Multi-Head Low-Rank Attention

Anonymous authors
Paper under double-blind review

Abstract

Long-context inference in large language models is bottlenecked by Key-Value (KV) cache loading during the decoding stage, where the sequential nature of generation requires repeatedly transferring the KV cache from off-chip to on-chip memory at each step. Recent architectures like Multi-Head Latent Attention (MLA) significantly reduce the KV cache size to $4.5d_h$ per token per layer while maintaining high model quality. However, when using tensor parallelism (TP) with sufficient devices for inference, MLA still decodes slower than Grouped-Query Attention (GQA) because its single latent vector cannot be sharded, forcing each device to load $4.5d_h$ versus $2d_h$ for GQA. In this work, we propose Multi-Head Low-Rank Attention (MLRA), a TP-friendly attention mechanism that slashes the per-device KV cache under TP to just $1.5d_h$. Extensive experiments show that MLRA achieves state-of-the-art perplexity and downstream task performance, while also delivering a 2.8× decoding speedup over MLA.

1 Introduction

Inference-time scaling (OpenAI et al., 2024) is critical for large language models (LLMs) to produce high-quality responses. Both retrieval-augmented generation (RAG) (Lewis et al., 2020) and long chain-of-thought (CoT) reasoning (Wei et al., 2022) rely on maintaining long context before generating the final answer, substantially increasing the number of tokens that must be processed at each decoding step. Sequential token generation under Multi-Head Attention (MHA) (Vaswani et al., 2017) requires reloading the Key-Value (KV) cache from high-bandwidth memory every step, so data movement (Ivanov et al., 2021; Ootomo & Yokota, 2023; Gholami et al., 2024), not computation, dominates latency for long-context inference (Sadhukhan et al., 2025). The small amount of compute per step relative to this data movement leads to low arithmetic intensity (Williams et al., 2009) and poor GPU utilization (He & Zhai, 2024; Zadouri et al., 2025).

Recent work (Zadouri et al., 2025) proposes to analyze the performance and decoding efficiency of inference-aware attention mechanisms (Hu et al., 2024; Sun et al., 2025b; Zheng et al., 2025) along four axes: (1) model quality, (2) KV cache footprint per token, (3) arithmetic intensity, and (4) the degree of tensor parallelism (TP) (Pope et al., 2023) across attention heads. Multi-Query Attention (MQA) (Shazeer, 2019) shares single key and value heads across all query heads, reducing the KV cache to one head ($2d_h$) yet maintaining the same floating point operations per second (FLOPs) as MHA. Despite the higher arithmetic intensity and smaller KV cache than MHA, it often leads to noticeable quality degradation. Grouped-Query Attention (GQA) (Ainslie et al., 2023) shares single key and value heads within a group of query heads, improving model quality over MQA while achieving higher arithmetic intensity than MHA. However, its KV cache of $2gd_h$ still scales with the group size (with g=8 in LLaMA-3-70B (Llama et al., 2024) and Grok-2), which can be memory intensive. As a result, reaching the fastest decoding ($2d_h$ per device) typically requires g-way TP.

Multi-Head Latent Attention (MLA) (DeepSeek et al., 2024c) compresses the KV cache into a latent vector ($4.5d_h$ per token). By absorbing the up-projection matrices into the queries during decoding, its attention computation is similar to that of MQA with shared KV states, which increases arithmetic intensity by 2× over MQA and delivers better model quality compared with GQA and MHA. However, MLA is unfriendly to TP because its single latent vector cannot be sharded. To mitigate this, Grouped Latent Attention (GLA) (Zadouri et al., 2025) compresses the KV cache into two latent vectors, each $2d_h$, plus an additional $0.5d_h$ for partial RoPE (Su et al., 2024). Nevertheless, its per-device $2.5d_h$ KV cache still leaves decoding slower than GQA when enough devices are available. In this work, we target a high TP degree with per-device KV cache below $2d_h$.

