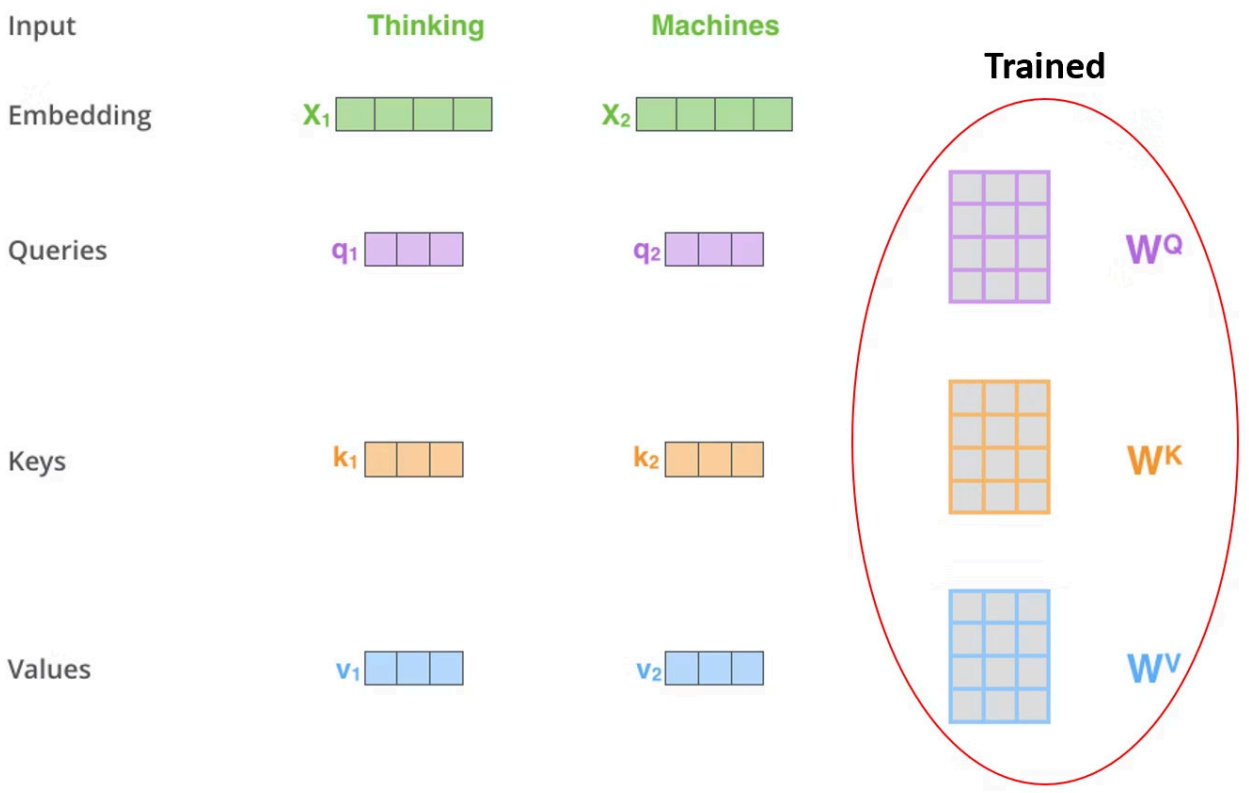
TransMLA: Multi-head latent attention is all you need

2025.09.05

Muhan Zhang@PKU

Attention Mechanism Basics



In each layer, every input token x is mapped to query, key, and value.

To update x's embedding, calculate inner products between query q and **all previous tokens**' key k (the causal mask).

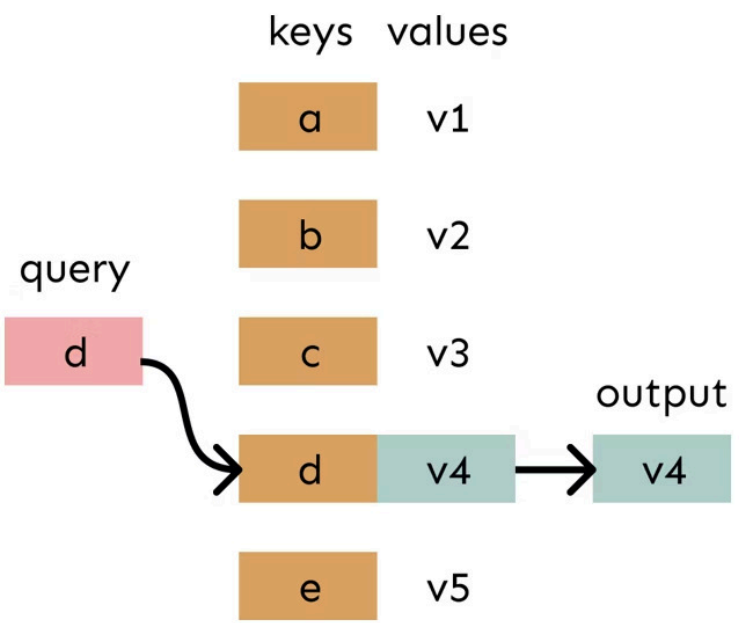
Use softmax to normalize the inner products, then take weighted sum of the values.

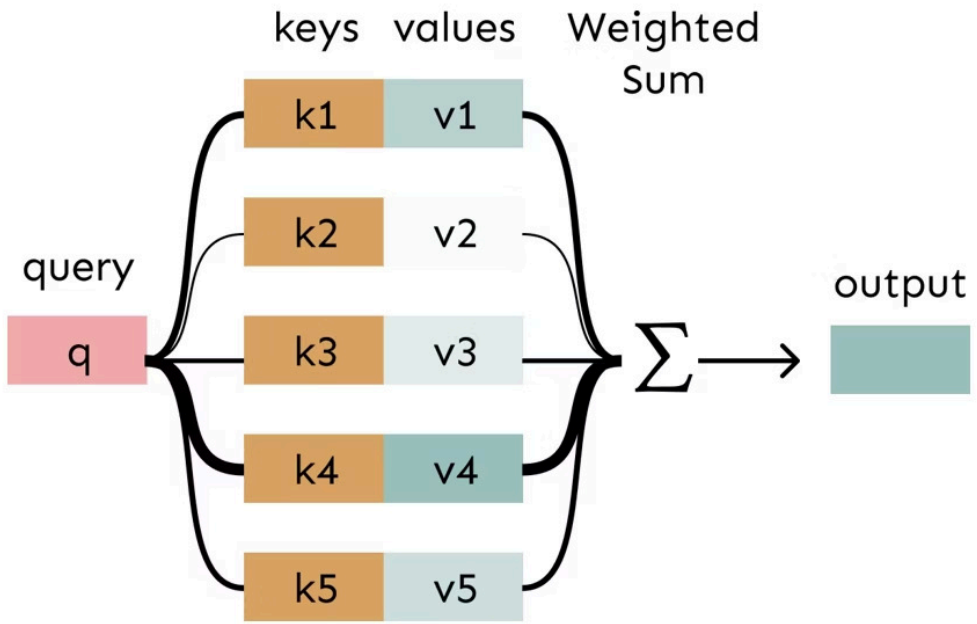
Finally map the weighted sum to the original dimension using an **output matrix**.

