



Efficient LLM Inference

Jie Hu

Kun Yuan

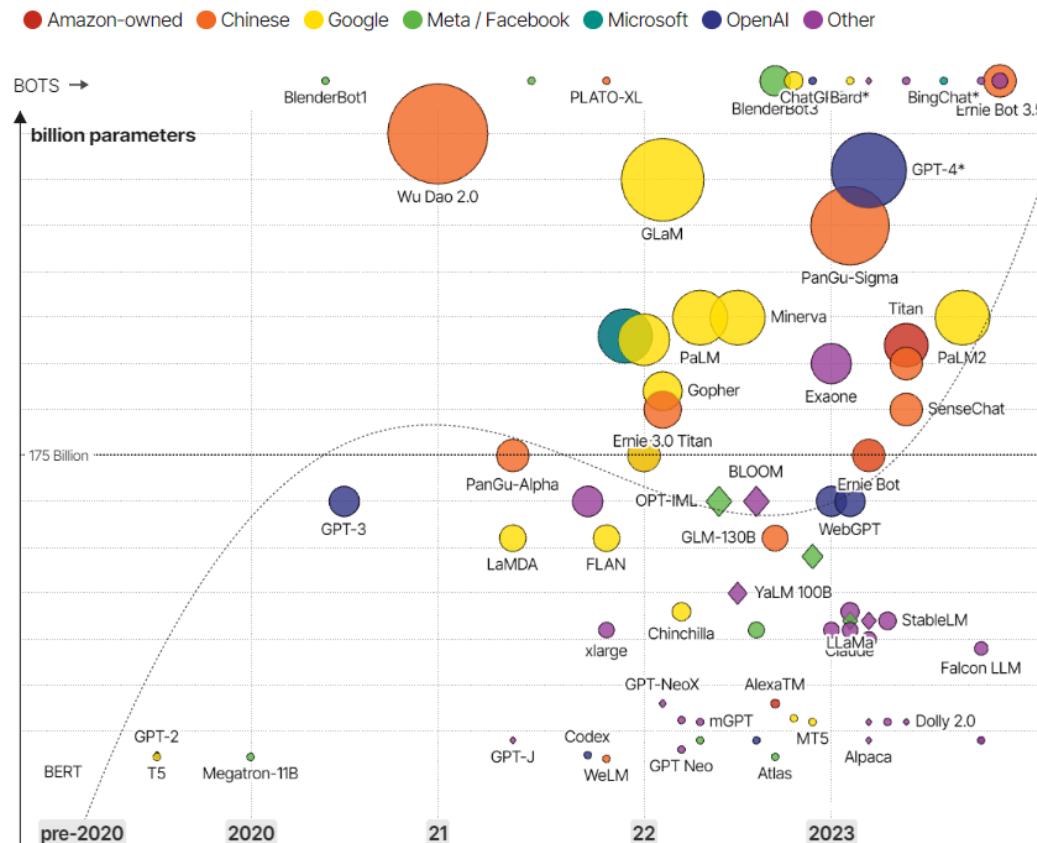
Peking University

Table of Contents

- Motivation
- Inference process of LLMs
- KV cache compression
 - MQA, GQA and MLA
 - Attention Sparsity
 - Quantization
- Speculative decoding
- Efficient management: vLLM

Why we need efficient inference

- LLM sizes have increased rapidly.



- Deployment Concerns – **Inference latency, Throughput, Cost, etc.**
- For a LLaMA-2-70B model
 - Gigantic model size : 140GB for weights (FP16)
 - Huge inference cache : 160GB for KV cache
- For LLaMA-13B and moderate-size inputs, 1 A100 can process < 1 requests per second.

Attention

$$\text{输入 } X \times \text{WQ} = Q$$

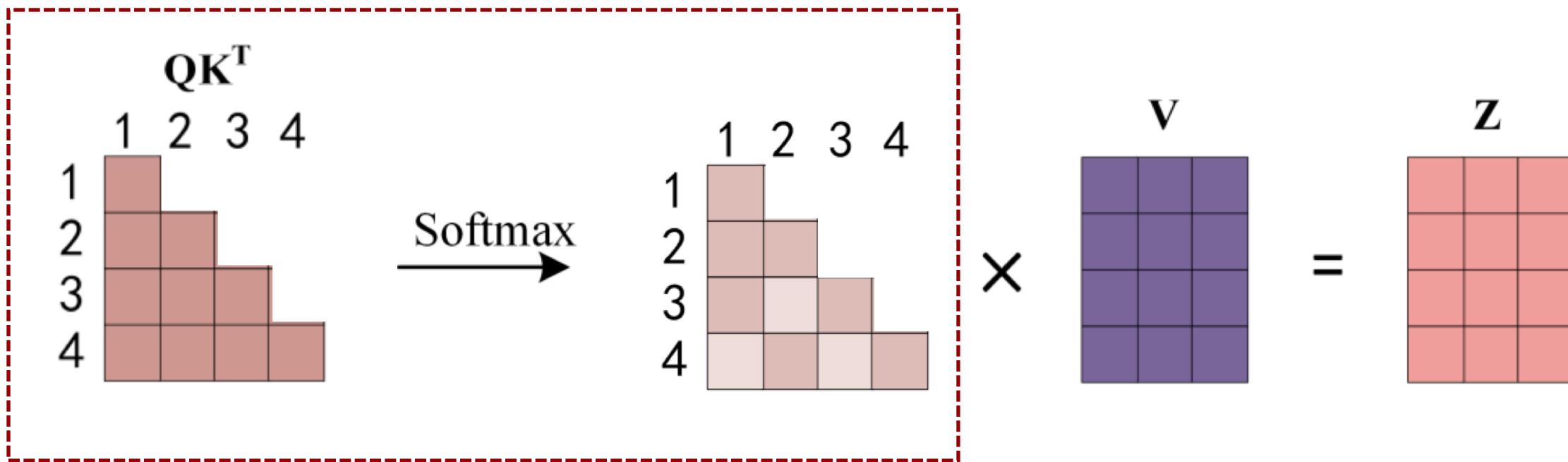
The diagram illustrates the multiplication of two matrices. On the left, a blue square grid labeled "输入 X" represents the input matrix. To its right is a yellow square grid labeled "WQ" representing the weight matrix. A large "X" symbol between them indicates multiplication. To the right of the multiplication result is another yellow square grid labeled "Q", representing the output matrix.

$$\text{输入 } X \times \text{WK} = K$$

$$\text{输入 } \mathbf{X} \quad \times \quad \mathbf{WV} \quad = \quad \mathbf{V}$$

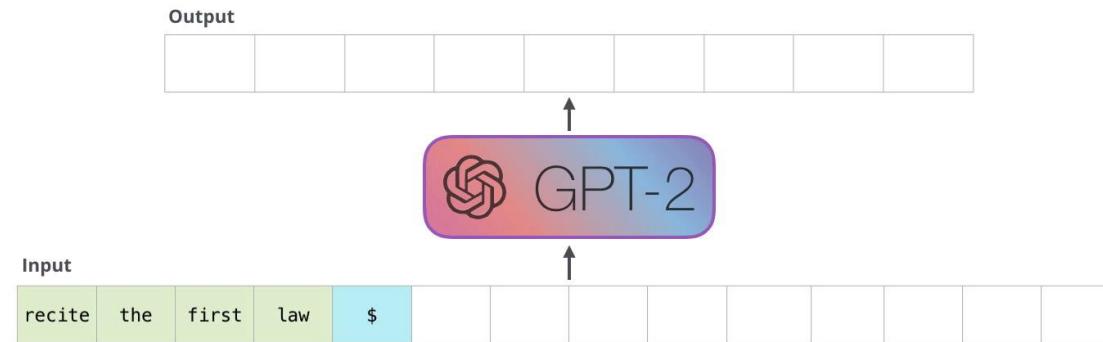
$$\begin{array}{c}
 Q \\
 \times \\
 K^T
 \end{array}
 \quad = \quad
 \begin{array}{c}
 QK^T \\
 \boxed{1 \ 2 \ 3 \ 4} \\
 \boxed{1 \ 2 \ 3 \ 4} \\
 \boxed{1 \ 2 \ 3 \ 4} \\
 \boxed{1 \ 2 \ 3 \ 4}
 \end{array}$$

Attention



Inference process of LLMs

➤ Auto-regressive decoding



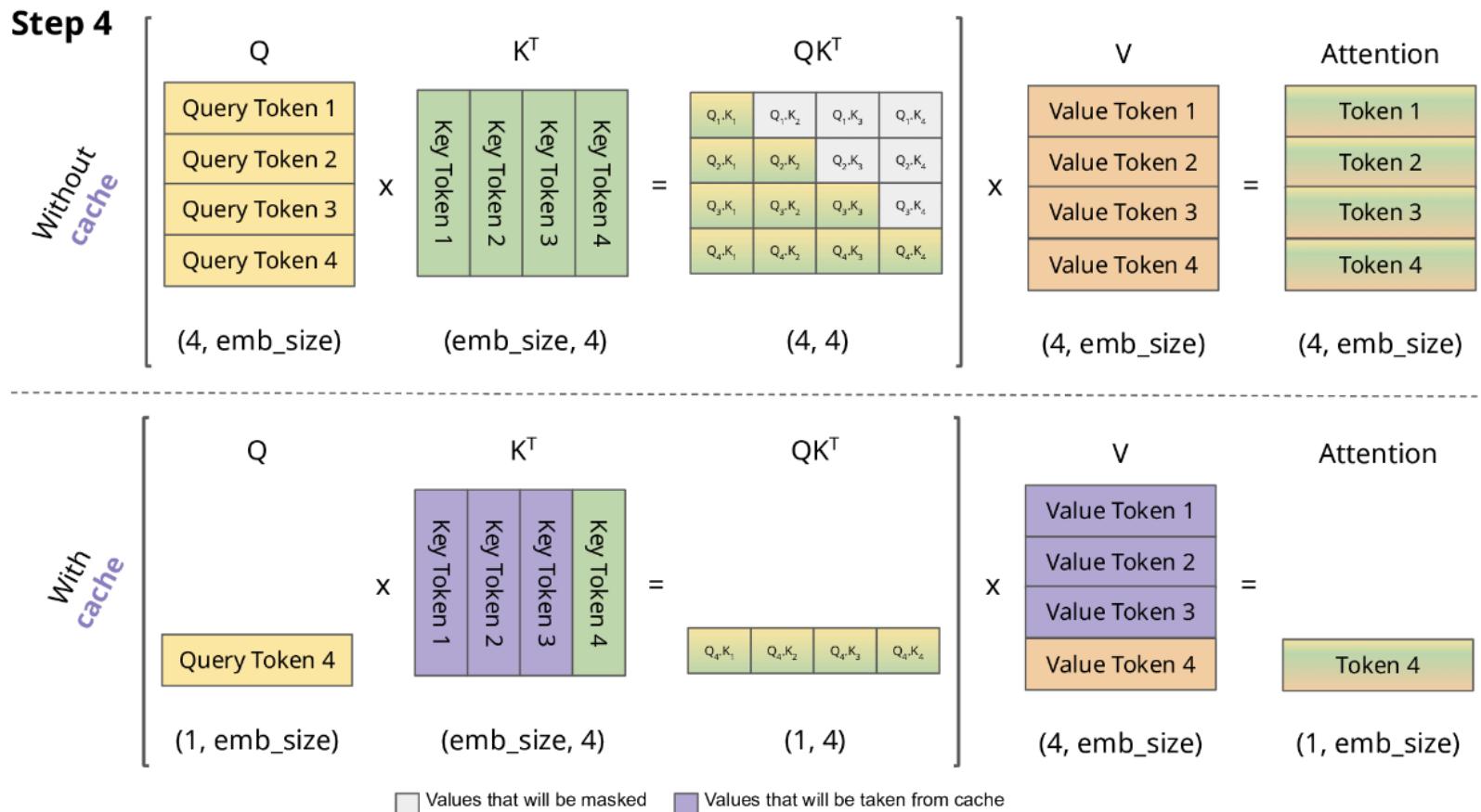
Source: <https://jalammar.github.io/illustrated-gpt2/>

After generating the first token, the model takes the prompt along with the newly generated token, computes the KV values again, and generates the next token.

- This process repeats for every subsequent token until maximum sequence length is reached or the stopping criteria is met.
- If the model needs to generate 500 tokens, this step-by-step procedure will be repeated 500 times.

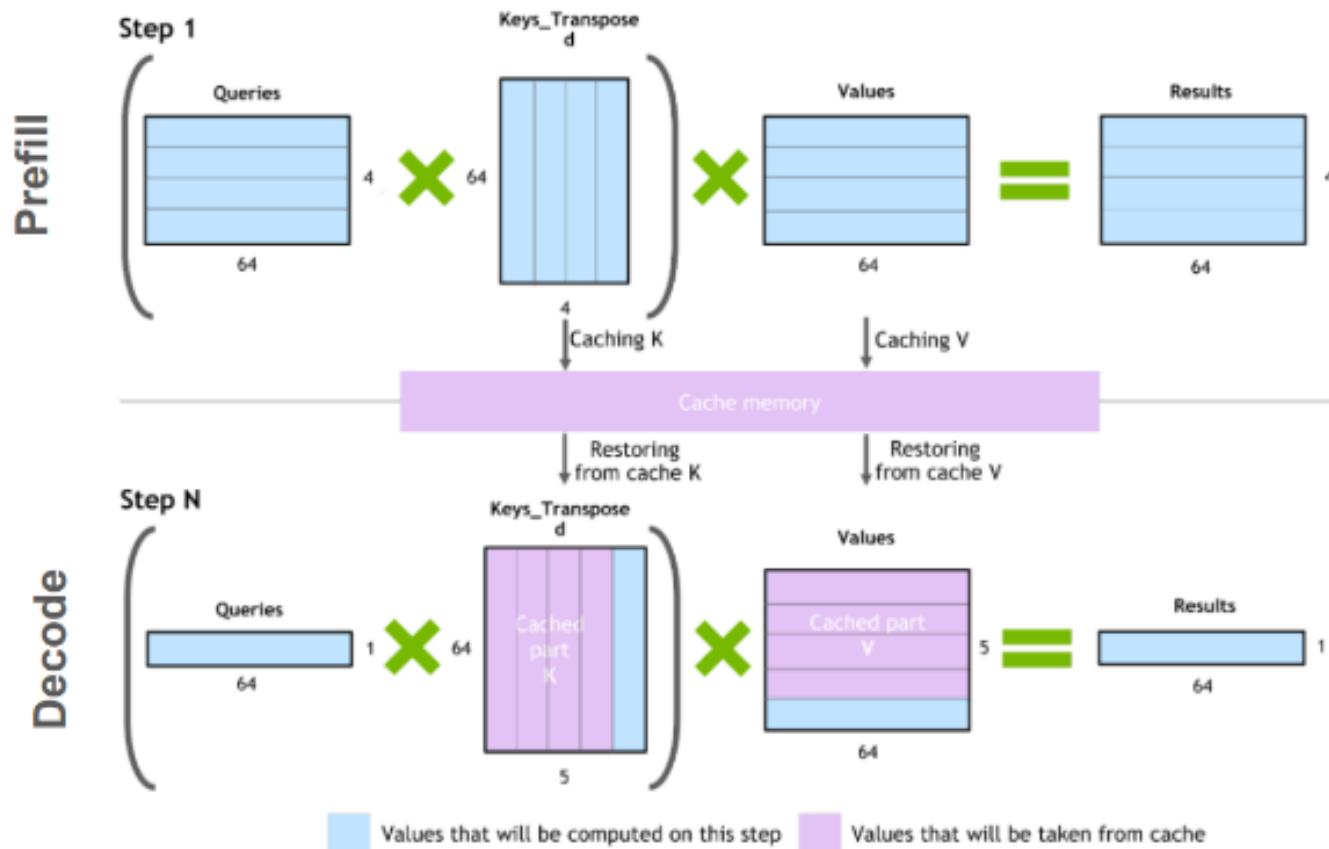
Key-Value Cache

- KV states for previously generated tokens will be cached to avoid recomputation. It only needs to compute the KV values for the most recently generated token, rather than entire sequence.