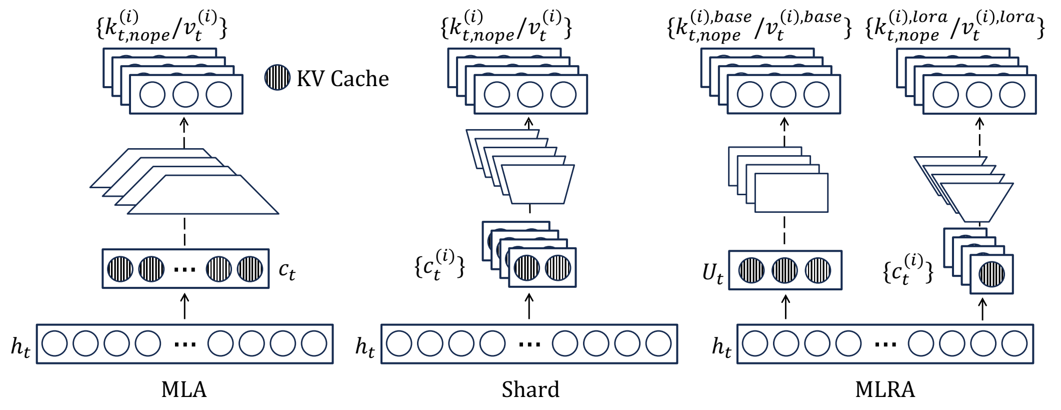
$\{k_{t,nope}^{(i)}/v_t^{(i)}\}$
KV Cache
c_t
h_t
MLA
$\{k_{t,nope}^{(i)}/v_t^{(i)}\}$
$\{c_t^{(i)}\}$
h_t
Shard
$\{k_{t,nope}^{(i),base}/v_t^{(i),base}\}$
$\{k_{t,nope}^{(i),lora}/v_t^{(i),lora}\}$
U_t
$\{c_t^{(i)}\}$
h_t
MLRA

Figure 1: Illustration of MLA, Shard, and MLRA. MLRA adopts a balanced design that achieves TP through the low-rank path while maintaining model quality via the base path.

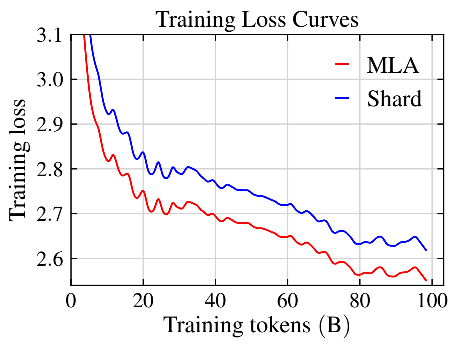

Figure 2: Training loss curves on FineWeb-Edu (100B tokens) for 354M models: MLA vs. Shard. Shard splits the latent vector into 16 smaller vectors.

A simple and straightforward approach is to split MLA's single high-dimensional latent vector into h smaller latent vectors (illustrated in the middle of Figure 1), as in GLA, which splits the latent vector into two smaller vectors. However, this simple approach can cause a significant drop in model quality, as illustrated by the 354M-model training loss curves in Figure 2. To address this limitation, we introduce Multi-Head Low-Rank Attention (MLRA), a dual-path attention mechanism shown on the right of Figure 1, which can achieve a small per-device KV cache under TP and high model quality. In the **low-rank** path, the KV cache is compressed to h tiny latent vectors, each of which is up-projected to single key and value heads, and can be partitioned across devices to support TP. In the **base** path, we follow MLA by compressing the KV cache into a base latent vector whose dimension matches the per-head query (d_h), then up-project it to produce h key and value heads. We then sum the attention outputs from the two paths. Therefore, our proposed MLRA can achieve a per-device KV cache of $1.5d_h$ with 4-way TP. Based on our 2.9B scale experiments, MLRA achieves the lowest perplexity (12.998 vs. 13.115 for MLA and 13.399 for GQA) and highest downstream accuracy (57.06% vs. 55.46% for MLA and 55.68% for GQA). Our kernel delivers a 1.05-1.26$\times$ speedup over GQA in long-context decoding.

2 Preliminaries

2.1 Tensor Parallelism

Long-context decoding is bottlenecked by KV cache loading from high-bandwidth memory at each step. Given the sequential nature of generation, we can only accelerate inference by distributing the KV cache and model weights across devices via tensor parallelism. With high-bandwidth GPU interconnects such as NVLink, the synchronization overhead in TP can be substantially reduced.