$$q_i = x_i W^Q \text{ (queries)} \qquad k_i = x_i W^K \text{ (keys)} \qquad v_i = x_i W^V \text{ (values)}$$

$$e_{ij} = q_i^T k_j \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_j \exp(e_{ij})} \qquad o_i = \left(\sum_j \alpha_{ij} v_j\right) W^O$$

Attention Mechanism Basics



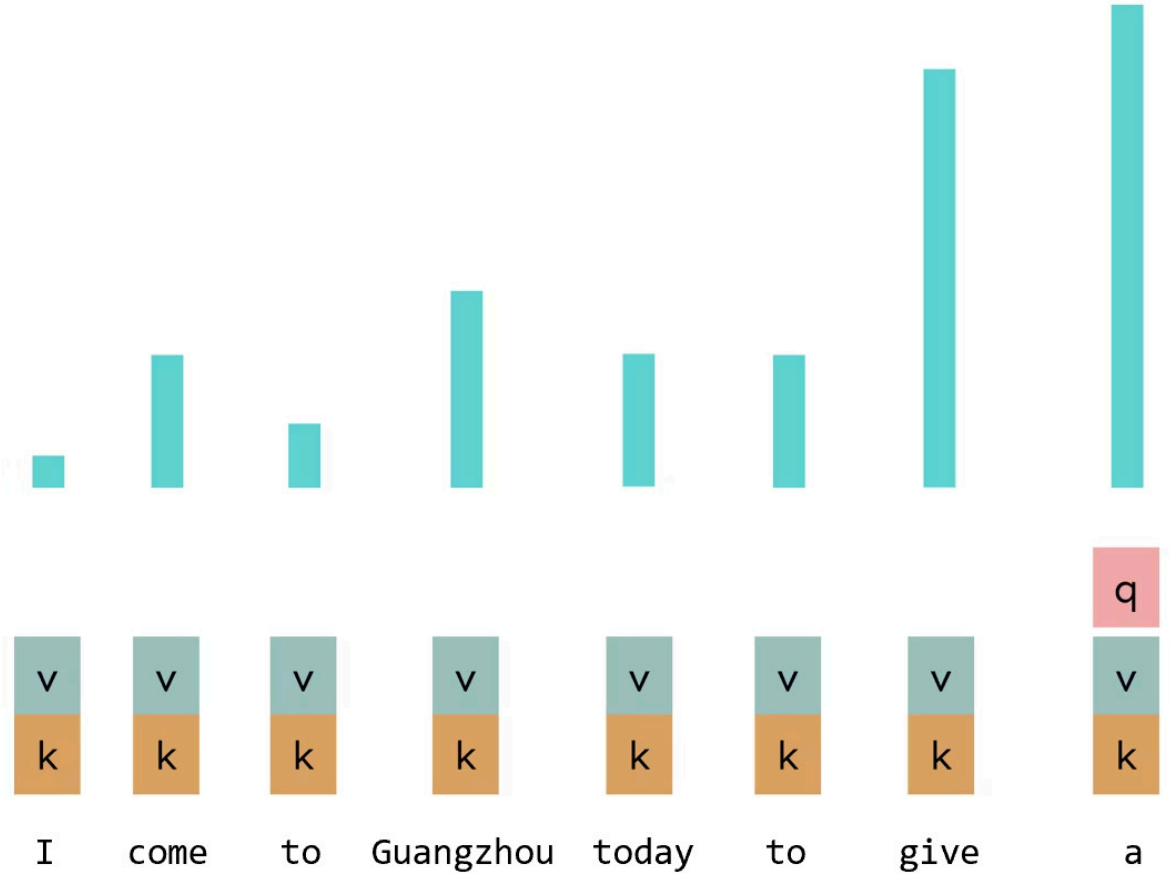


Attention is like a **fuzzy lookup table**. The query looks for the most relevant keys, and retrieves the values behind the keys.

Take weighted sum; the weight depends on the relevance score (inner product) between query and keys.

Use **multiple attention heads** (each maps x to a different set of q, k, v and performs attention within this space) to increase expressive power.

Attention Mechanism Basics



During generation, **one new token is decoded every time**.

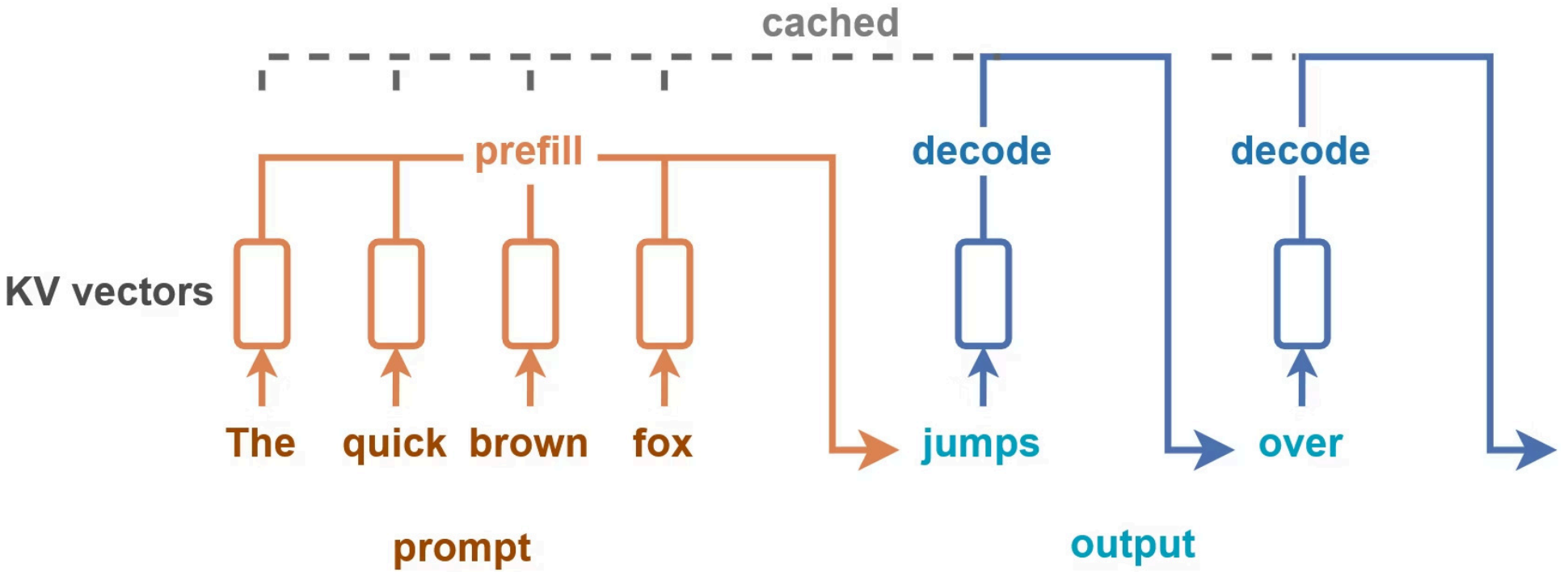
Don't need to recompute previous tokens' values and keys.

Due to causal mask, future tokens don't affect old tokens' embeddings at all.

