Block Size

vLLM RFC #45702 (proposed by Moonshot) — introduce `hash_block_size` to hit a partial cache block **Example: `block_size = 6`, `hash_block_size = 2`**

Request A prompt = a b c d e f g h (written to cache; registers both kinds of hash keys)



$H0 = \text{hash}([a\ b\ c\ d\ e\ f])$ full-block key

$P0 = \text{hash}(\text{parent}=H0, [g\ h])$ partial-tail key (at most one, for the final tail of a request)

Request B (before) prompt = a b c d e f g h x y — full-block (`block_size`) matching only



Recompute 4

6 tokens reused

✗ Second full-block key ($H1$) not cached $\Rightarrow$ only block 0 is reusable; g h x y all recomputed from token 6

Request B (proposed) after the first full-block miss, probe partial-tail keys inside that block

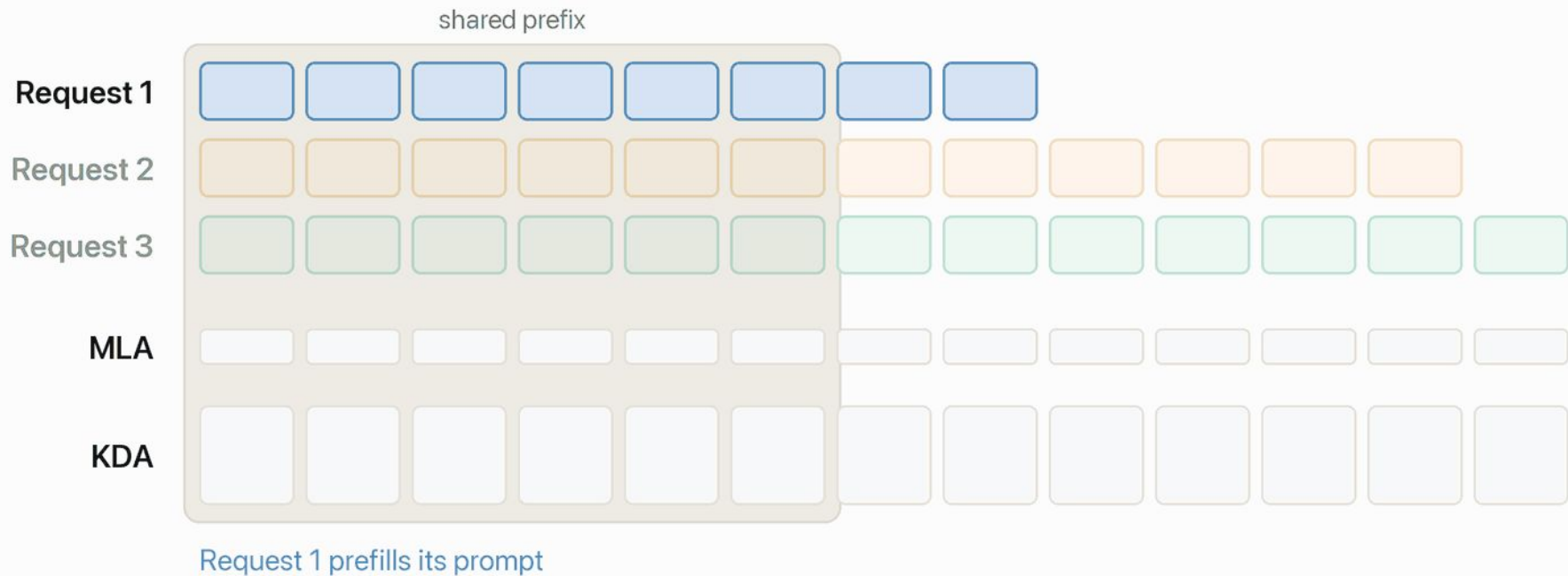


Recompute 2

8 tokens reused

✓ Hit (`hit_length = 8`) $\Rightarrow$ allocate block_B, **Copy-on-Write** [g h], recompute only x y; $H1$ replaces $P1$ once the block fills