2.2 Translation Equivariance

Equivariance is a fundamental property in geometric systems such as molecules, where vector features such as atomic forces or dipole moments must transform consistently with the coordinate system (Thomas et al., 2018; Weiler et al., 2018; Fuchs et al., 2020; Satorras et al., 2021). In the context of sequence models, a common transformation is sequence translation. Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in$

X be a sequence of tokens, and define a translation operator $T_s : X \to X$ that translates the entire sequence by s positions. Let $\phi : X \to Y$ be a function that maps a sequence to a matrix of attention scores $\phi(\mathbf{x}) \in \mathbb{R}^{n \times n}$, where each element $\phi(\mathbf{x})_{i,j} = A(\mathbf{x}_i, \mathbf{x}_j)$ denotes the attention score between tokens $\mathbf{x}_i$ and $\mathbf{x}_j$. We say that ϕ is *translation equivariant* if there exists a corresponding output-space transformation $S_s : Y \to Y$ such that:

$$\phi(T_s(\mathbf{x})) = S_s(\phi(\mathbf{x})), \quad \forall s. \tag{1}$$

This property ensures that the attention score between two tokens depends only on their relative positions, not their absolute positions. That is crucial for batch inference using left padding, where sequences of different lengths are offset to align ends. The first non-padding token of a sequence is no longer at position 0, yet attention scores remain invariant to this translation.

2.3 Rotary Position Embedding

Rotary Position Embedding (RoPE) (Su et al., 2024) is a positional encoding method designed to incorporate relative position information directly into the attention mechanism. We show in this section that RoPE is translation equivariant.

Theorem 1. *Given two tokens with query $\mathbf{q}$ and key $\mathbf{k}$ at positions m and n, respectively, and applying RoPE to obtain $\mathrm{RoPE}(\mathbf{q}, m)$ and $\mathrm{RoPE}(\mathbf{k}, n)$, we aim to prove that RoPE is translation equivariant with respect to the inner product. In the definition of translation equivariance (Eq.(1)), this corresponds to setting the output transformation S_s as the identity map, i.e., $S_s = \mathbf{I}$. Specifically, we show that translating both positions by an offset s leaves the inner product unchanged:*

$$\langle \mathrm{RoPE}(\mathbf{q}, m+s), \mathrm{RoPE}(\mathbf{k}, n+s) \rangle = \langle \mathrm{RoPE}(\mathbf{q}, m), \mathrm{RoPE}(\mathbf{k}, n) \rangle .$$

Proof can be found in Appendix A.

Remark 1. *While RoPE is translation equivariant, applying a linear projection $\mathbf{W}$ after RoPE generally breaks this property. Specifically, suppose we define the inner product after RoPE and linear projection as:*

$$\langle \mathrm{RoPE}(\mathbf{q}, m)\mathbf{W}_Q,\ \mathrm{RoPE}(\mathbf{k}, n)\mathbf{W}_K \rangle = \mathbf{q}\mathbf{R}_m \mathbf{W}_Q \mathbf{W}_K^\top \mathbf{R}_n^\top \mathbf{k}^\top . \tag{2}$$

The term $\mathbf{W}_{\mathrm{Q}}\mathbf{W}_{\mathrm{K}}^\top$ breaks translation equivariance by disrupting the expression's dependence on the relative position $m-n$. The property would only be preserved in the specific case where $\mathbf{W}_{\mathrm{Q}}\mathbf{W}_{\mathrm{K}}^\top = \mathbf{I}$, which would reduce the expression to its original form. However, since this constraint is difficult to enforce during training, translation equivariance is generally lost when applying a linear projection after RoPE.

3 Multi-Head Low-Rank Attention

3.1 Implementation

As noted in Remark 1, applying a linear projection after RoPE destroys translation equivariance. Therefore, we employ MLA's partial RoPE (DeepSeek et al., 2024c). We first apply a linear down-projection over the hidden states $\mathbf{H} \in \mathbb{R}^{n \times d}$ for an n-token sequence:

$$\left[\mathbf{U}_{\mathrm{KV}},\ \overline{\mathbf{C}}_{\mathrm{KV}},\ \widetilde{\mathbf{K}}_{\mathrm{RoPE}}\right] = \mathbf{H}\mathbf{A}_{\mathrm{KV}}, \tag{3}$$

where $\mathbf{A}_{\mathrm{KV}} \in \mathbb{R}^{d \times \left(d_u + hr + d_h^R\right)}$ is a down-projection matrix. By default we set $d_u = d_h, r = \frac{3d_h}{h}$, and $d_h^R = 0.5d_h$ so that the total KV cache size matches that of MLA. The projected output is then partitioned as follows: a base latent head $\mathbf{U}_{\mathrm{KV}} \in \mathbb{R}^{n \times d_u}$ for the base path; $\overline{\mathbf{C}}_{\mathrm{KV}} \in \mathbb{R}^{n \times hr}$, reshaped to $\mathbf{C}_{\mathrm{KV}} \in \mathbb{R}^{n \times h \times r}$ to yield h tiny latent heads for the low-rank path; and $\widetilde{\mathbf{K}}_{\mathrm{RoPE}} \in \mathbb{R}^{n \times d_h^R}$, which is RoPE-encoded to obtain $\mathbf{K}_{\mathrm{RoPE}}$.