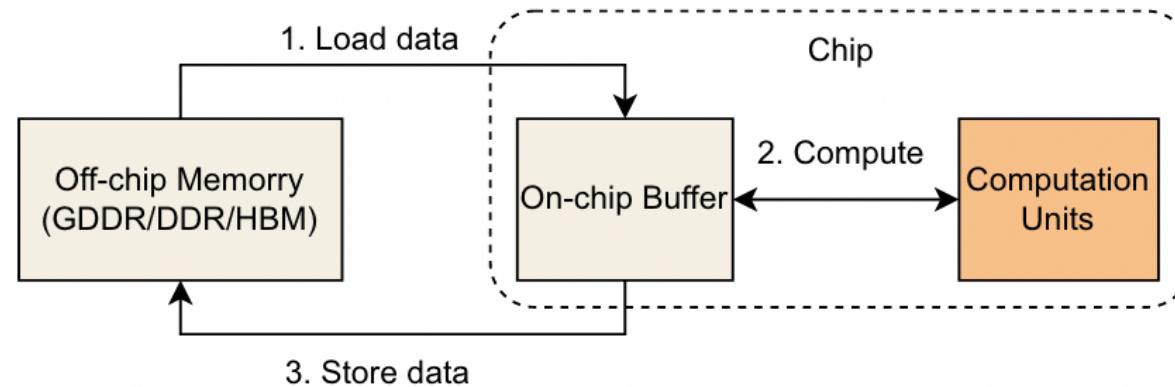
Key-Value Cache

$(Q * K^T) * V$ computation process with caching



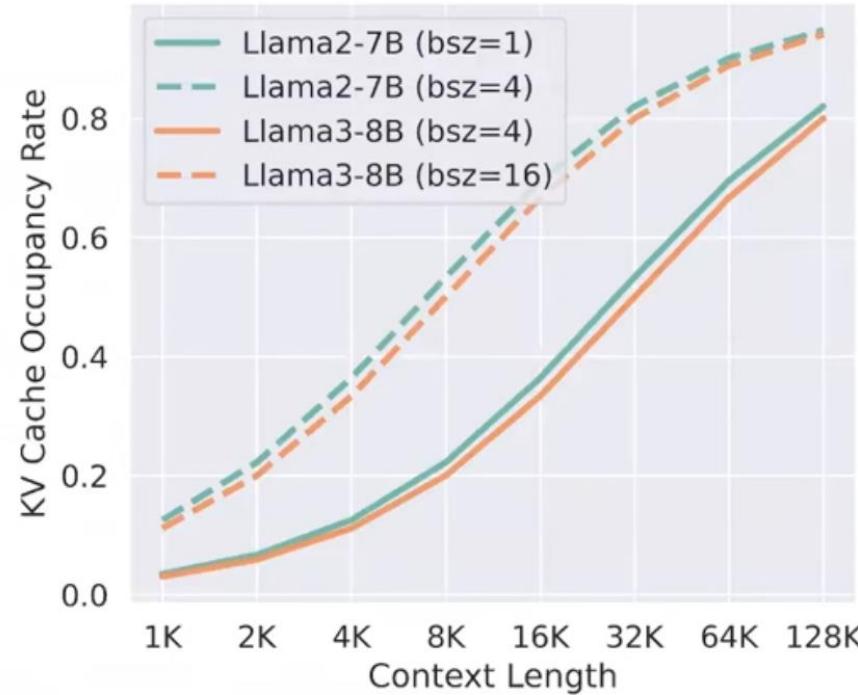
- Two Phases of LLM inference:
 - **Prefill**: LLM calculates and stores the keys and values of the initial input tokens, and generates the first new token.
 - **Decode**: LLM generates the output tokens one by one fetching KV from cache memory.

Challenge



- Prefill: compute bound
 - matrix-matrix multiplication, highly parallel, high usage of GPU
- Decode: **memory bound**
 - matrix-vector multiplication, underutilize GPU
 - The **speed at which the data (weights, keys, values, activations) is transferred** to the GPU from memory dominates the latency, not how fast the computation actually happens.

Challenge

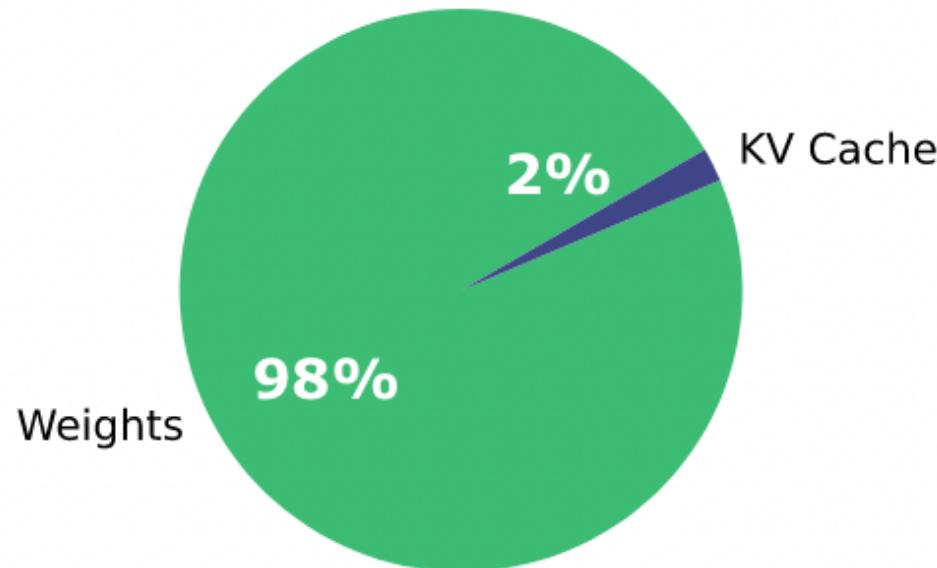


- Cache size:
 $2 \times \text{precision in bytes} \times \text{batch size} \times \text{sequence length} \times \text{number of layers} \times \text{embedding dimension}$

It limits the **throughput** that can be served, and poses challenges for **long-context** inputs.
TriForce: Lossless Acceleration of Long Sequence Generation with Hierarchical Speculative Decoding. H. Sun et al.

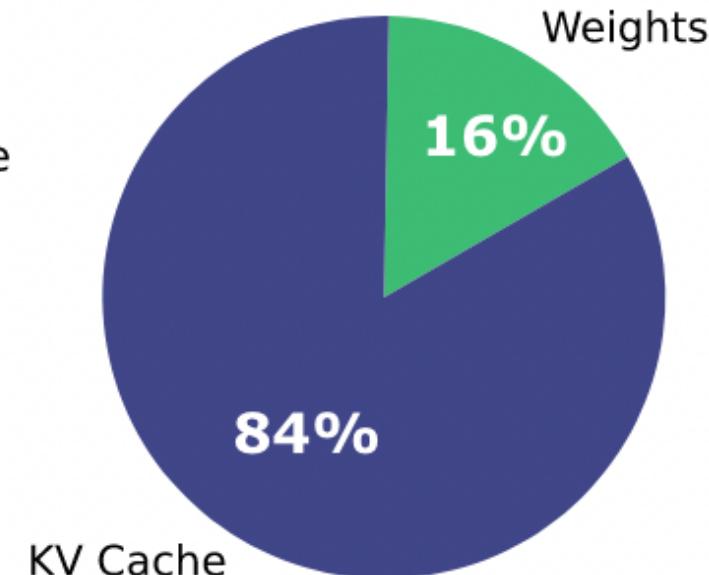
KV Cache is the bottleneck in long-context inference

SeqLen 512, Batch Size 1



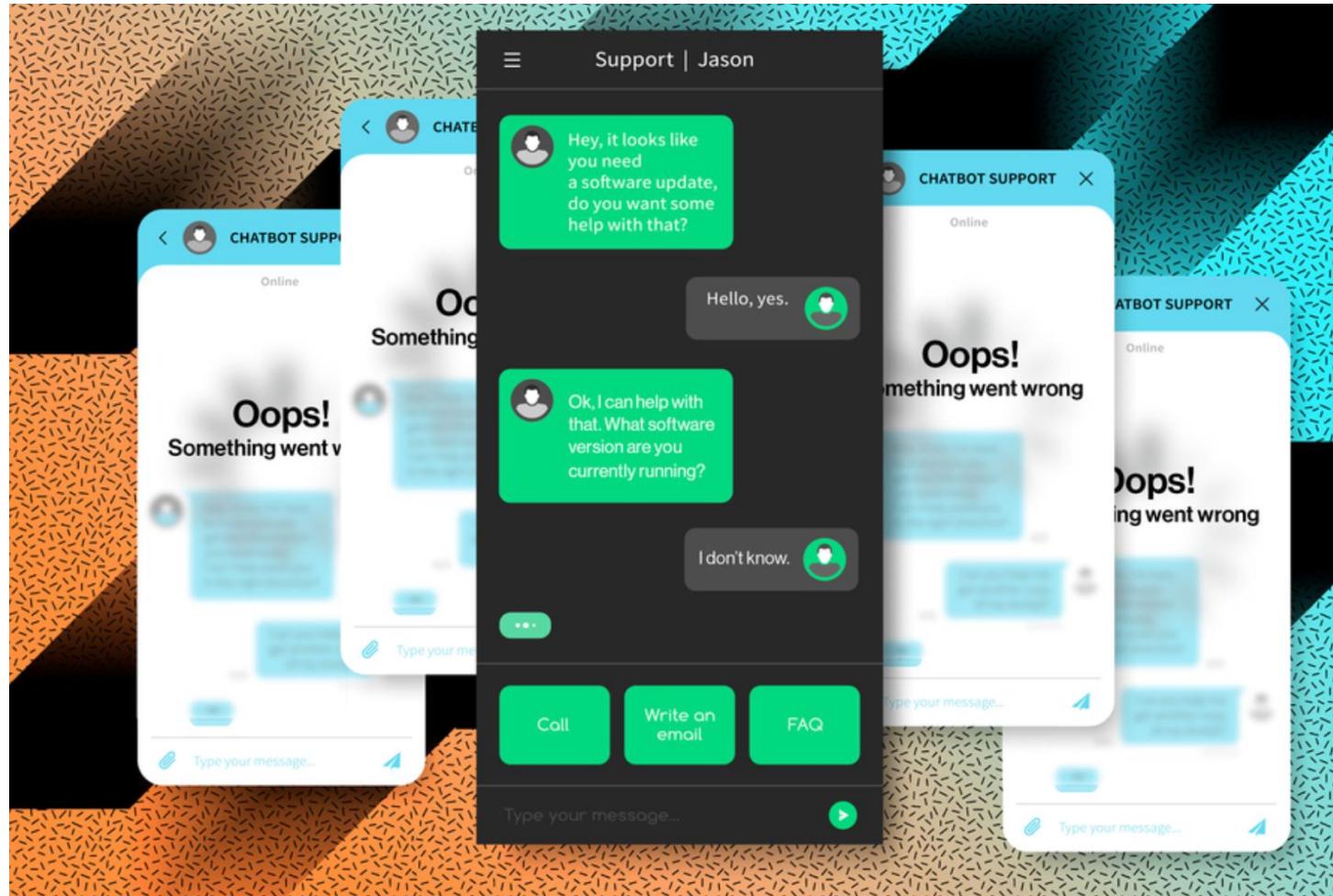
Short sequence length
Weights are the bottleneck

SeqLen 128K, Batch Size 1



Long Sequence Lengths
KV Cache is the bottleneck

Chatbot will crash if KV cache is out of memory



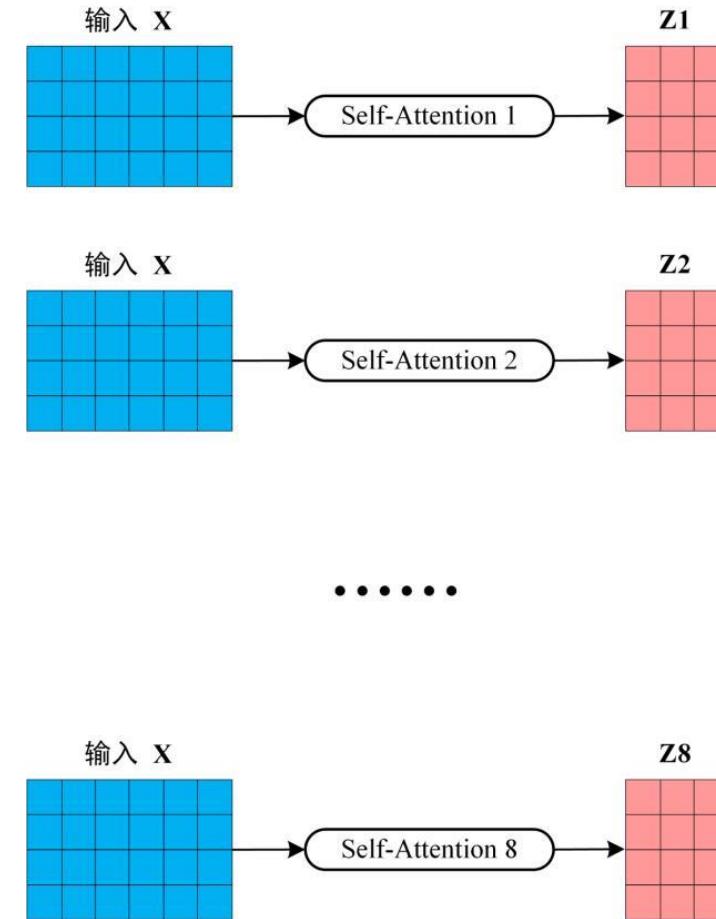
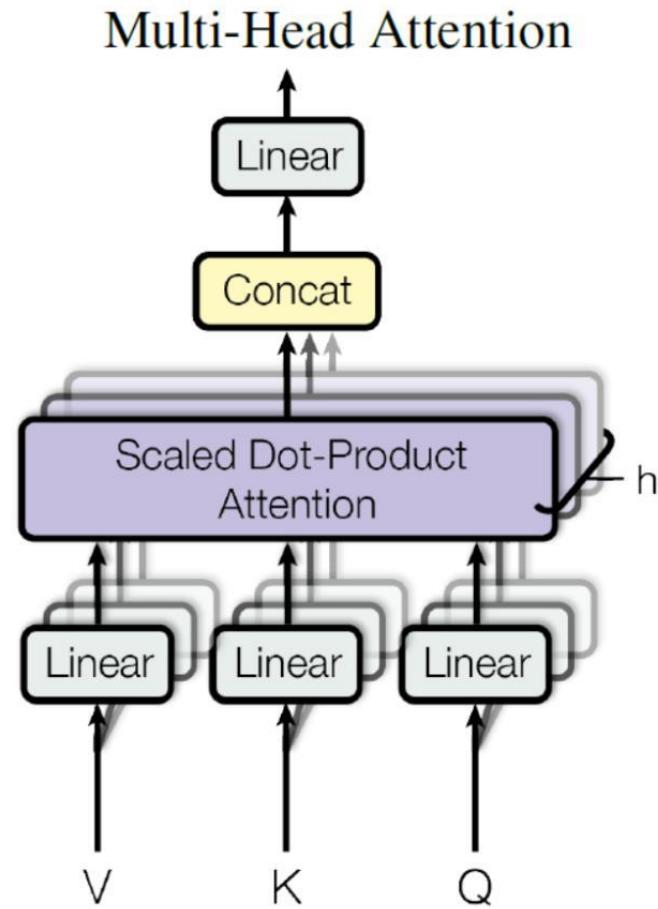
[A new way to let AI chatbots converse all day without crashing, MIT news]