Therefore, just **cache all generated tokens' value and key embeddings** at every layer into GPU memory. Trade storage for computation.

KV-cache significantly accelerates modern causal Transformer's decoding speed!

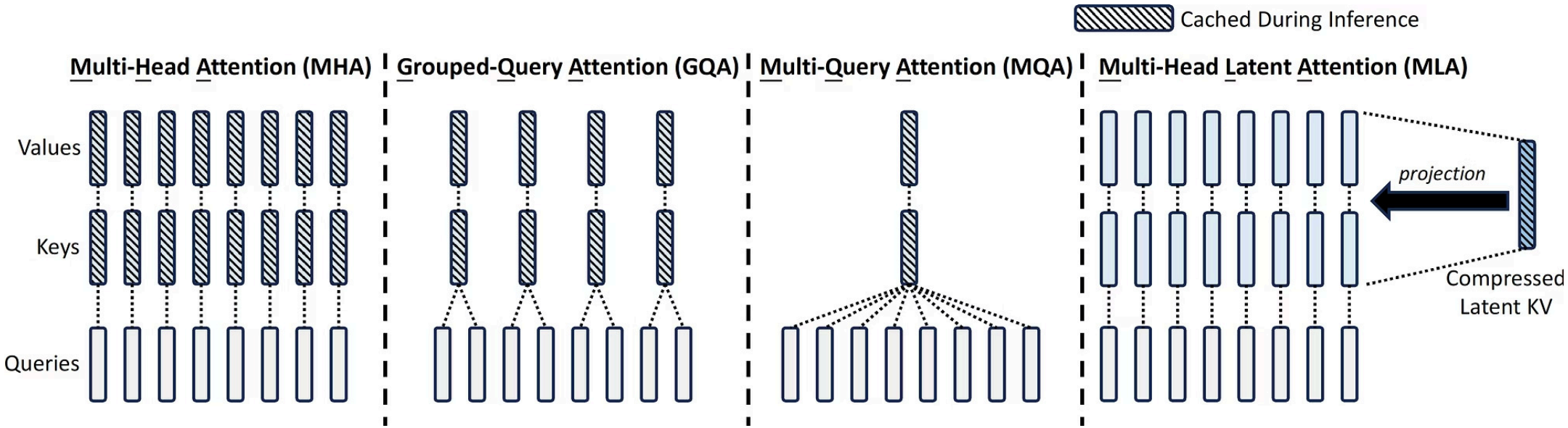
Prefill & Decode



prefill: parallel processing of prompt tokens, *decode:* sequential processing of single output tokens.

Think of a question: **Why only key and value are cached, while query isn't?**

Mainstream Attention Types



Standard **Multi-Head Attention (MHA)** uses different key and value projections in all different heads—**too much KV cache size**!

To save KV cache, **Multi-Query Attention (MQA)** uses the same key and value projections across all heads—**loses too much power**!

Grouped-Query Attention (GQA) shares the same key and value **within a group of heads**—a better balance!

Multi-Head Latent Attention (MLA) from DeepSeek stores a **shared larger latent KV cache**, and **projects it to different heads when used**. It increases some computation, but **largely reduces the KV cache size**!

Practical Challenges and Optimization

Key Performance Metrics

When working with LLMs, four critical metrics will shape your implementation decisions:

1. **Time to First Token (TTFT)**: How quickly can you get the first response? This is crucial for user experience and is primarily affected by the prefill phase.
2. **Throughput**: How many requests can you handle simultaneously? This affects scaling and cost efficiency.
3. **VRAM Usage**: How much GPU memory do you need? This often becomes the primary constraint in real-world applications.

The Context Length Challenge

One of the most significant challenges in LLM inference is managing context length effectively. Longer contexts provide more information but come with substantial costs:

- **Memory Usage**: Grows quadratically with context length
- **Processing time**: Increases linearly with longer contexts
- **Resource Allocation**: Requires careful balancing of VRAM usage

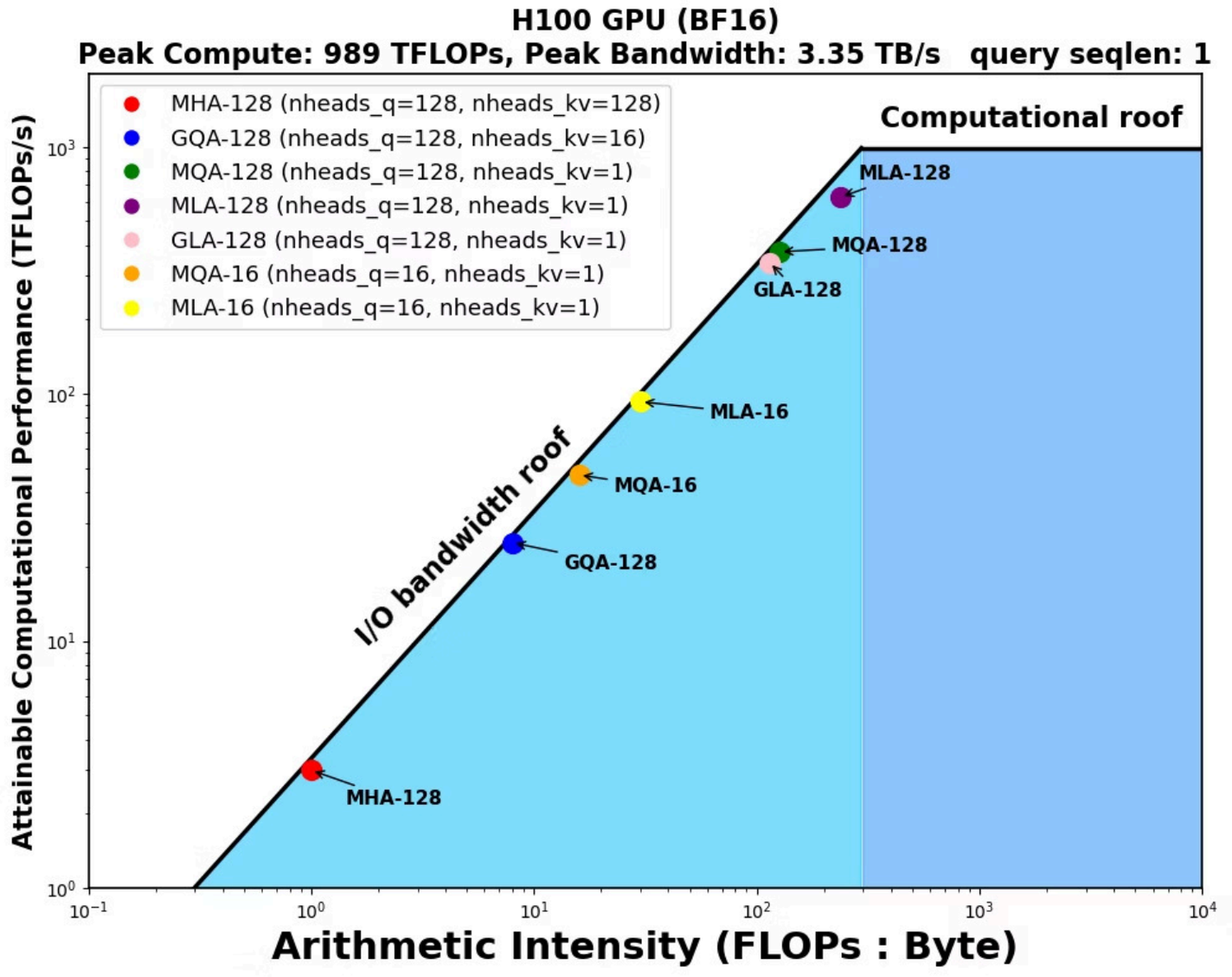
The KV Cache Optimization

KV (Key-Value) caching significantly improves inference speed by storing and reusing intermediate calculations.