Base Path. We up-project the base latent head $\mathbf{U}_{\mathrm{KV}} \in \mathbb{R}^{n \times d_u}$ with the up-projection matrix $\mathbf{B}_{\mathrm{base}} \in \mathbb{R}^{d_u \times (2hd_h)}$ to produce concatenated keys and values:

$$\left[\overline{\mathbf{K}}_{\mathrm{base}},\ \overline{\mathbf{V}}_{\mathrm{base}}\right] = \mathbf{U}_{\mathrm{KV}}\,\mathbf{B}_{\mathrm{base}} \in \mathbb{R}^{n \times (2hd_h)}. \tag{4}$$

We then split and reshape this output to obtain h heads of keys $\widetilde{\mathbf{K}}_{\mathrm{base}} \in \mathbb{R}^{n \times h \times d_h}$ and values $\widetilde{\mathbf{V}}_{\mathrm{base}} \in \mathbb{R}^{n \times h \times d_h}$ for the base path.

Low-Rank Path. We up-project the h tiny latent heads using a head-wise matrix $\mathbf{B}_{\text{lora}} \in \mathbb{R}^{h\times r\times 2d_h}$ to form concatenated keys and values via Einstein summation over the latent dimension:

$$\left[\overline{\mathbf{K}}_{\text{lora}},\ \overline{\mathbf{V}}_{\text{lora}}\right] = \text{einsum}\left(\texttt{"nhr,hrd}\rightarrow\texttt{nhd"},\ \mathbf{C}_{\text{KV}},\ \mathbf{B}_{\text{lora}}\right), \tag{5}$$

where the result is then split along the last dimension to obtain h heads of keys $\widetilde{\mathbf{K}}_{\text{lora}} \in \mathbb{R}^{n\times h\times d_h}$ and values $\widetilde{\mathbf{V}}_{\text{lora}} \in \mathbb{R}^{n\times h\times d_h}$ for the low-rank path.

Attention. To preserve translation equivariance, we concatenate the keys with partial RoPE $\mathbf{K}_{\text{RoPE}}$ in both the base and low-rank paths.

$$\mathbf{K}_{\text{base}} = \text{Concat}\left[\widetilde{\mathbf{K}}_{\text{base}},\ \text{repeat}\left(\mathbf{K}_{\text{RoPE}},\ h\right)\right], \qquad \mathbf{K}_{\text{lora}} = \text{Concat}\left[\alpha\widetilde{\mathbf{K}}_{\text{lora}},\ \text{repeat}\left(\mathbf{K}_{\text{RoPE}},\ h\right)\right], \tag{6}$$

where $\text{repeat}\left(\mathbf{K}_{\text{RoPE}},\ h\right)$ replicates the partial RoPE across h heads to match the dimensionality required for concatenation. The scalar α is a tunable hyperparameter that controls the scale of the low-rank keys. Finally, we sum the base and low-rank attention outputs for each head i:

$$\textbf{head}^{(i)} = \text{Attention}\left(\mathbf{Q}^{(i)}, \mathbf{K}^{(i)}_{\text{base}}, \mathbf{V}^{(i)}_{\text{base}}\right) + \text{Attention}\left(\mathbf{Q}^{(i)}, \mathbf{K}^{(i)}_{\text{lora}}, \alpha\mathbf{V}^{(i)}_{\text{lora}}\right), \tag{7}$$

where the computation of $\mathbf{Q}$ can be found in Appendix B.