Overview of KV cache compression

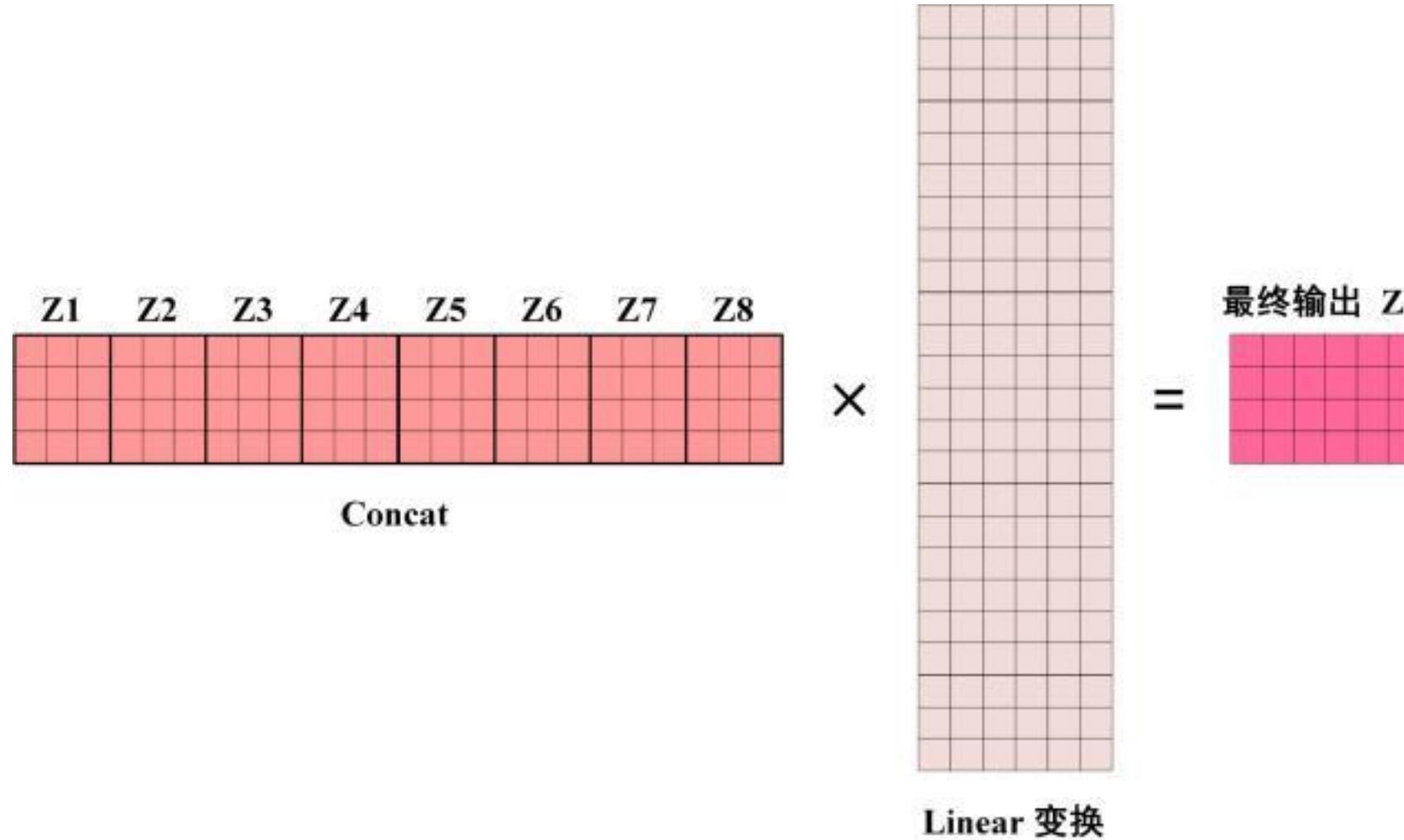


- Attention heads merge: MQA, GQA, MLA.
- Attention score matrix sparsity.
- Cross transformer layer sharing/merging.
- Precision reduction: Quantization.
- Hidden dimension reduction...

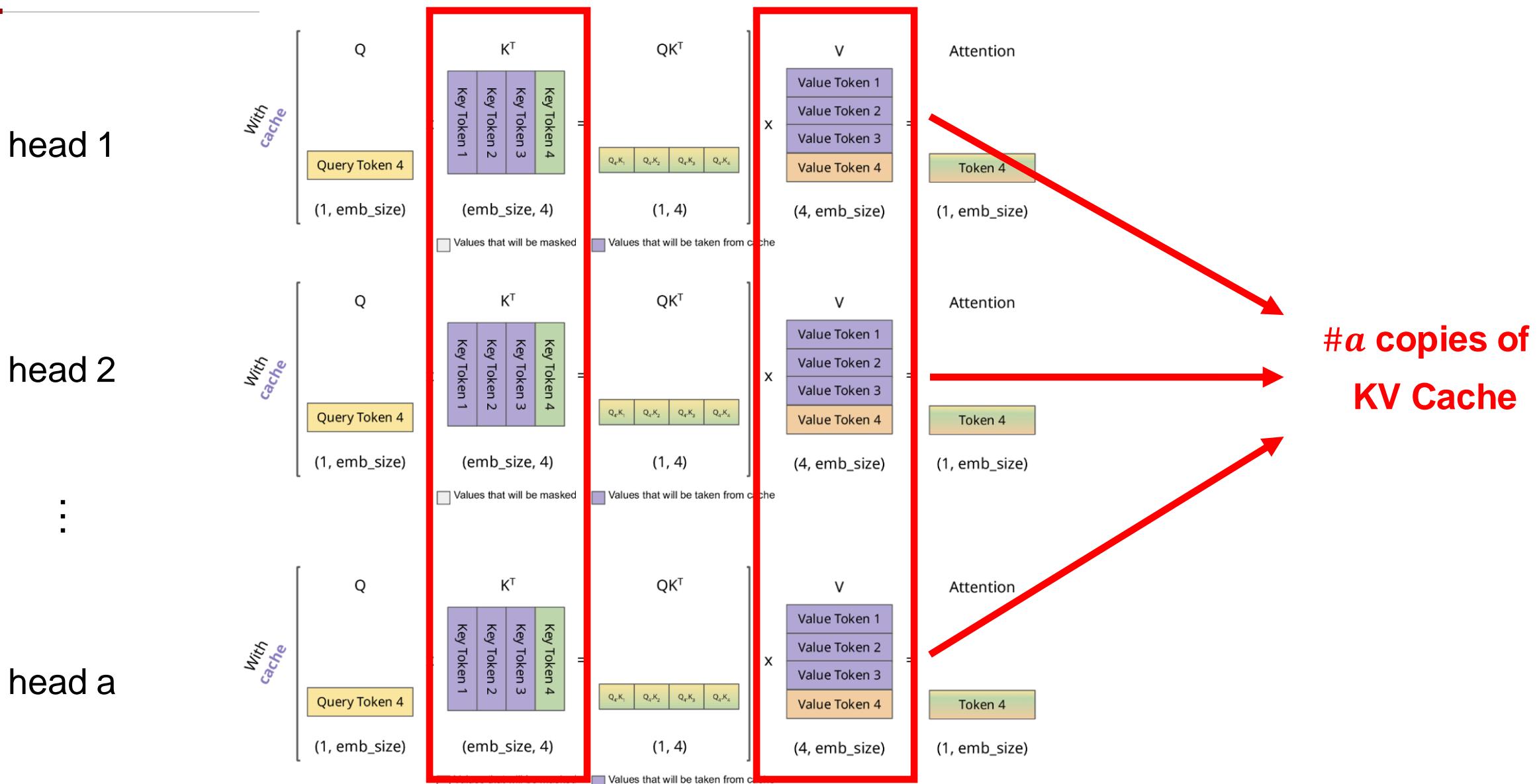
Recall the multi-head attention



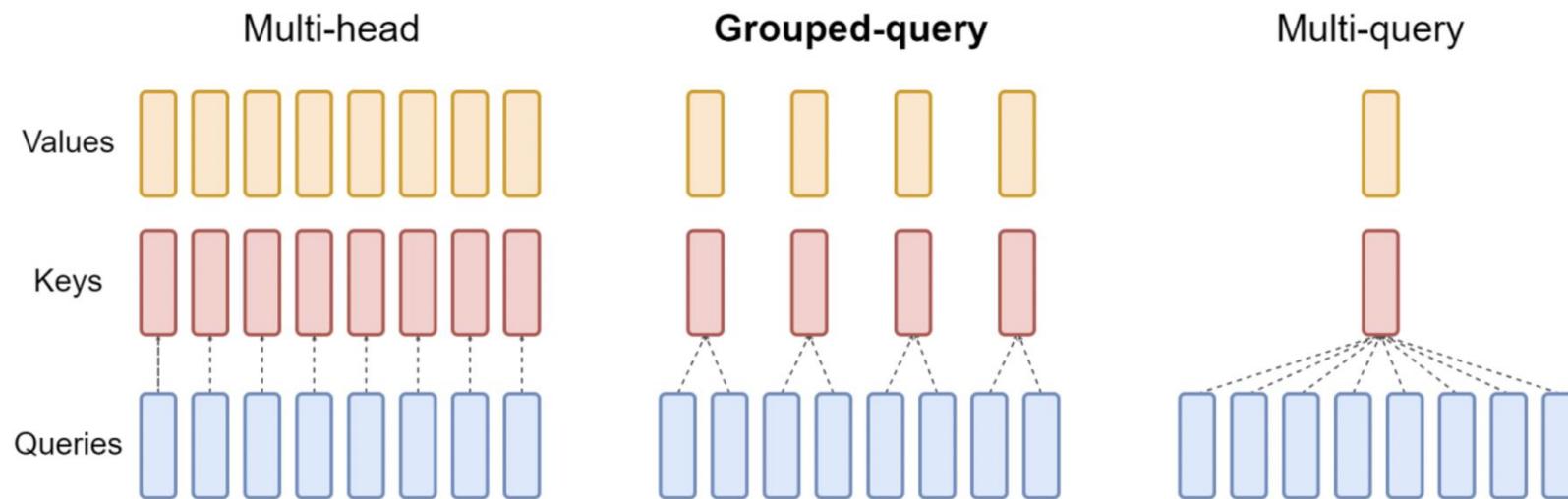
Multi-head attention: concatenation and linear transformation



Multi-head attention in KV Cache



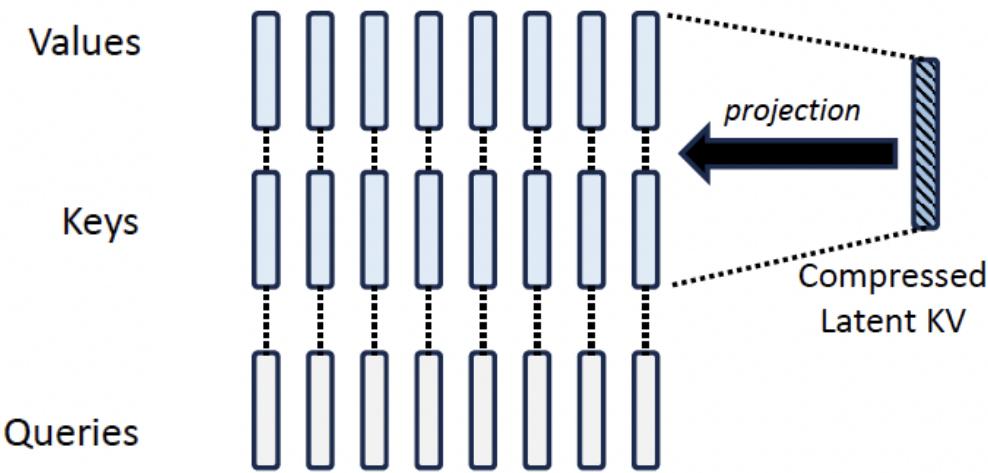
- Architecture optimization: reduce memory usage with **sharing KV between heads.**



- Fine-tuning needed: With MQA/GQA, models **need to train** with a fraction of the original training volume.
- Widely used: Falcon, PaLM: MQA; Llama-2-70B and Llama-3-8B use GQA.