- Reduces repeated calculations
- Improves generation speed
- Makes long-context generation practical

The trade-off is additional memory usage, but the performance benefits usually far outweigh this cost.

Roofline Model



MLA has the highest throughput. No matter how many actual heads are used, it only stores one shared latent KV cache.

GPU Information

GPU Name	GeForce RTX 4090	A100 SXM4 40 GB	HGX H20	H800 SXM
Architecture	NVIDIA Ada Lovelace	NVIDIA Ampere	NVIDIA Hopper	NVIDIA Hopper
Memory	24 GB GDDR6X	40 GB HBM2e	96 GB HBM3	80 GB HBM3
Bandwidth	1.0 TB/s	1.55 TB/s	4.0 TB/s	3.35 TB/s
BF16 Tensor Core	82.58 TFLOPS	312 TFLOPS	148 TFLOPS	1979 TFLOPS
TF32 Tensor Core	82.58 TFLOPS	156 TFLOPS	74 TFLOPS	989 TFLOPS
NVLink	—	600 GB/s	900 GB/s	**H100：** 900 GB/s **H800：** 400 GB/s
PCIe	PCIe 4.0: 64 GB/s	PCIe 4.0: 64 GB/s	PCIe 5.0: 128 GB/s	PCIe 5.0: 128 GB/s

Arithmetic Intensity in Decoding

An NVIDIA H800 SXM5 GPU has a peak memory bandwidth (HBM) of 3.35 TB/s and peak FLOPs of 990 TFlops. However, due to throttling (reducing to ~1600 MHz in our case), the practical peak FLOPs drops to ~865 TFlops. Therefore, when Arithmetic Intensity $> \frac{865}{3.35} = 256$, the kernel is compute-bound; otherwise, it's memory-bound. Let L be the KV sequence length, h_q the number of query heads, h_{kv} the number of KV heads. The KV multiplicity is $m_{kv} \in 1, 2$, with $m_{kv} = 1$ for shared KV states $(K = V)$ and $m_{kv} = 2$ for distinct KV states $(K \neq V)$.

$\text{Arithmetic Intensity} = \frac{\text{FLOPs}}{\text{Memory Access}} = \frac{2h_qLd}{2L_qh_qd+m_{kv}h_{kv}LD} \approx \frac{2h_q}{m_{kv}h_{kv}}$, where $q \in \mathbf{R}^{h_qd}$, $k, v \in \mathbf{R}^{h_{kv}Ld}$; $qk^\top \in \mathbf{R}^{h_qL}$, $O(qk^\top) = h_qLd$;
$o = \text{softmax}(\frac{qk^\top}{\sqrt{d}})v \in \mathbf{R}^{h_qd}, O(o) = h_qLd$

Attention Variant	MHA	GQA	MQA	MLA	Formulation
Arithmetic Intensity	$\frac{L}{1+L}$	$\frac{Lh_q}{h_q+h_{kv}L}$	$\frac{Lh_q}{h_q+L}$	$\frac{L}{1+\frac{L}{2h_q}}$	$\frac{2 \cdot L}{2+\frac{m_{kv}h_{kv}}{h_q}L}$
Roughly	≈ 1	$\approx \frac{h_q}{h_{kv}}$	$\approx h_q$	$\approx 2h_q$	$\approx \frac{2h_q}{m_{kv}h_{kv}}$

meta-llama/Llama-2-7b (MHA), $h_q = 32, h_{kv} = 32$, memory-bound;

openai/gpt-oss-120b (GQA), $h_q = 64, h_{kv} = 8$, memory-bound;

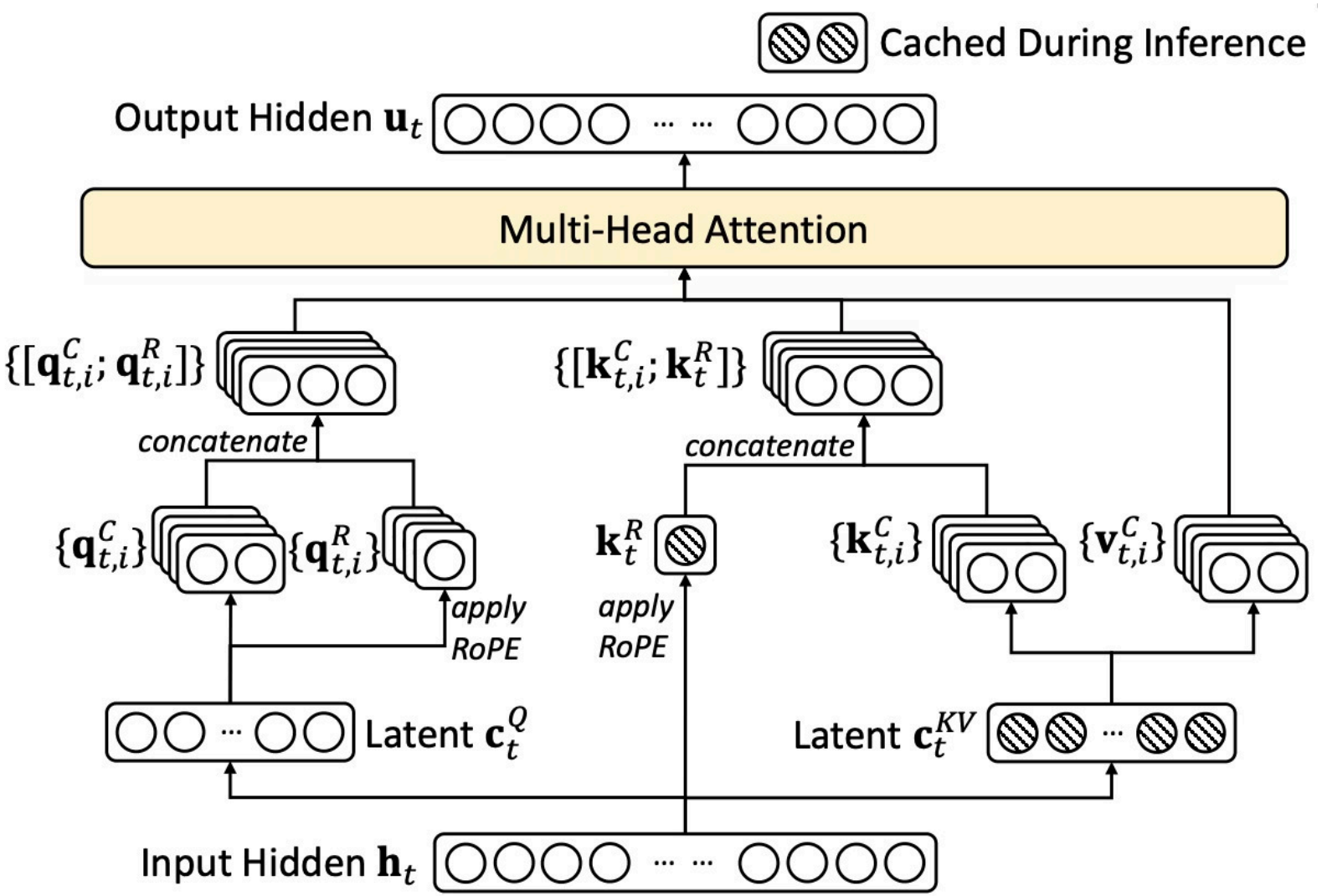
Qwen/Qwen3-235B-A22B (GQA), $h_q = 64, h_{kv} = 4$, memory-bound;