3.2 Decoding without KV Materialization

We refer to the DeepSeek official inference implementation (DeepSeek et al., 2024b) to illustrate how to "absorb" up-projection matrices into the queries and attention output to avoid explicit KV materialization in our MLRA decoding. Assume we have a query $\mathbf{q} = \text{Concat}\left[\mathbf{q}_{\text{NoPE}},\ \mathbf{q}_{\text{RoPE}}\right]$ and KV cache $\mathbf{U}_{\text{KV}},\ \mathbf{C}_{\text{KV}},\ \mathbf{K}_{\text{RoPE}}$. We split the up-projection matrices $\mathbf{B}_{\text{base}}$ and $\mathbf{B}_{\text{lora}}$ into per-head blocks, yielding $\mathbf{B}^{(i)}_{\text{base,K}}, \mathbf{B}^{(i)}_{\text{base,V}} \in \mathbb{R}^{d_u\times d_h}$ and $\mathbf{B}^{(i)}_{\text{lora,K}}, \mathbf{B}^{(i)}_{\text{lora,V}} \in \mathbb{R}^{r\times d_h}$. We likewise partition $\mathbf{C}_{\text{KV}}$ along the head dimension as $\mathbf{C}_{\text{KV}} = \left[\mathbf{C}^{(1)}_{\text{KV}}\cdots\mathbf{C}^{(h)}_{\text{KV}}\right]$ with $\mathbf{C}^{(i)}_{\text{KV}} \in \mathbb{R}^{n\times r}$. For head i, letting $\tau = \frac{1}{\sqrt{d_h + d_h^R}}$, we compute the attention output for the base path:

$$\begin{aligned}
&\text{Softmax}\left(\tau\,\text{Concat}\left[\mathbf{q}^{(i)}_{\text{NoPE}}, \mathbf{q}^{(i)}_{\text{RoPE}}\right]\left(\text{Concat}\left[\mathbf{U}_{\text{KV}}\mathbf{B}^{(i)}_{\text{base,K}}, \mathbf{K}_{\text{RoPE}}\right]\right)^{\top}\right)\left(\mathbf{U}_{\text{KV}}\mathbf{B}^{(i)}_{\text{base,V}}\right)\\
=&\text{Softmax}\left(\tau\mathbf{q}^{(i)}_{\text{NoPE}}\left(\mathbf{U}_{\text{KV}}\mathbf{B}^{(i)}_{\text{base,K}}\right)^{\top} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right)\left(\mathbf{U}_{\text{KV}}\mathbf{B}^{(i)}_{\text{base,V}}\right)\\
=&\text{Softmax}\left(\tau\mathbf{q}^{(i)}_{\text{NoPE}}\left(\mathbf{B}^{(i)}_{\text{base,K}}\right)^{\top}\mathbf{U}^{\top}_{\text{KV}} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right)\left(\mathbf{U}_{\text{KV}}\mathbf{B}^{(i)}_{\text{base,V}}\right)\\
=&\text{Softmax}\left(\tau\,\underbrace{\mathbf{q}^{(i)}_{\text{NoPE}}\left(\mathbf{B}^{(i)}_{\text{base,K}}\right)^{\top}}_{\tilde{\mathbf{q}}^{(i)}_{\text{base}}\in\mathbb{R}^{d_u}}\mathbf{U}^{\top}_{\text{KV}} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right)\mathbf{U}_{\text{KV}}\mathbf{B}^{(i)}_{\text{base,V}}.
\end{aligned} \tag{8}$$

Similarly, we compute the attention output for the low-rank path:

$$\begin{aligned}
&\text{Softmax}\left(\tau\,\text{Concat}\left[\mathbf{q}^{(i)}_{\text{NoPE}}, \mathbf{q}^{(i)}_{\text{RoPE}}\right]\left(\text{Concat}\left[\alpha\mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{lora,K}}, \mathbf{K}_{\text{RoPE}}\right]\right)^{\top}\right)\left(\alpha\mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{lora,V}}\right)\\
=&\text{Softmax}\left(\tau\alpha\mathbf{q}^{(i)}_{\text{NoPE}}\left(\mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{lora,K}}\right)^{\top} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right)\left(\alpha\mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{lora,V}}\right)\\
=&\text{Softmax}\left(\tau\alpha\mathbf{q}^{(i)}_{\text{NoPE}}\left(\mathbf{B}^{(i)}_{\text{lora,K}}\right)^{\top}\left(\mathbf{C}^{(i)}_{\text{KV}}\right)^{\top} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right)\left(\alpha\mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{lora,V}}\right)\\
=&\text{Softmax}\left(\tau\,\underbrace{\alpha\mathbf{q}^{(i)}_{\text{NoPE}}\left(\mathbf{B}^{(i)}_{\text{lora,K}}\right)^{\top}}_{\tilde{\mathbf{q}}^{(i)}_{\text{lora}}\in\mathbb{R}^{r}}\left(\mathbf{C}^{(i)}_{\text{KV}}\right)^{\top} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right)\left(\alpha\mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{lora,V}}\right).
\end{aligned} \tag{9}$$

Below, we show how to avoid materializing h heads of keys and values during decoding by exploiting matrix multiplication associativity. Pseudocode can be found in Appendix D.