Multi-Head Latent Attention (MLA)

- Recent progress: Multi-Head Latent Attention (MLA) by DeepSeek-V2



Key insight: **key-value is low rank.**

$$\mathbf{c}_t^{KV} = W^{DKV} \mathbf{h}_t,$$

$$\mathbf{k}_t^C = W^{UK} \mathbf{c}_t^{KV},$$

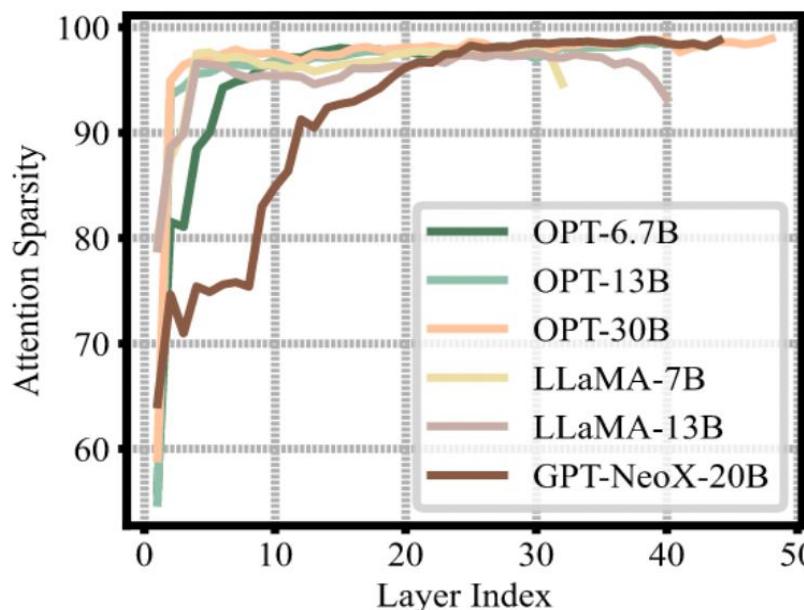
$$\mathbf{v}_t^C = W^{UV} \mathbf{c}_t^{KV},$$

- Low-Rank Key-Value Joint Compression

where $\mathbf{c}_t^{KV} \in \mathbb{R}^{d_c}$ is the compressed latent vector for keys and values; $d_c (\ll d_h n_h)$ denotes the KV compression dimension; $W^{DKV} \in \mathbb{R}^{d_c \times d}$ is the down-projection matrix; and $W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$ are the up-projection matrices for keys and values, respectively. During inference, MLA only needs to cache \mathbf{c}_t^{KV} , so its KV cache has only $d_c l$ elements, where l denotes the number of layers.

KV cache compression: Sparsity

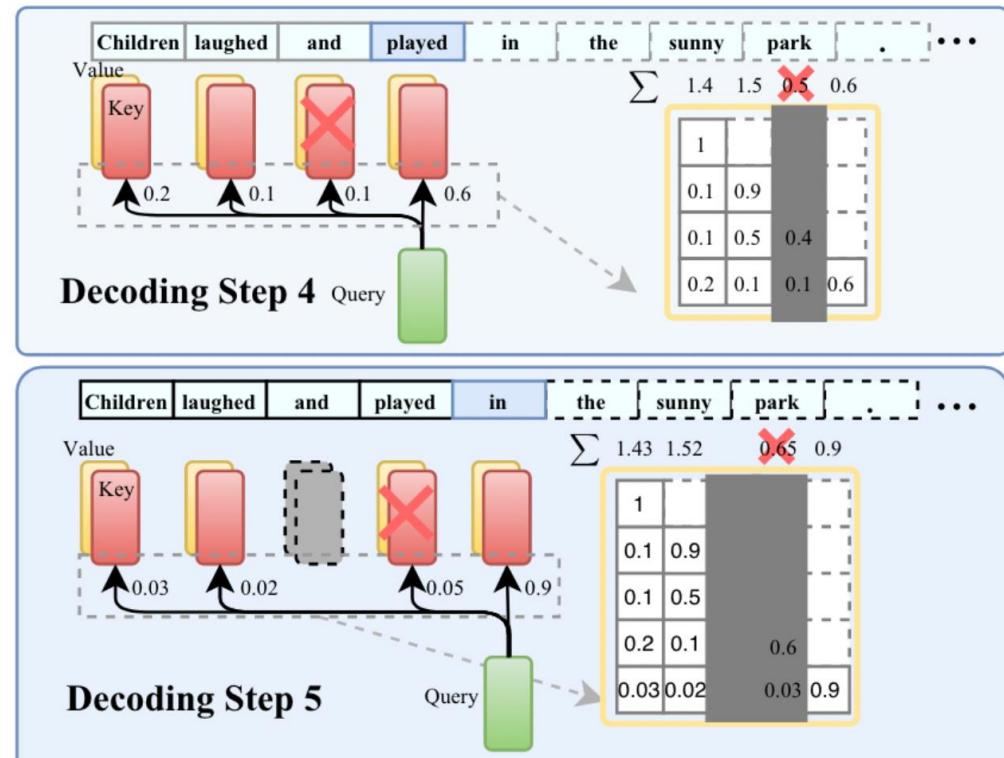
- Key Observation: Although LLM is densely trained, **Attention is naturally sparse.**
 - In OPT family, almost KV cache of all layers have a sparsity over **95%**, which means potential **20X** KV cache reduction.



- Idea:
Approximate the attention score matrix with sparse matrix, only load/save tokens with large score.
- Challenges:
 - How to **efficiently find** the “important” tokens?
 - How to **maintain the performance** of LLM without heavy finetuning?

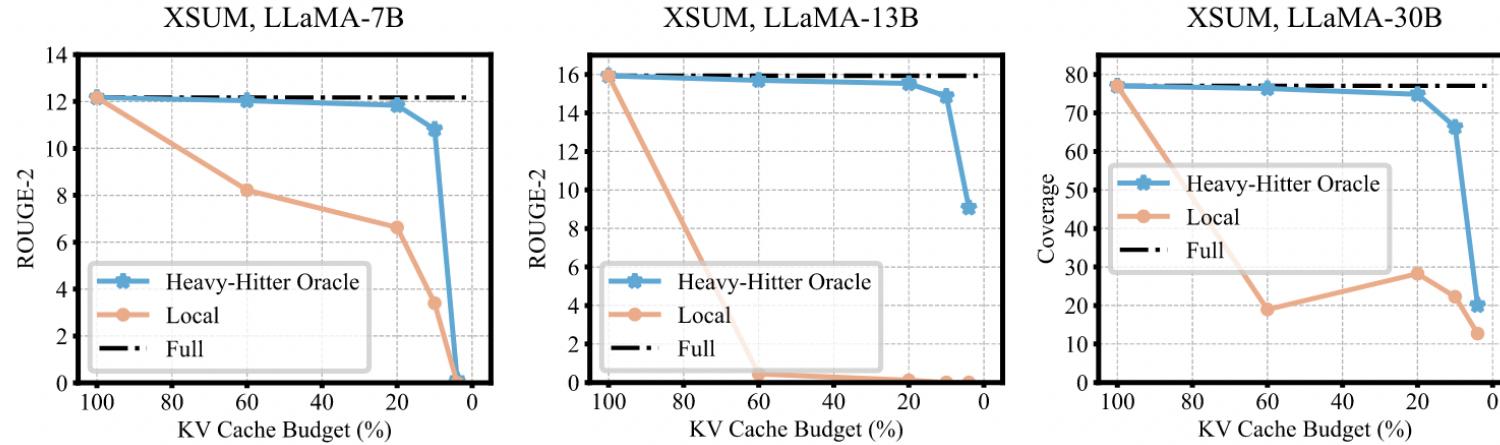
Heavy Hitter Oracle

- Motivation: A small set of tokens are critical during generation, called heavy-hitters.



- Heavy Hitter Oracle (H2O): a KV cache eviction policy that dynamically retains a balance of recent and H2 tokens.
- Local greedy algorithm
 - sum up the attention score of the previous tokens every decoding step.
 - Add local tokens.

Heavy Hitter Oracle



A100 GPU	FlexGen	H ₂ O
Throughput (token/s)	494	918 (1.9X)
Latency (s)	99	53 (1.9X)

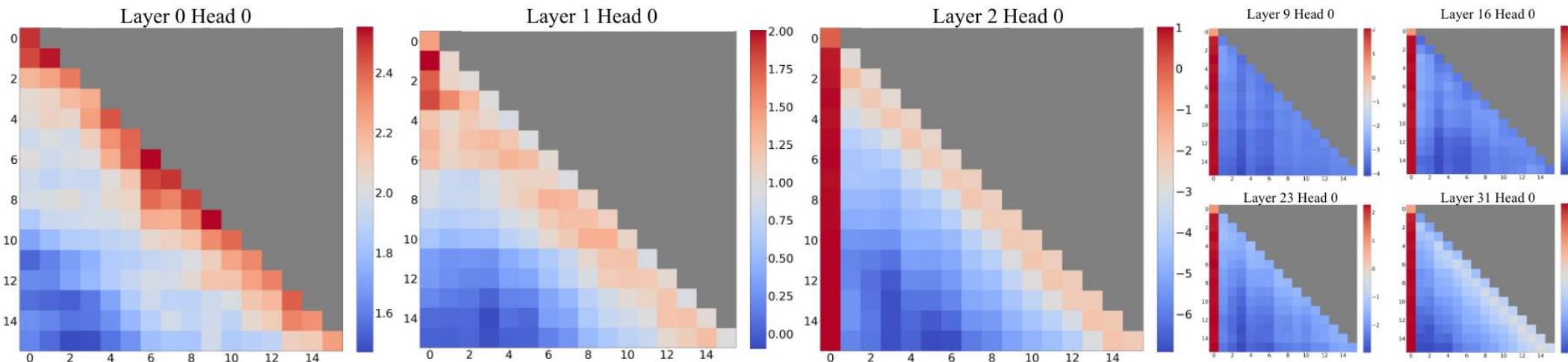
T4 GPU	Hugging Face	Deep Speed	FlexGen	H ₂ O
Throughput token/s	0.6	0.6	8.5	18.83 (3-29X)

3-29X throughput and **1.9X** latency, compatible with other method(quantization)

What are these heavy hitters?

- Interesting phenomenon: **Attention sink**

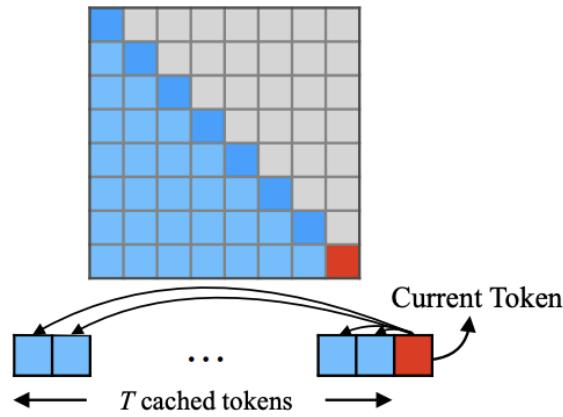
Large attention scores are given to initial tokens as a “sink” **even if they are not semantically important**, especially in the deeper layer.



average attention logits in Llama-2-7B over 256 sentences,

Streaming LLM

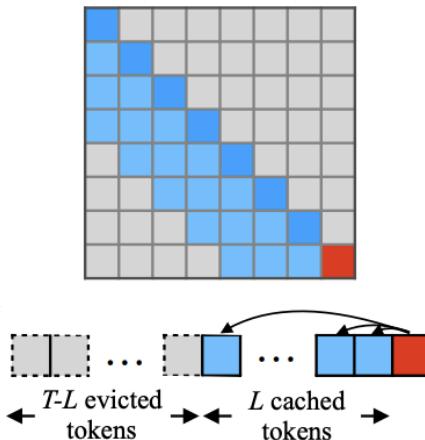
(a) Dense Attention



$O(T^2)\times$ PPL: 5641 \times

Has poor efficiency and performance on long text.

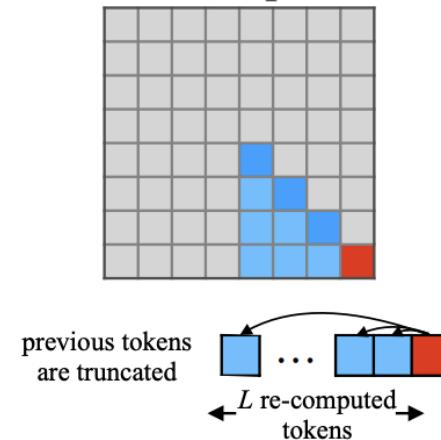
(b) Window Attention



$O(TL)\checkmark$ PPL: 5158 \times

Breaks when initial tokens are evicted.

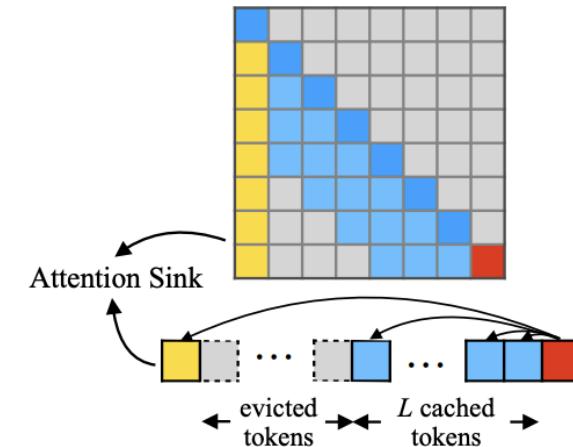
(c) Sliding Window w/ Re-computation



$O(TL^2)\times$ PPL: 5.43 \checkmark

Has to re-compute cache for each incoming token.

(d) StreamingLLM (ours)



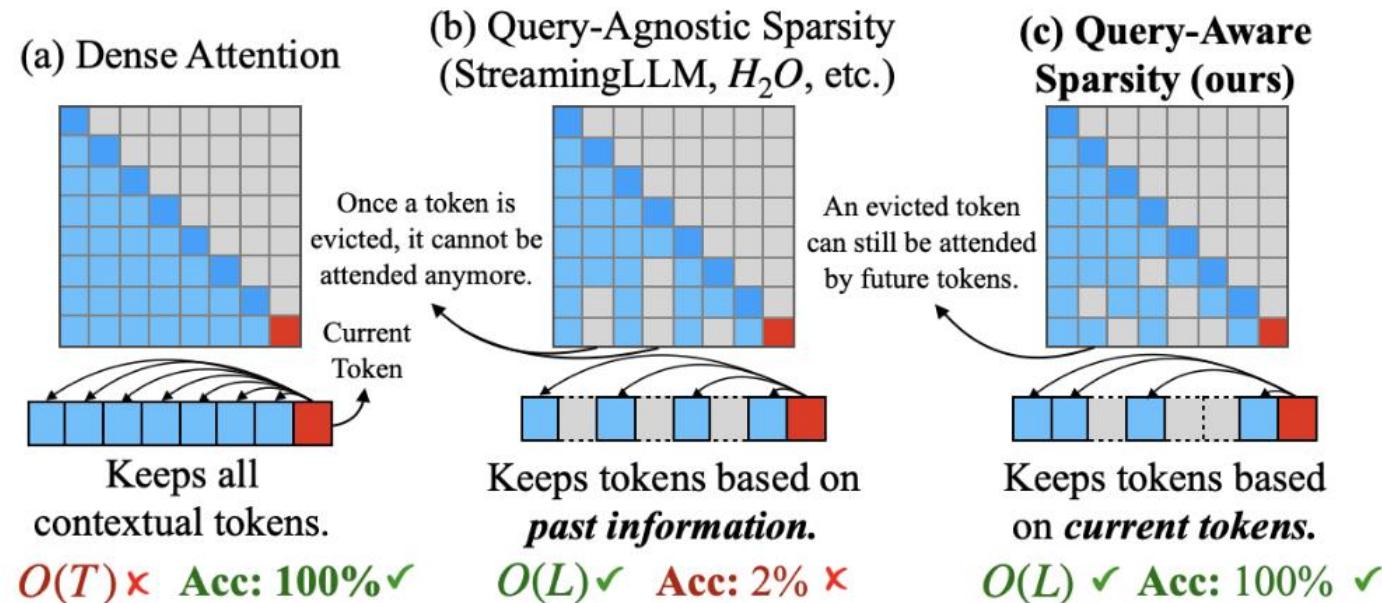
$O(TL)\checkmark$ PPL: 5.40 \checkmark

Can perform efficient and stable language modeling on long texts.