deepseek-ai/DeepSeek-V3 (MLA), $h_q = 128$, compute-bound;

moonshotai/Kimi-K2 (MLA), $h_q = 64$, memory-bound;

meituan/LongCat-Flash (MLA), $h_q = 64$, memory-bound.

Multi-Head Latent Attention (MLA)



hidden_size: 5120,

q_rank: 1536,

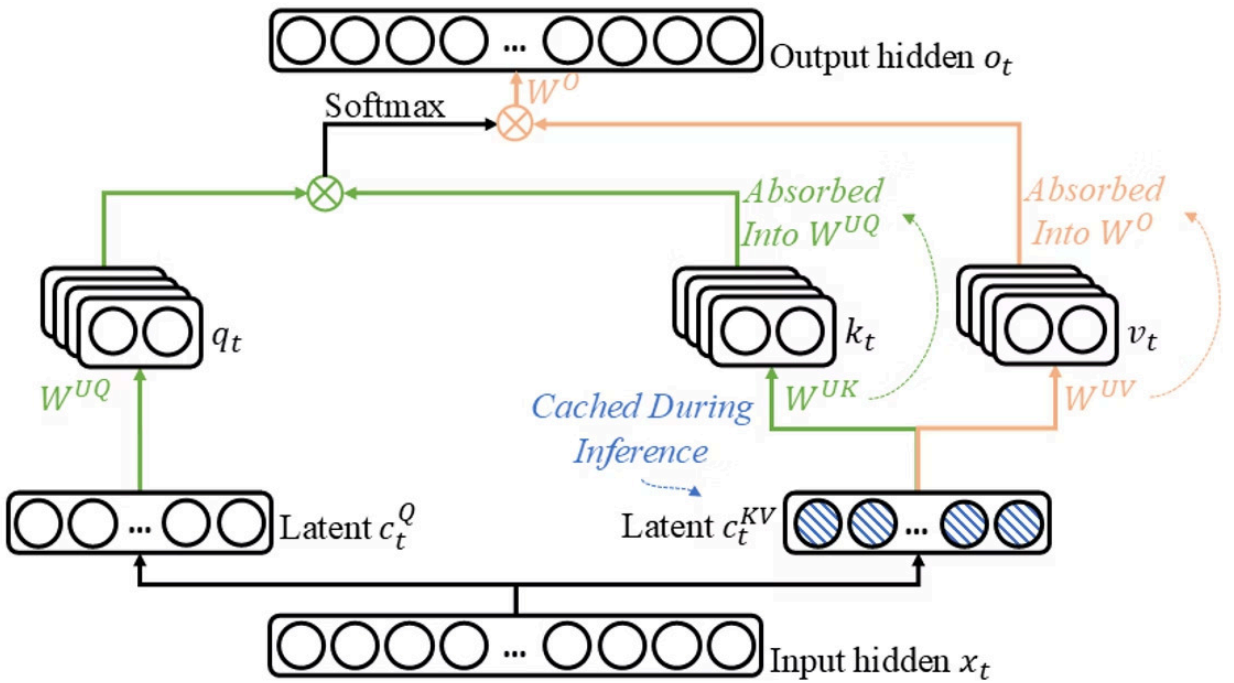
kv_rank: 512,

num_attention_heads: 128,

qk_nope_head_dim: 128,

qk_rope_head_dim: 64,

MLA Absorb



$$
\begin{aligned}
\mathbf{q}_i^\top \mathbf{k}_i &= \left(W_i^{UQ}\,\mathbf{c}_t^Q\right)^\top \left(W_i^{UK}\,\mathbf{c}_t^{KV}\right) \sim (MLA) \\
&= (\mathbf{c}_t^Q)^\top (W_i^{UQ})^\top W_i^{UK}\,\mathbf{c}_t^{KV} \\
&= (\mathbf{c}_t^Q)^\top (W_i^{UQK})^\top \mathbf{c}_t^{KV} \\
&= \left(W_i^{UQK}\,\mathbf{c}_t^Q\right)^\top \mathbf{c}_t^{KV} \quad \sim (MQA).
\end{aligned}
$$

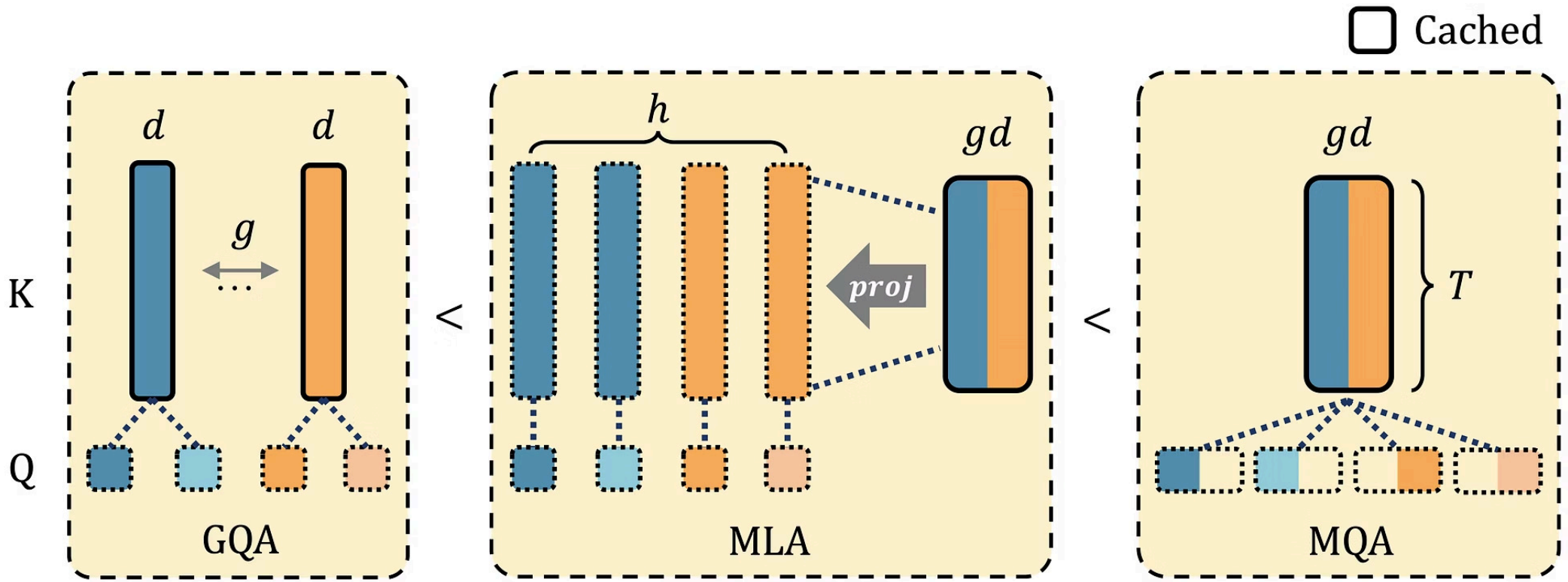
$$
W_i^{UQK} \in \mathbb{R}^{d_c^{kv} \times d_c^q} \leftarrow \begin{cases} W_i^{UQ} \in \mathbb{R}^{d_h \times d_c^q} \\ W_i^{UK} \in \mathbb{R}^{d_h \times d_c^{kv}} \end{cases}
$$