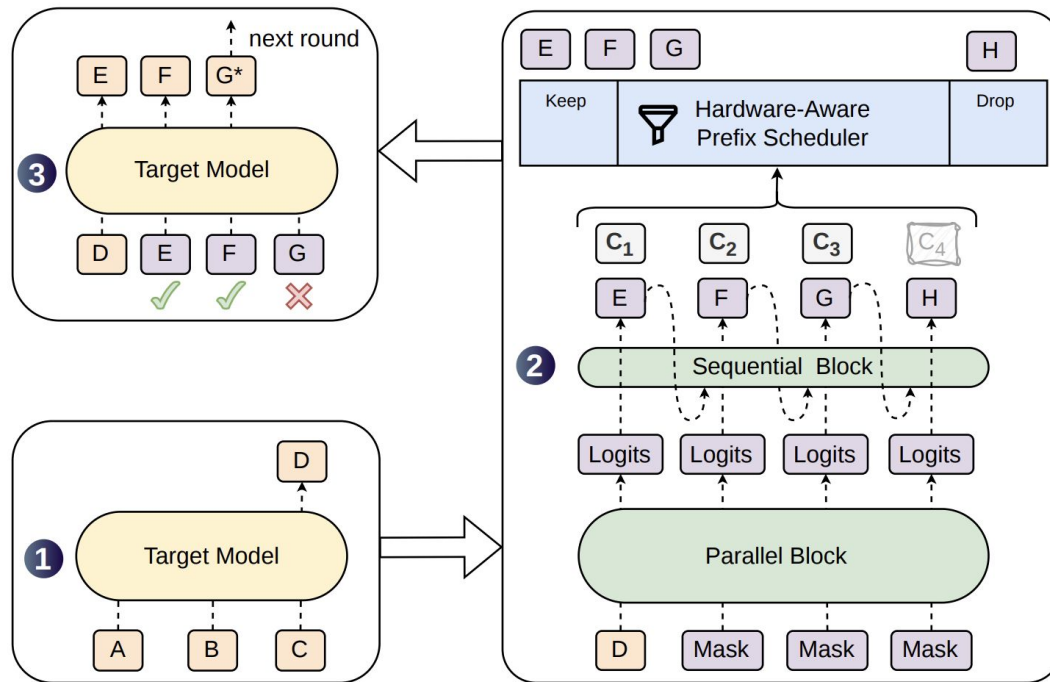
Marconi-style selective retention



Others

- Tensor reuse may lead to NaN, need to zero block before reusing
- Zeroing and RDMA write has data race when using PD

Dspark: Inferact/Kimi-K3-DSpark



Low Latency Optimization

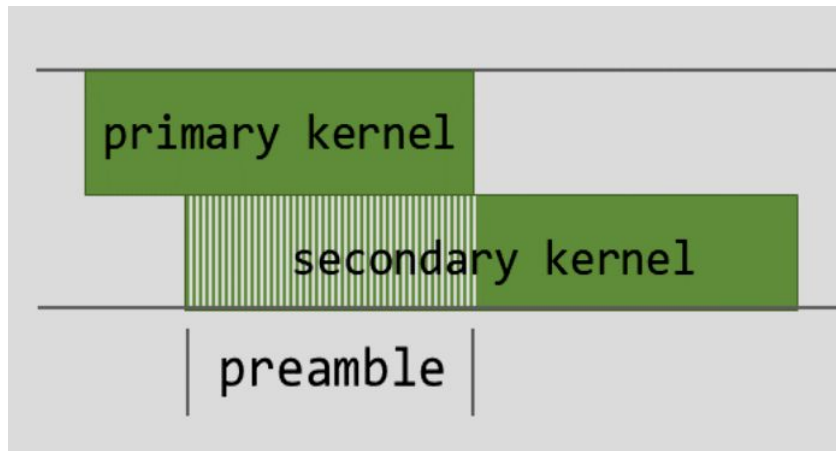
- Aggressively Using Programmatic Dependent Launch (PDL)
- Tail MoE optimizations

Programmatic Dependent Launch

- Before PDL, two kernels are launched sequentially
- After PDL, two kernels can be partial overlapped.



Before



After

Reference

- <https://vllm.ai/blog/2026-07-27-k3>
- <https://recipes.vllm.ai/moonshotai/Kimi-K3>
- <https://mlsys.org/virtual/2025/poster/3260>
- <https://www.kimi.com/blog/kimi-k3>
- <https://arxiv.org/abs/2601.18089>
- <https://github.com/vllm-project/vllm/pull/50000>
- <https://zhiyuan1i.github.io/posts/kda-mathematics>
- <https://github.com/MoonshotAI/Kimi-Linear>
- <https://github.com/MoonshotAI/Kimi-K3>
- <https://huggingface.co/Inferact/Kimi-K3-DSpark>

THANK YOU



inferact.ai

vllm.ai