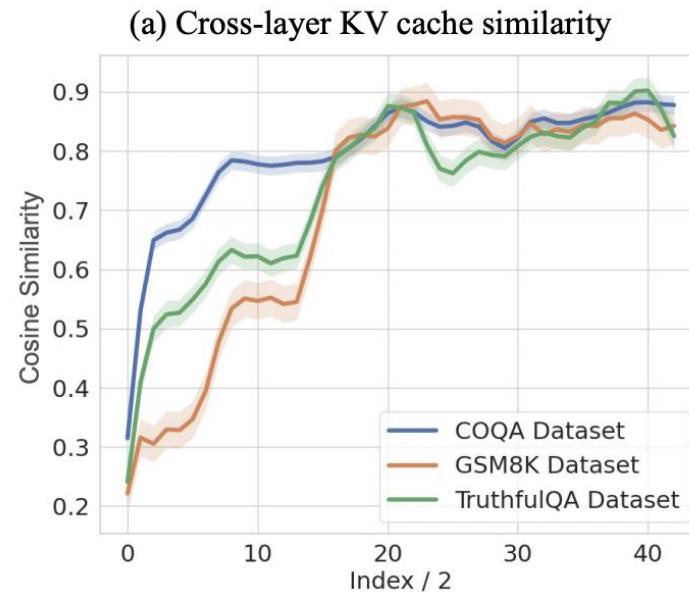
A video demo: <https://github.com/mit-han-lab/streaming-llm>

- Drawbacks of eviction based approaches: **Once evict the token, no longer can we attach to it.**
- Key idea: **Dynamic sparsity.**

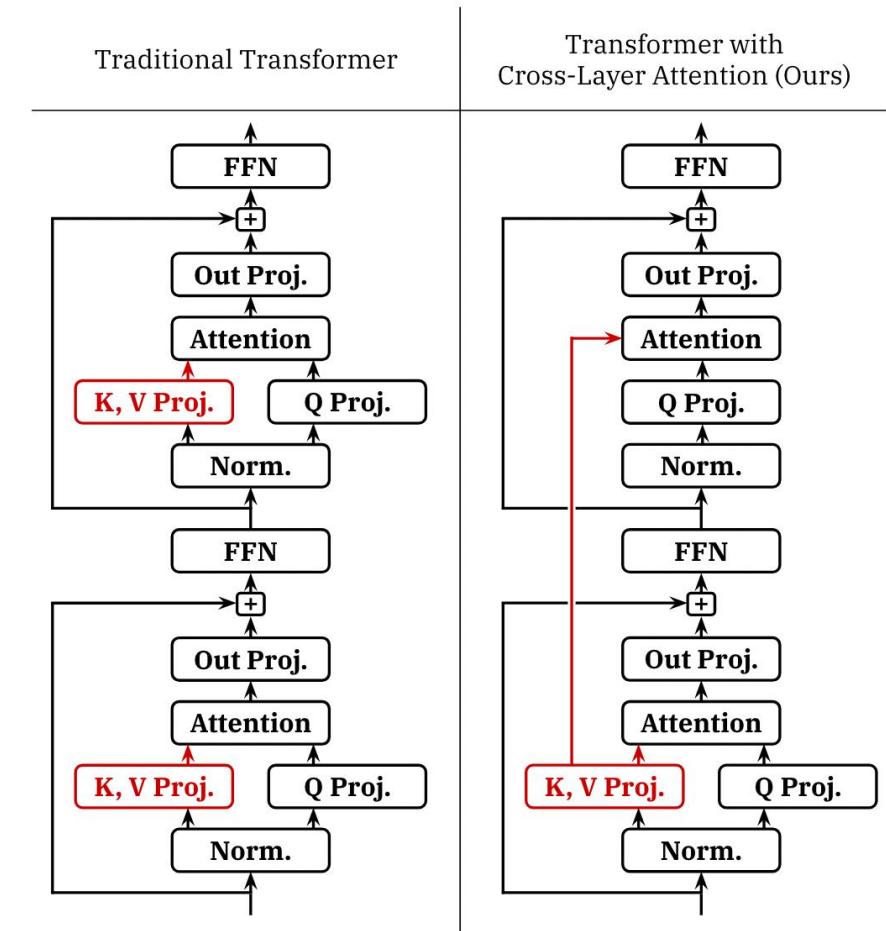
Keep all KV cache but **only load the critical KV cache** for attention computation.



KV cache compression along layer dimension



- Insight: **KV cache is similar between adjacent layers.**
- Cross-Layer Attention: **sharing KV activations across layers.**
 - Model with CLA need retraining.
- There are many recent work which aim to design smart algorithms to compress cache along layer dimension, such as interpolation.



- Quantization: widely used in model compression.

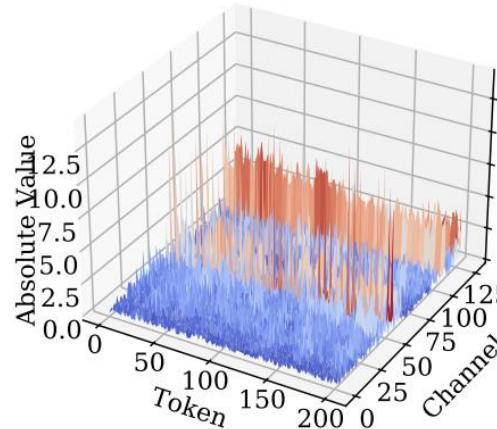
$$Q(\mathbf{w}) = \Delta \cdot \text{Round}\left(\frac{\mathbf{w}}{\Delta}\right), \quad \Delta = \frac{\max(|\mathbf{w}|)}{2^{N-1}},$$

- What's the challenge of KV cache quantization compared to weight quantization?
 - Impact of position encoding like RoPE.
 - KV cache keep updating along generation, which indicates we need to compute statistics on-the-fly (which is potentially expensive) or else we need to use offline calibration data (which potentially has negative accuracy implications).

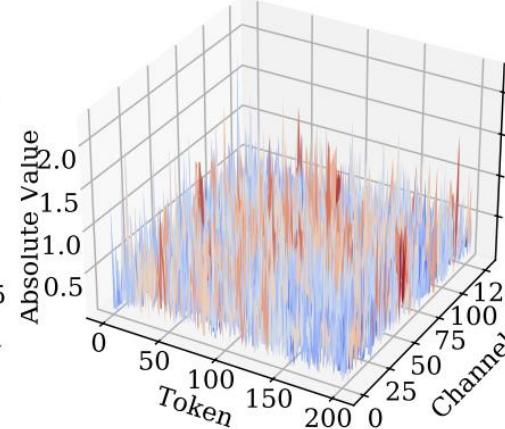
Quantization of KV cache

- Different data distribution of key and value

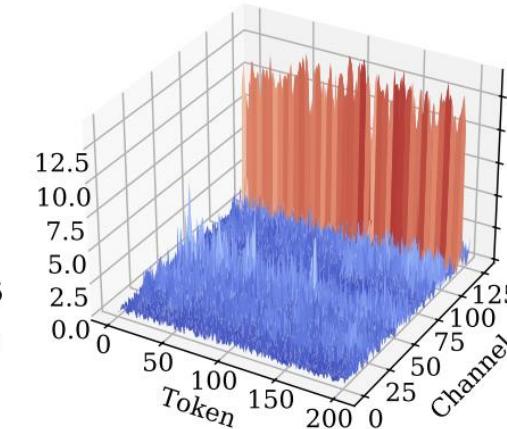
Llama-2-13B Layer 16
Head 0 Key Cache



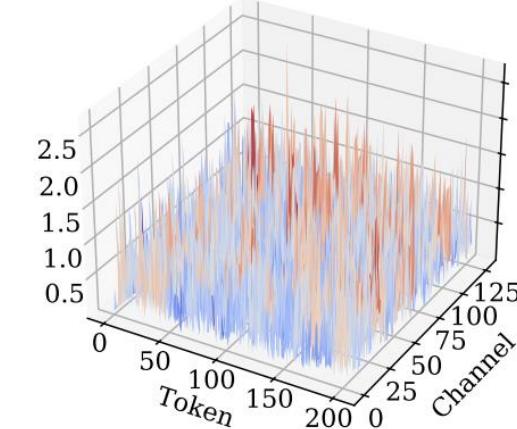
Llama-2-13B Layer 16
Head 0 Value Cache



Llama-2-13B Layer 31
Head 0 Key Cache



Llama-2-13B Layer 31
Head 0 Value Cache

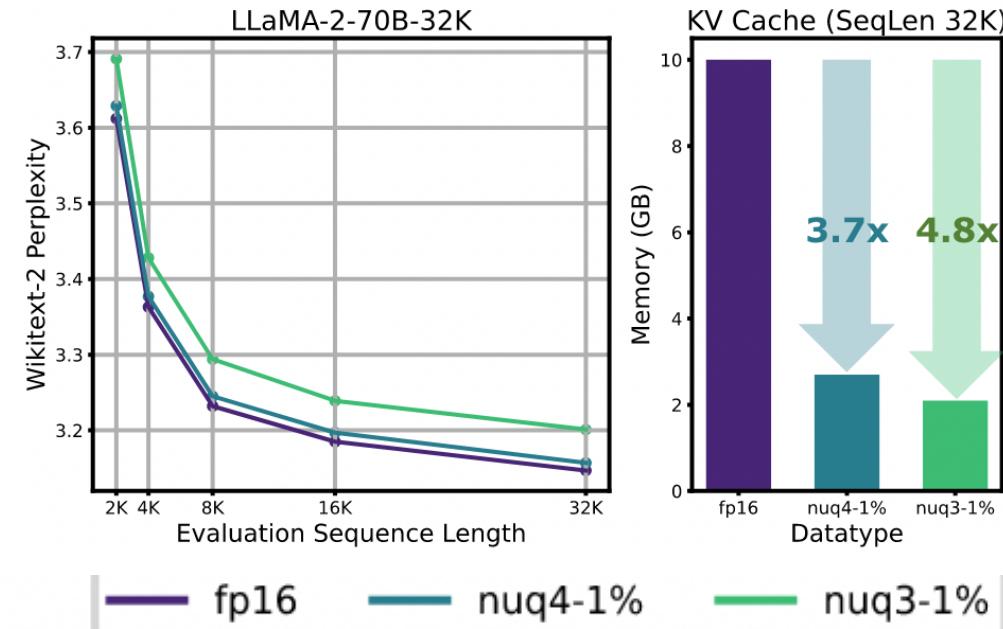
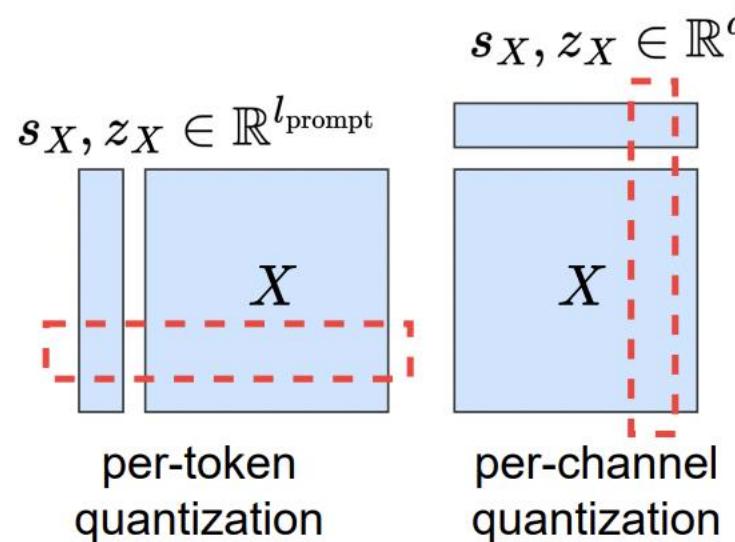


Key matrices tend to have distinct outlier channels, which have larger average magnitudes than other channels. For value cache, there is no obvious outlier pattern.

Quantization of KV cache

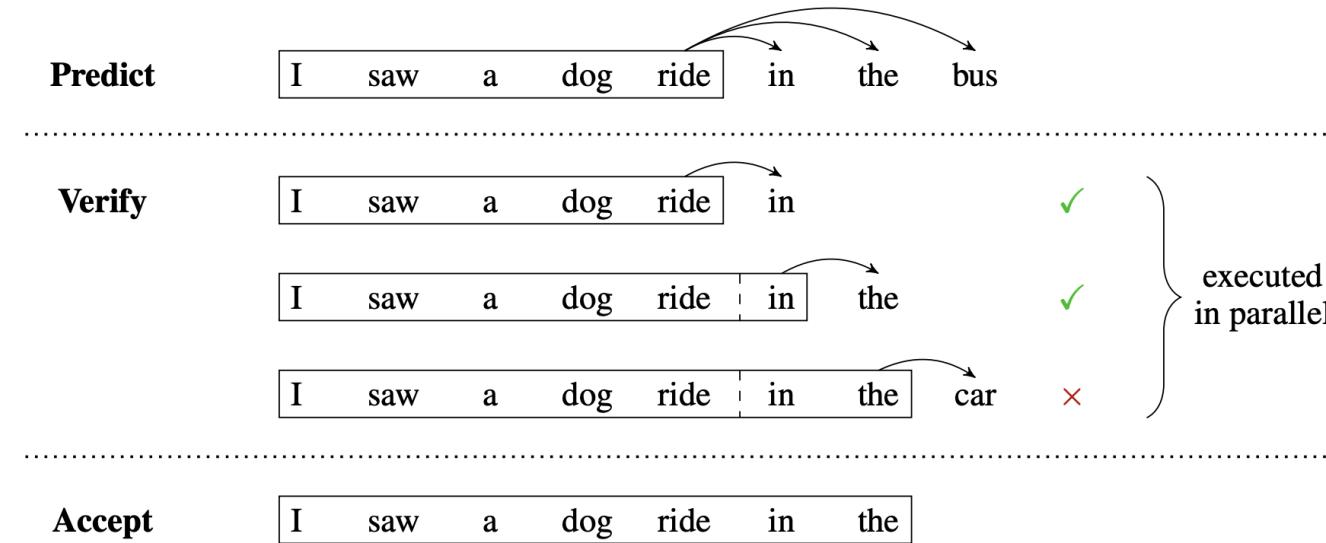
- Popular pattern:

For key, use per-channel quantization, while for value, use per-token quantization.