$$
d_c^{kv} = 512 > d_h = 128, \text{ computation cost increases}
$$

During training, Deepseek uses the MLA form, which is computation-efficient;

During inference, it uses the MQA form, which is memory-access-efficient.

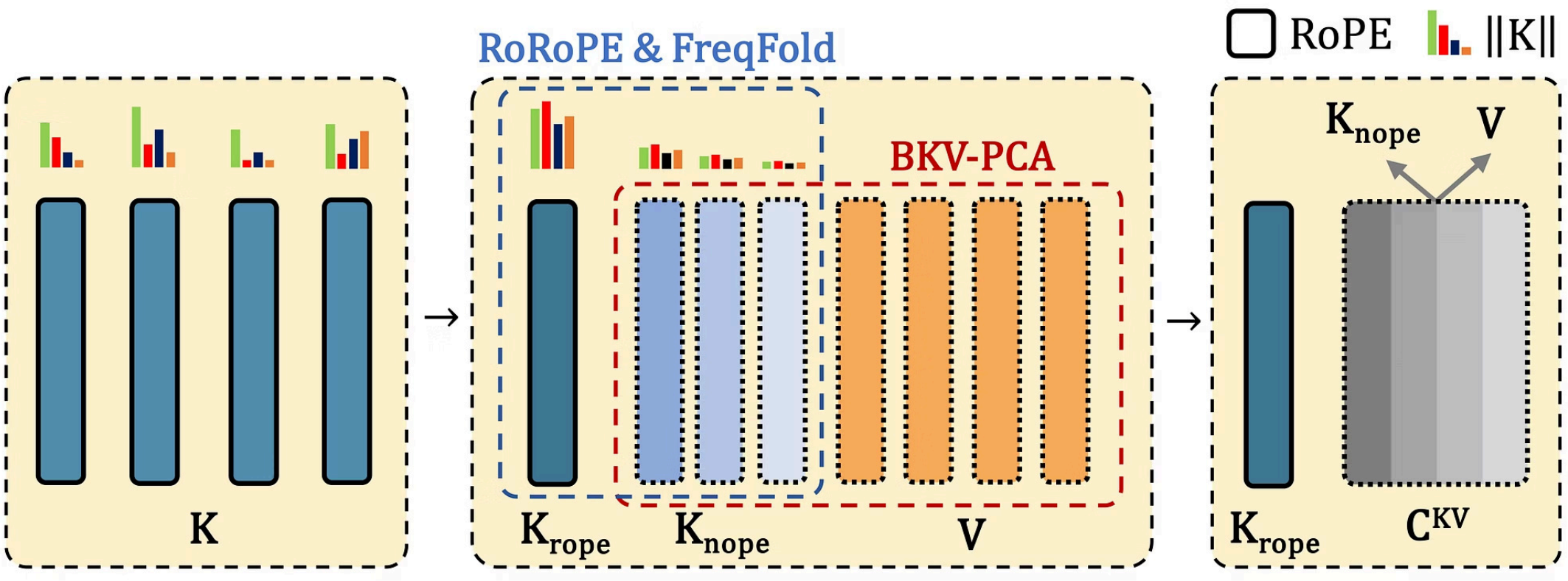
GQA<MLA<MQA



Theorem 1. For a fixed KV-cache budget, MLA is strictly more expressive than GQA.

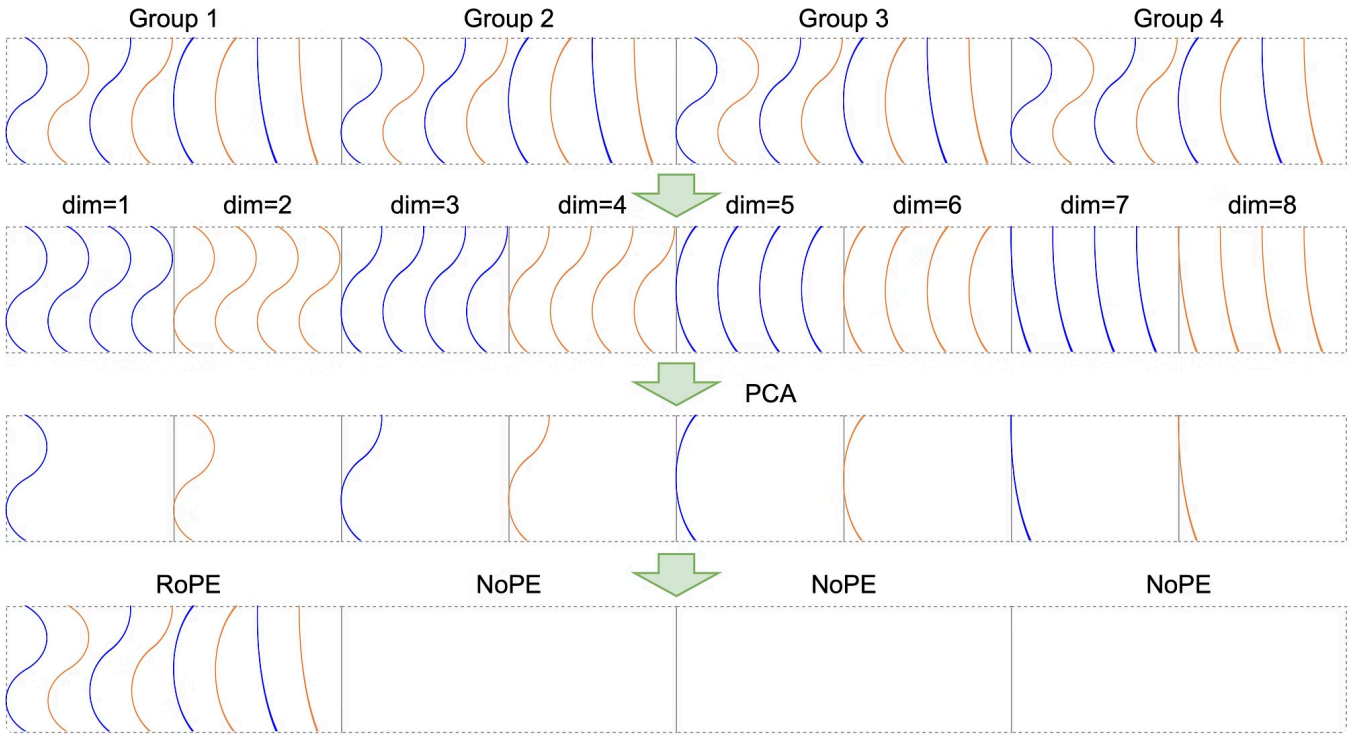
GQA layer can be rewritten as an MLA layer by introducing a single additional projection matrix. The reverse transformation is not always possible, implying that MLA subsumes GQA. When Rotary Positional Embeddings (RoPE) are present, the MLA equivalent must be expressed in the **absorbed** form.