- Result: Work of KVQuant can serve LLaMA-7B with a context length of up to **1 million on a single A100-80GB GPU** and up to **10 million on an 8-GPU system**.

Speculative decoding



- Motivation: the vanilla generation process outputs only one token at a time.
- Intuition: some inference steps are “harder” but some are “easier”.
- Idea: **running two models in parallel.**
 - Target Model: the main LLM we want to use for our task.(Llama-70B)
 - Small Draft Model: a smaller, lightweight model to predict several potential completions.(Llama-7B)

Speculative decoding

```
[START] japan ' s benchmark bond n
[START] japan ' s benchmark nikkei 22 75
[START] japan ' s benchmark nikkei 225 index rose 22 76
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 7 points
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 0 1
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 9859
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 7 in
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in tokyo late
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]
```

- The green tokens are made by the draft model that the target model accepted.
- The red tokens are the rejected suggestions.
- The blue tokens are made by the target model to correct the prediction.

Speculative decoding

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1×
SpS (Nucleus)		0.114	7.52ms/Token	1.92×
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1×
SpS (Greedy)		0.156	7.00ms/Token	2.01×
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1×
SpS (Nucleus)		47.0%	5.73ms/Token	2.46×

TASK	M_q	TEMP	γ	α	SPEED
ENDE	T5-SMALL ★	0	7	0.75	3.4X
ENDE	T5-BASE	0	7	0.8	2.8X
ENDE	T5-LARGE	0	7	0.82	1.7X
ENDE	T5-SMALL ★	1	7	0.62	2.6X
ENDE	T5-BASE	1	5	0.68	2.4X
ENDE	T5-LARGE	1	3	0.71	1.4X
CNNNDM	T5-SMALL ★	0	5	0.65	3.1X
CNNNDM	T5-BASE	0	5	0.73	3.0X
CNNNDM	T5-LARGE	0	3	0.74	2.2X
CNNNDM	T5-SMALL ★	1	5	0.53	2.3X
CNNNDM	T5-BASE	1	3	0.55	2.2X
CNNNDM	T5-LARGE	1	3	0.56	1.7X

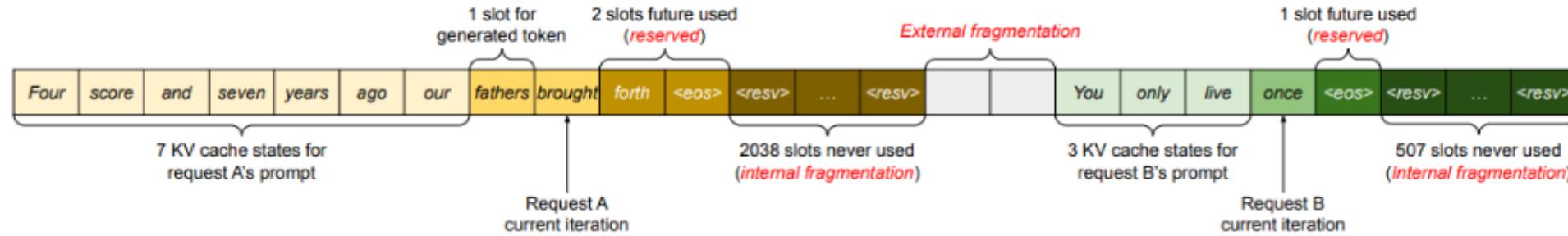


- Recommends $K = 3 \sim 4$, finds **2-2.5X** speed up



- Recommends $K = 3 \sim 7$, finds **2-3.4X** speed up

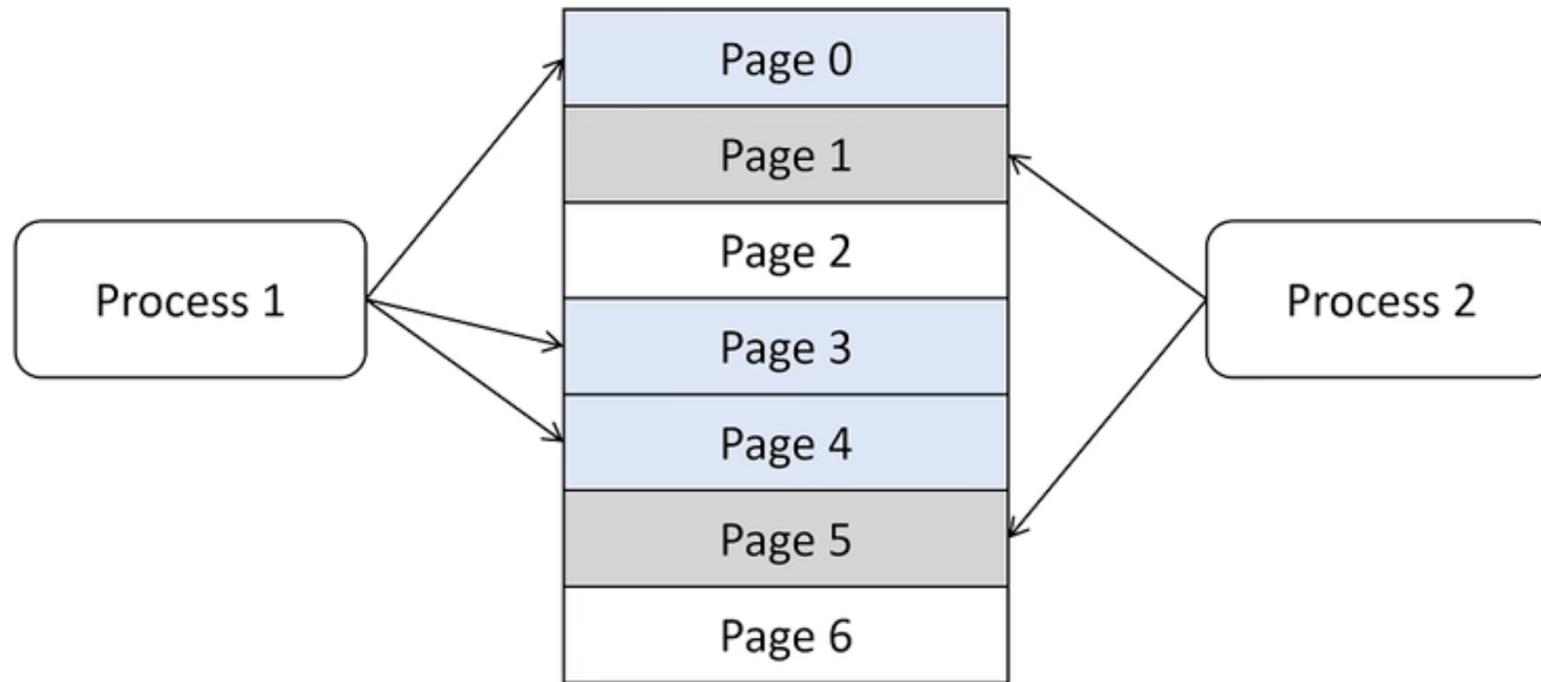
vLLM: Efficient management of KV cache



An illustration of memory wastage and fragmentation due to over-provisioning and inefficient KV cache management.

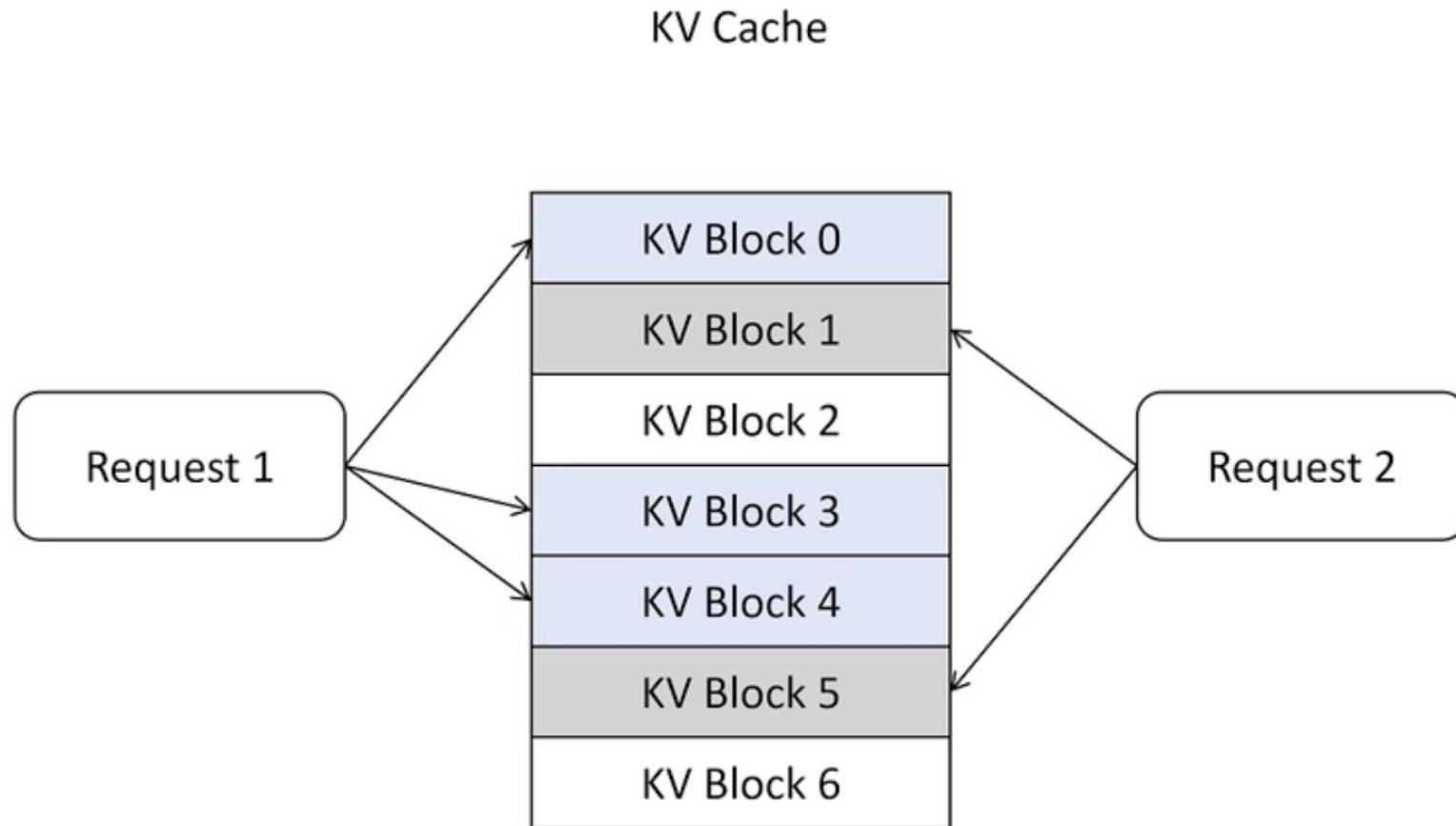
- Motivation: Although reducing KV cache size itself is effective, but there still be **inefficiencies in how this KV cache is managed.**
- Memory waste:
 - Reservation: not used at the current step, but used in the future
 - Internal fragmentation: over-allocated due to the unknown output length.
 - External fragmentation: due to different sequence lengths.

借鉴操作系统中的虚拟内存和页管理技术



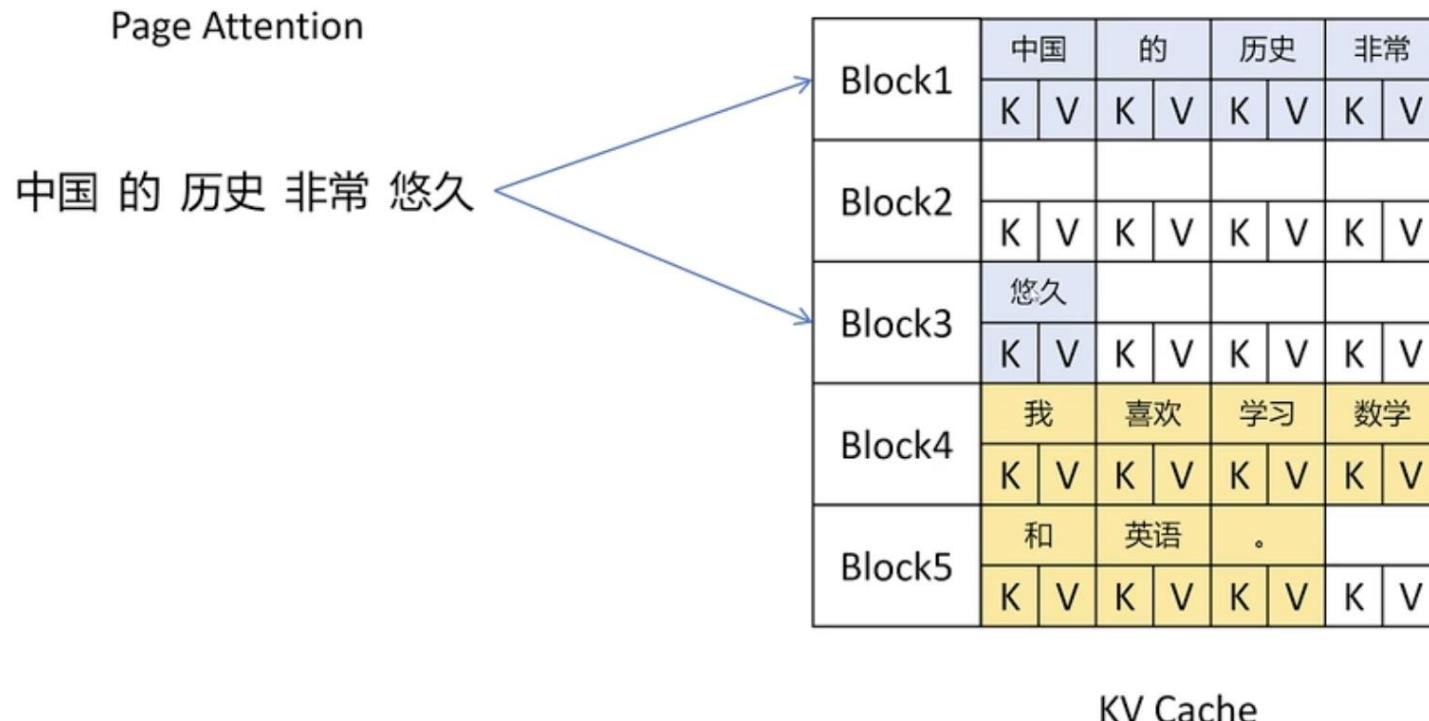
[B站Up主RethinkFun:[怎么加快大模型推理？](#)]