TransMLA



TransMLA converts a GQA-based network into a DeepSeek-like MLA architecture, allowing the transformed model to run directly on DeepSeek's optimized inference stack and realize the full memory–latency benefits. Although one can build an MLA-equivalent representation of a GQA model, speedups arise only if the number of stored KV vectors is actually reduced.

RoRoPE



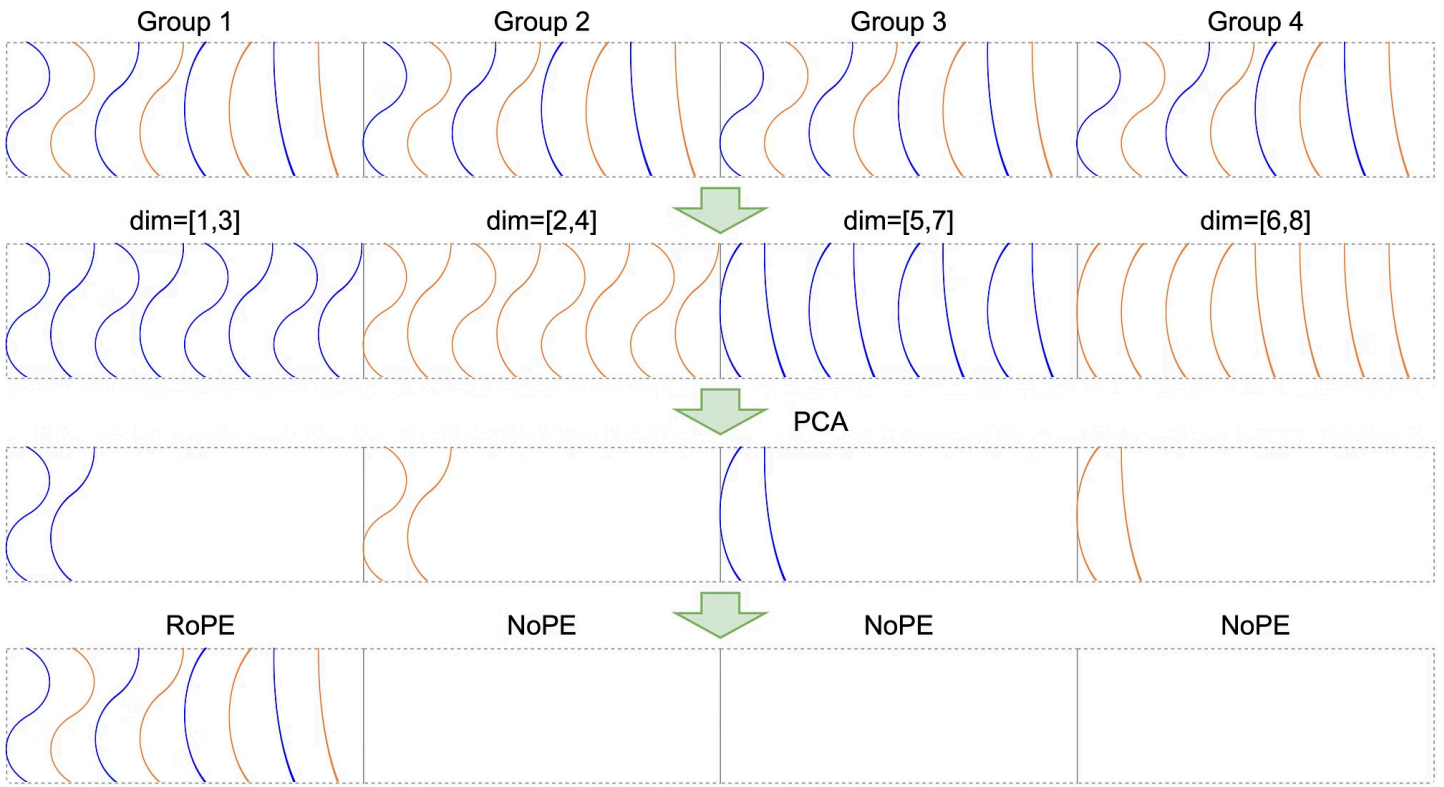
Theorem 2. The inner product between the query and key vectors remains invariant under a transformation by a rotation matrix $\mathbf{U}$, $\mathbf{U}^\top$, provided that $\mathbf{U}$ satisfies the specified two conditions:

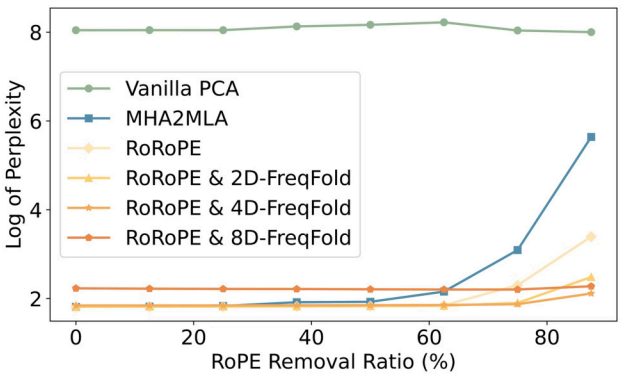
1. rotation occurs only within the same dimension across all attention heads,
2. the real and imaginary components of RoPE are rotated in the same manner.

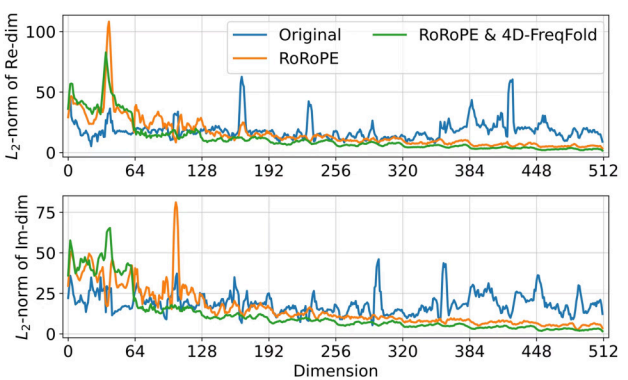
Step 1: merging grouped heads to a latent representation