vLLM: Efficient management of KV cache



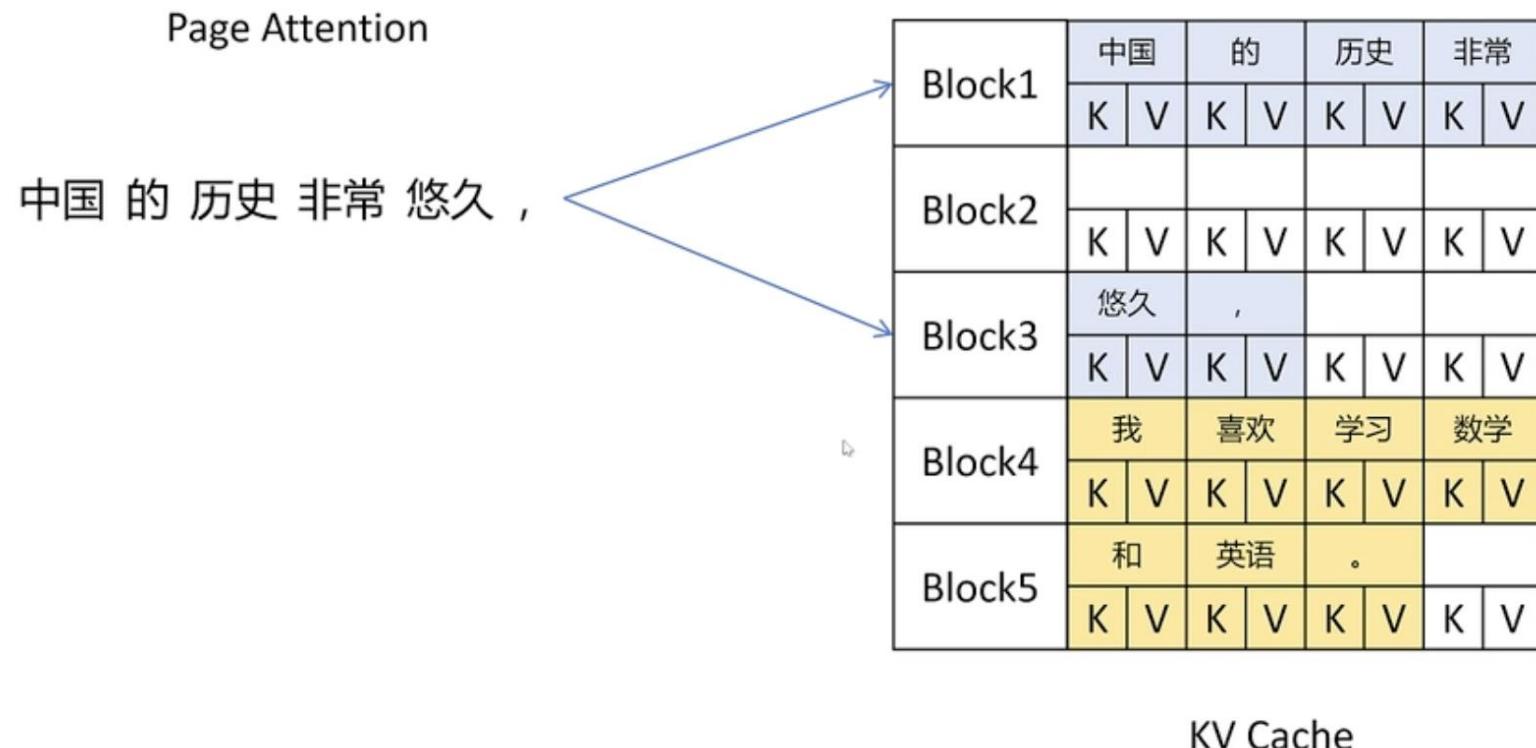
[B站Up主RethinkFun:[怎么加快大模型推理？](#)]

vLLM: Efficient management of KV cache



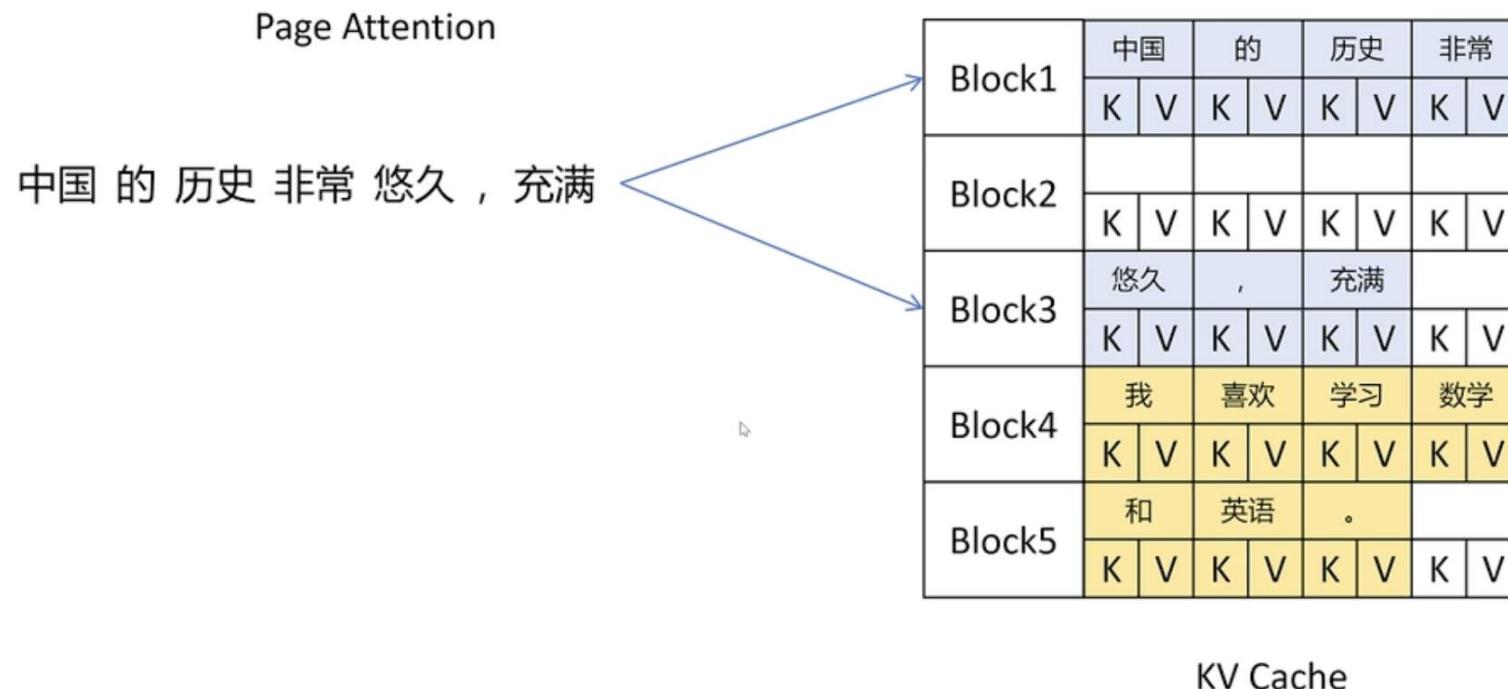
[B站Up主RethinkFun:[怎么加快大模型推理?](#)]

vLLM: Efficient management of KV cache



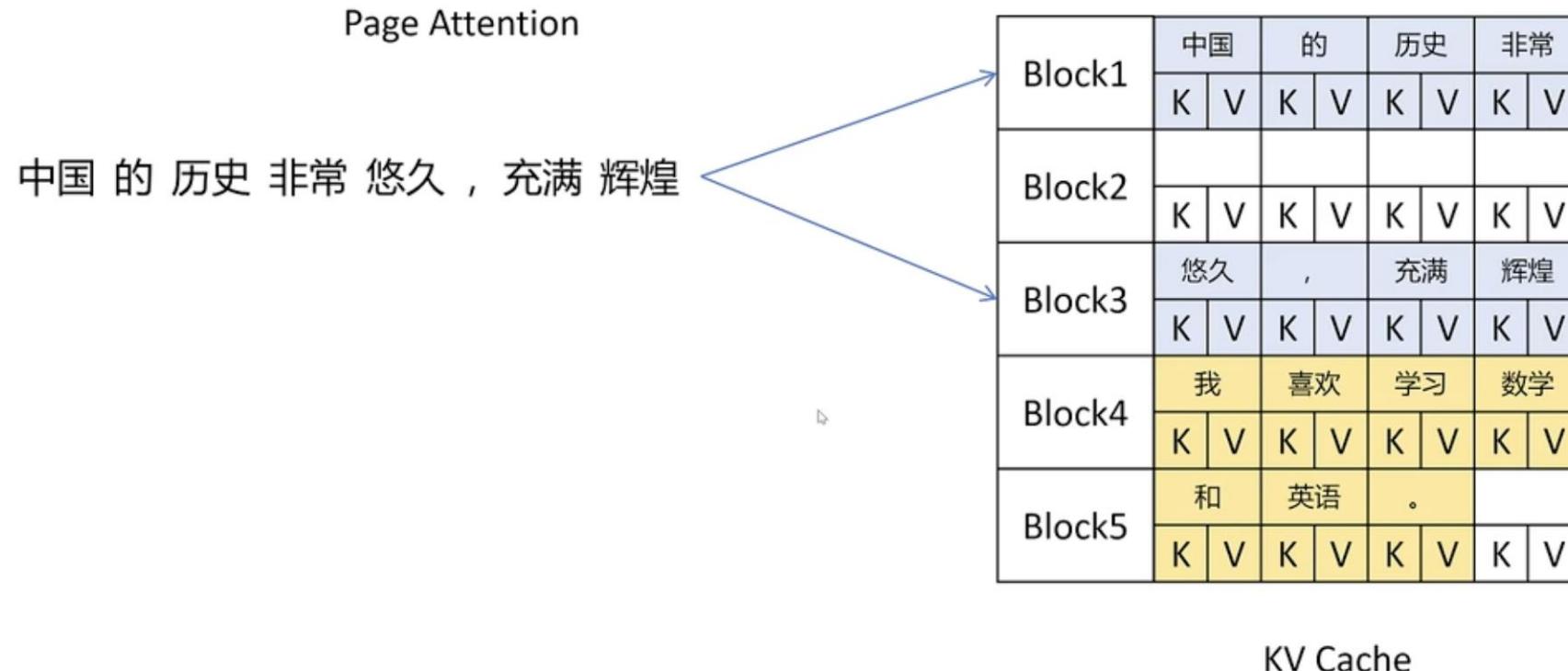
[B站Up主RethinkFun:[怎么加快大模型推理?](#)]

vLLM: Efficient management of KV cache



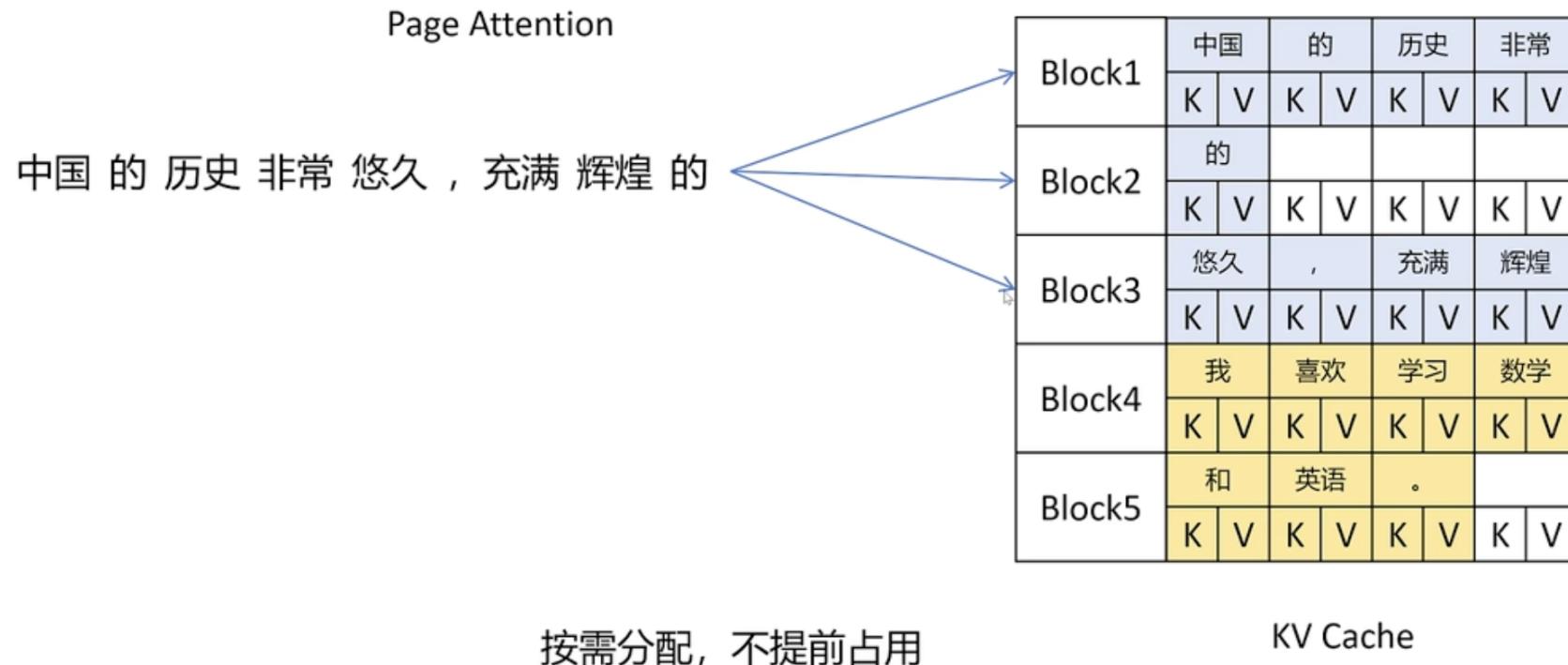
[B站Up主RethinkFun:[怎么加快大模型推理?](#)]

vLLM: Efficient management of KV cache



[B站Up主RethinkFun:[怎么加快大模型推理?](#)]

vLLM: Efficient management of KV cache



[B站Up主RethinkFun:[怎么加快大模型推理?](#)]

vLLM: Efficient management of KV cache

逻辑内存

Page Attention

中国 的 历史 非常 悠久

Block1	中国		的		历史		非常	
	K	V	K	V	K	V	K	V
Block2	悠久							
	K	V	K	V	K	V	K	V
Block3								
	K	V	K	V	K	V	K	V
Block4								
	K	V	K	V	K	V	K	V

逻辑KV Cache

物理Block	已填充
1	4
3	1
-	-
-	-

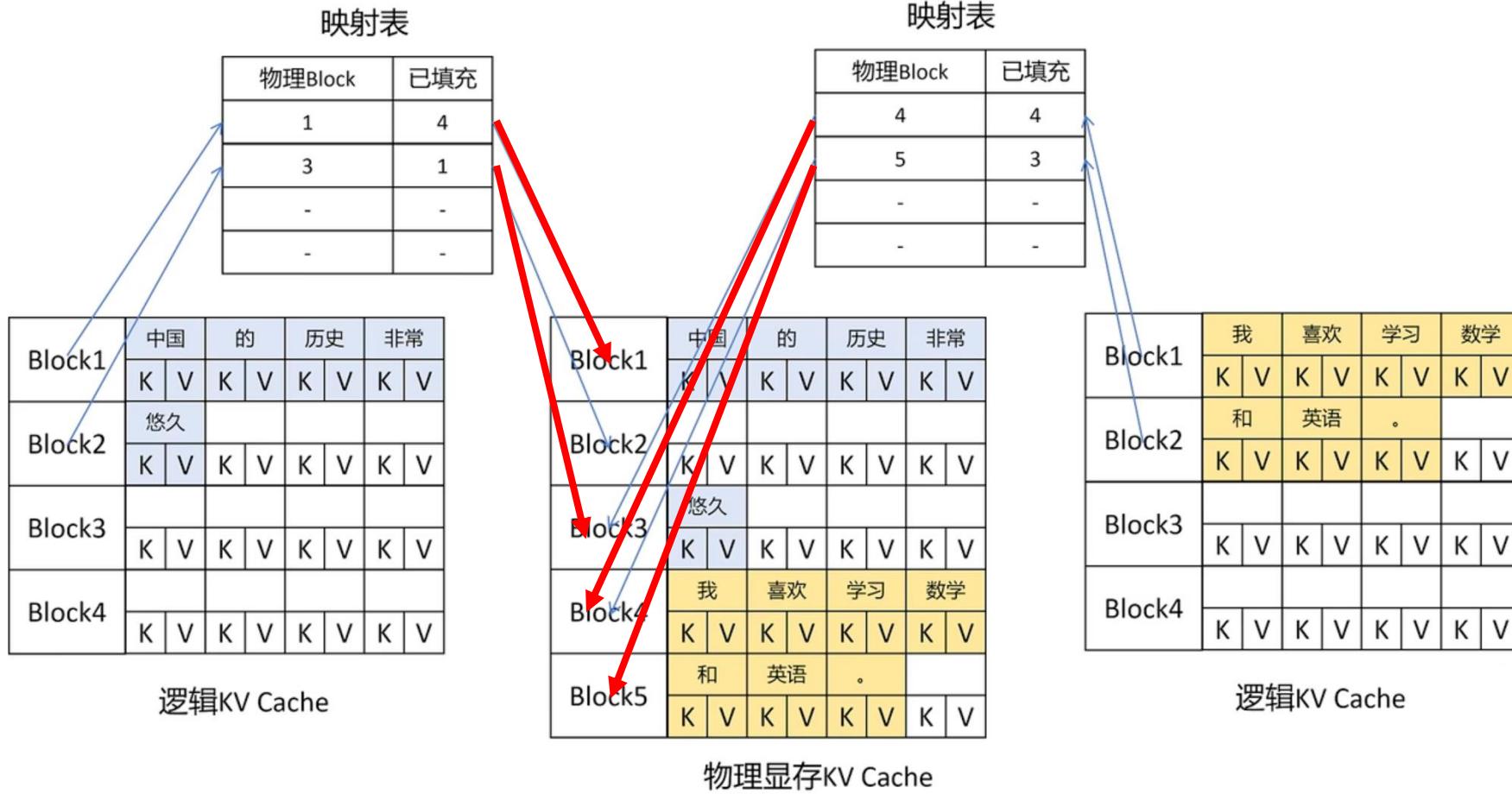
映射表

Block1	中国		的		历史		非常	
	K	V	K	V	K	V	K	V
Block2								
	K	V	K	V	K	V	K	V
Block3	悠久							
	K	V	K	V	K	V	K	V
Block4	我		喜欢		学习		数学	
	K	V	K	V	K	V	K	V
Block5	和		英语	.				
	K	V	K	V	K	V	K	V

物理显存KV Cache

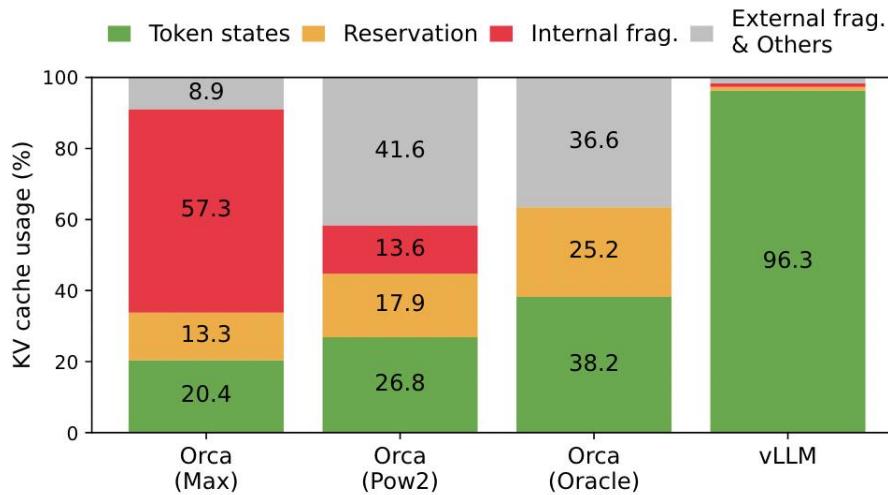
[B站Up主RethinkFun:[怎么加快大模型推理?](#)]

vLLM: Efficient management of KV cache

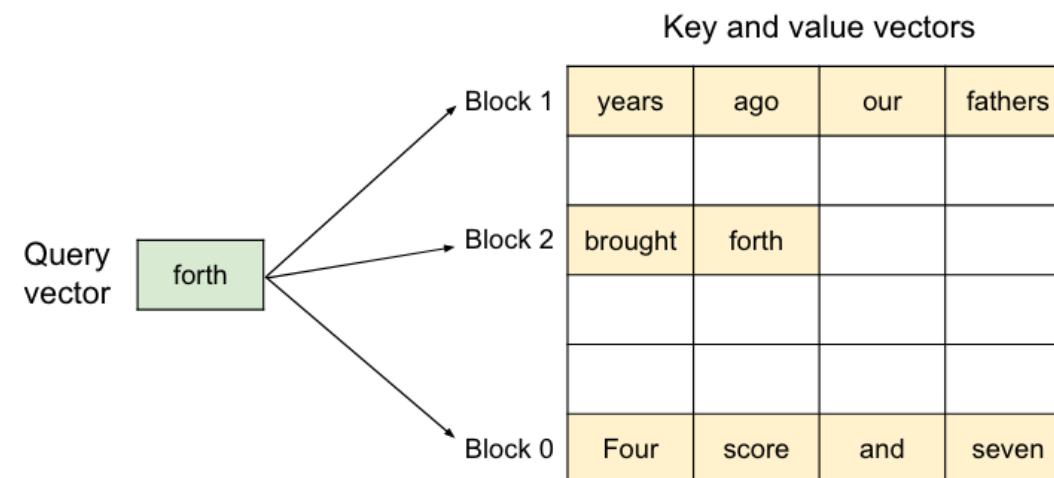
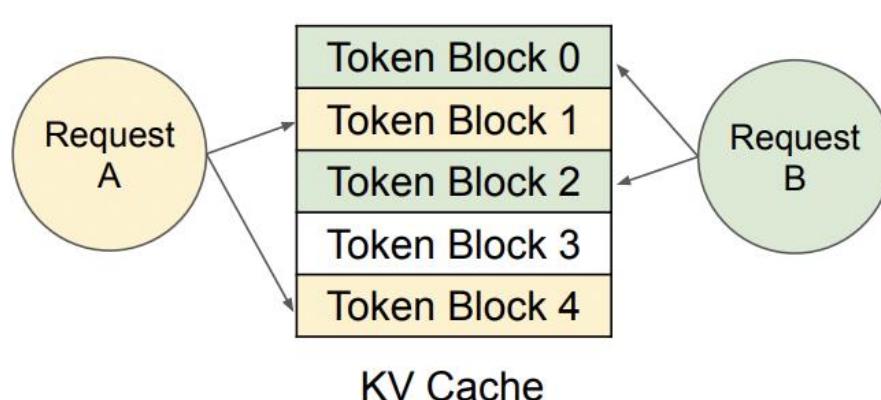


[B站Up主RethinkFun:[怎么加快大模型推理？](#)]

vLLM: Efficient management of KV cache

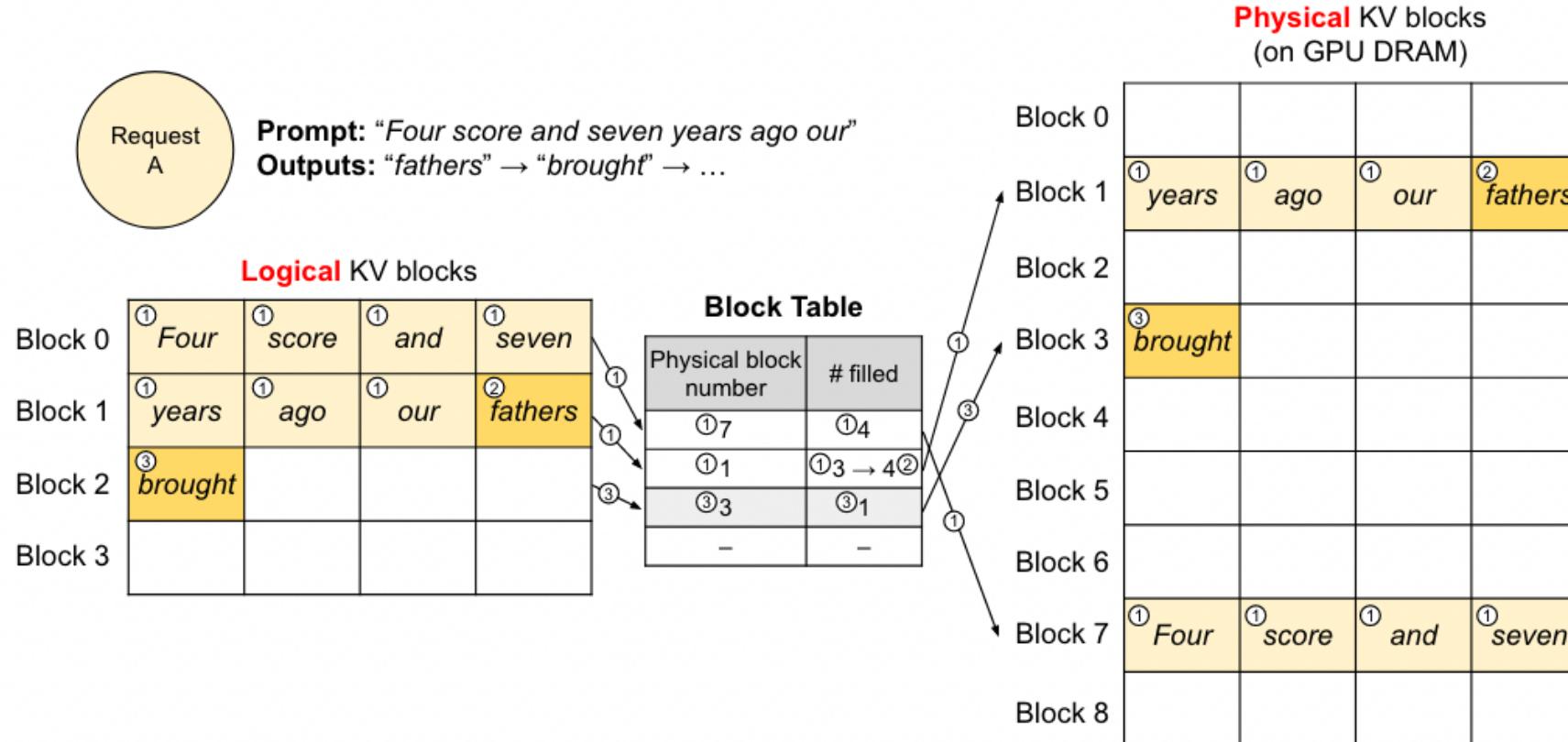


- Key idea: Inspired by virtual memory and paging,
Storing continuous KV in noncontiguous space in memory.
- Token block: A fixed-size contiguous chunk of memory that can store token states.



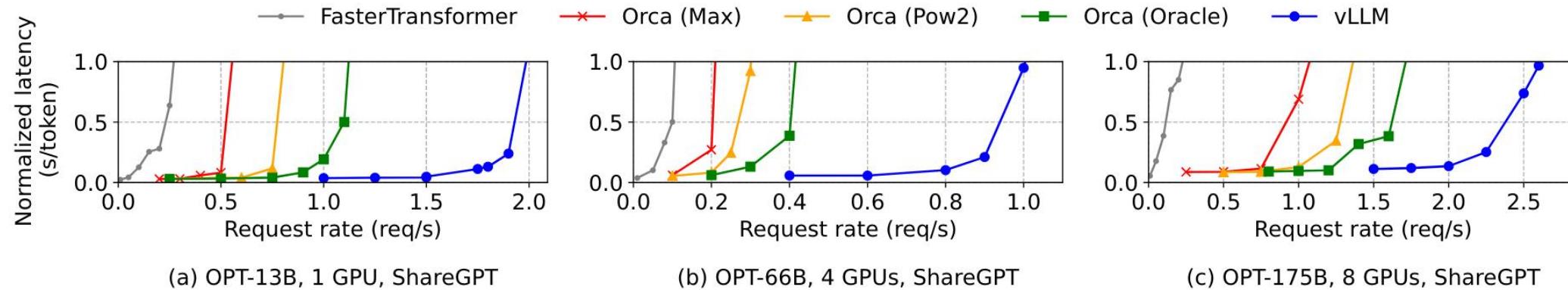
vLLM: Efficient management of KV cache

➤ Logical & physical token blocks



Physical blocks are fetched as required during attention computation using a block table that keeps account. As new tokens are generated, new block allocations are made.

vLLM: Efficient management of KV cache



➤ Conclusion

- vLLM improves the throughput of popular LLMs by **2-4x** with the same level of latency compared to SOTA systems, such as FasterTransformer and Orca.
- vLLM is more effective for large models, long sequences, and complex sampling methods.



Thank you!