Step 2: head-wise rotation for decoupled RoPE with minimal loss

FreqFold

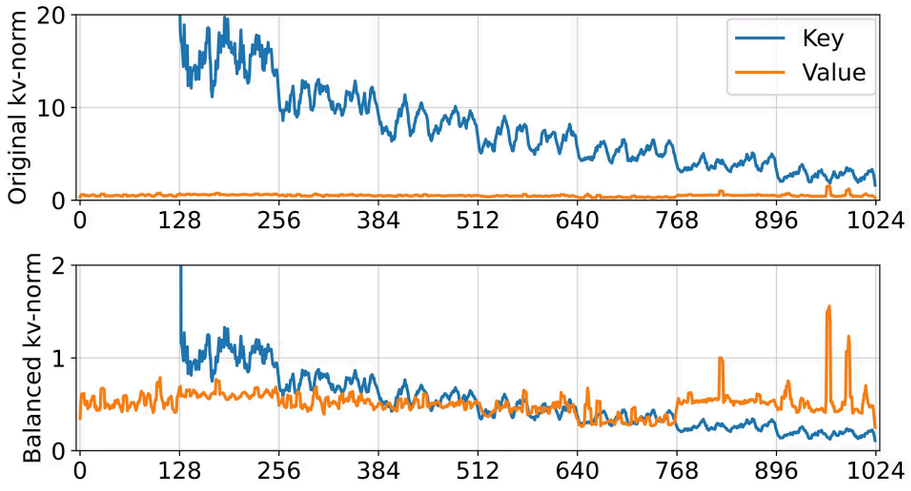


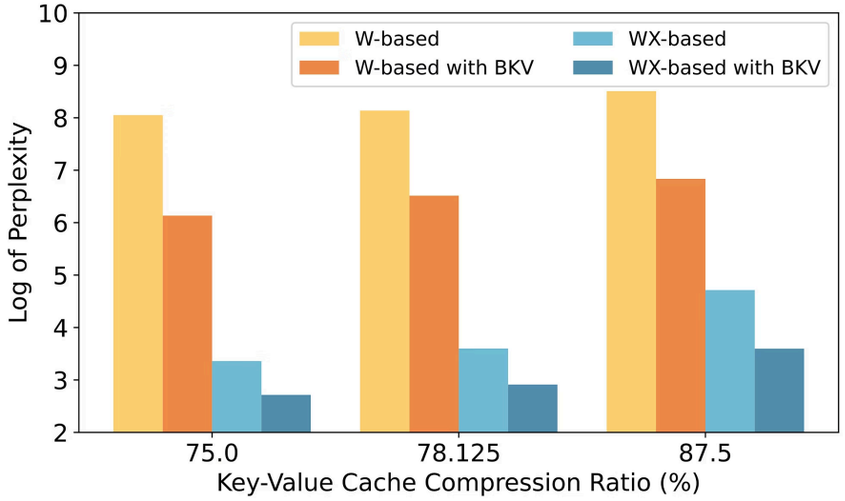




Using a one-dimensional space for all positional information proves limiting. To address this, we exploit the similar frequencies of adjacent dimensions in RoPE, treating them as equivalent positions. This allows us to use multiple dimensions within a single attention head to represent positional information

Balance-KV-PCA





To address the imbalance in the relative errors during reconstruction, we define the loss function to equally quantify the relative errors between the original and the approximated matrices. The loss is the sum of the relative errors for W_k and W_v, given by:

$$\mathcal{L}(a,b) = \| \frac{W_k - W_k'}{RMS(W_k)} \|_F^2 + \| \frac{W_v - W_v'}{RMS(W_v)} \|_F^2$$

The r-rank approximation of matrix W is given by:

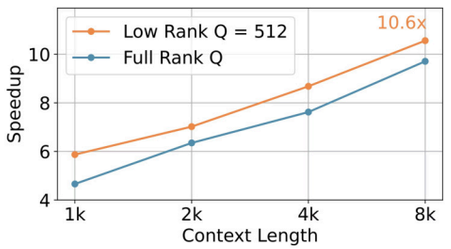
$$W' = \left[\frac{\sum_{i=1}^{r} u_i^k \sigma_i^k {v_i^k}^\top}{a}, \frac{\sum_{i=1}^{r} u_i^v \sigma_i^v {v_i^v}^\top}{b} \right],$$

where a and b are scaling factors that need to be optimized.

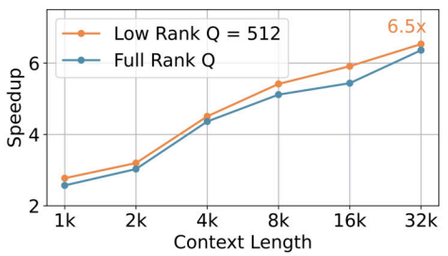
We derive the condition when the loss function reaches its minimum, specifically when the scaling factors (a) and (b) satisfy the ratio:

$$\frac{a}{b} = \frac{\text{RMS}(W_k)}{\text{RMS}(W_v)}$$

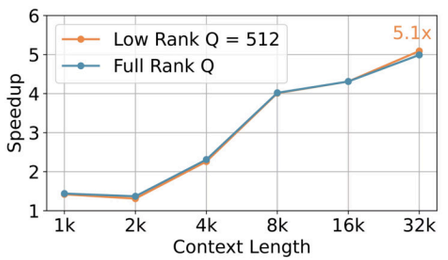
Experiments



a) 4090



b) A100



c) Ascend 910b

Model	Tokens	KV Mem.	Avg.	MMLU	ARC	PIQA	HS	OBQA	WG
LLaMA-2-7B	2T	–	59.85	41.43	59.24	78.40	73.29	41.80	64.96
- MHA2MLA	0	-68.75%	37.90	25.74	32.87	59.41	28.68	28.60	52.09
		-81.25%	34.02	25.50	26.44	53.43	27.19	22.60	49.01
		-87.50%	32.70	25.41	25.79	50.60	26.52	19.40	48.46
	6B	-68.75%	59.51	41.36	59.51	77.37	71.72	44.20	62.90
		-81.25%	59.61	40.86	59.74	77.75	70.75	45.60	62.98
		-87.50%	58.96	40.39	59.29	77.75	69.70	43.40	63.22
- TransMLA	0	-68.75%	58.20	39.90	57.66	77.48	70.22	41.00	62.90
		-87.50%	51.19	34.39	45.38	71.27	60.73	37.40	57.93
		-92.97%	43.26	28.93	36.32	63.38	45.87	31.60	53.43
	500M	-68.75%	59.82	40.87	59.18	77.91	71.82	45.20	63.93
	3B	-87.50%	59.36	40.77	58.84	78.18	71.28	43.60	63.46
	6B	-92.97%	59.19	40.41	58.68	77.53	70.39	45.00	63.14

Commonsense reasoning ability of two LLMs with TransMLA compared to MHA2MLA.

Tokens refers to the number of tokens used for further training after the TransMLA conversion. A value of 0 indicates that the model was evaluated immediately after conversion, without any fine-tuning.

Model Zoo

Model	Original ppl	Partial RoPE ppl	MLA ppl
Llama-2-7B	5.4732	8.9903	25.7731
Llama-3-8B	6.1371	8.3997	18.3500
Qwen2.5-7B	6.8480	7.3059	7.9812
Qwen2.5-72B-Instruct	4.2687	4.6931	7.7172
gemma-2-9b-it	10.1612	10.9948	22.0038
Mistral-7B-v0.3	5.3178	5.5915	7.0251
Mixtral-8x7B-v0.1	3.8422	4.1407	5.8374
MiMo-7B-Base	6.9108	7.9272	9.5810
GPT-OSS-20B	10.2563	9.2175	9.4072

Q&A