Table 1: Comparison of parameters and KV cache among attention mechanisms. Some results are taken from Zhang et al. (2025) and Zadouri et al. (2025). For attention mechanism details, refer to Appendix C.

Method	KV Cache	# Parameters	# Query Heads	# KV Heads
MHA (Vaswani et al., 2017)	$2hd_h$	$4dhd_h$	h	h
MQA (Shazeer, 2019)	$2d_h$	$(2d + 2d_h)\, hd_h$	h	1
GQA (Ainslie et al., 2023)	$2gd_h$	$(2d + 2gd_h)\, hd_h$	h	g
MLA (DeepSeek et al., 2024c)	$d_c + d_h^R$	$d_c'\big(d + hd_h + hd_h^R\big) + dd_h^R + d_c(d + 2hd_h) + dhd_h$	h	h
TPA (Zhang et al., 2025)	$2R_{\text{KV}}\,(h + d_h)$	$d(R_{\text{Q}} + 2R_{\text{KV}})\,(h + d_h) + dhd_h$	h	h
GLA (Zadouri et al., 2025)	$d_c + d_h^R$	$d_c'\big(d + hd_h + hd_h^R\big) + dd_h^R + d_c(d + hd_h) + dhd_h$	h	h
MLRA	$d_u + hr + d_h^R$	$d\big(d_u + hr + d_h^R\big) + 2d_uhd_h + 2hrd_h + d(d_u' + hr') + d_u'h\,\big(d_h + d_h^R\big) + hr'\big(d_h + d_h^R\big) + dhd_h$	h	h

Method	KV Cache Per Token	KV Cache Per Token Per Device (2 GPUs)	KV Cache Per Token Per Device (4 GPUs)	KV Cache per token Per Device (8 GPUs)
MHA (Vaswani et al., 2017)	$64d_h$	$32d_h$	$16d_h$	$8d_h$
MQA (Shazeer, 2019)	$2d_h$	$2d_h$	$2d_h$	$2d_h$
GQA (Ainslie et al., 2023)	$16d_h$	$8d_h$	$4d_h$	$2d_h$
MLA (DeepSeek et al., 2024c)	$4.5d_h$	$4.5d_h$	$4.5d_h$	$4.5d_h$
TPA (Zhang et al., 2025)	$4d_h + 256$	$4d_h + 128$	$4d_h + 64$	$4d_h + 32$
GLA (Zadouri et al., 2025)	$4.5d_h$	$2.5d_h$	$2.5d_h$	$2.5d_h$
MLRA	$4.5d_h$	$2.5d_h$	$1.5d_h$	$1.5d_h$

Step 1 (Keys/Logits). We exploit matrix-multiplication associativity by multiplying the query and the key up-projection matrix:

$$\tilde{\mathbf{q}}_{\text{base}}^{(i)} = \mathbf{q}_{\text{NoPE}}^{(i)}\big(\mathbf{B}_{\text{base,K}}^{(i)}\big)^{\top} \in \mathbb{R}^{1\times d_u}, \quad \boldsymbol{\lambda}_{\text{base}}^{(i)} = \text{Softmax}\Big(\tau\tilde{\mathbf{q}}_{\text{base}}^{(i)}\mathbf{U}_{\text{KV}}^{\top} + \tau\mathbf{q}_{\text{RoPE}}^{(i)}\mathbf{K}_{\text{RoPE}}^{\top}\Big)\,, \tag{10}$$

$$\tilde{\mathbf{q}}_{\text{lora}}^{(i)} = \alpha\mathbf{q}_{\text{NoPE}}^{(i)}\big(\mathbf{B}_{\text{lora,K}}^{(i)}\big)^{\top} \in \mathbb{R}^{1\times r}, \quad \boldsymbol{\lambda}_{\text{lora}}^{(i)} = \text{Softmax}\left(\tau\tilde{\mathbf{q}}_{\text{lora}}^{(i)}\left(\mathbf{C}_{\text{KV}}^{(i)}\right)^{\top} + \tau\mathbf{q}_{\text{RoPE}}^{(i)}\mathbf{K}_{\text{RoPE}}^{\top}\right). \tag{11}$$

This absorbs the key up-projection matrices into queries and avoids explicitly materializing keys $\mathbf{K}_{\text{base}}^{(i)} = \mathbf{U}_{\text{KV}}\mathbf{B}_{\text{base,K}}^{(i)} \in \mathbb{R}^{n\times 1\times d_h}$, $\mathbf{K}_{\text{lora}}^{(i)} = \alpha\mathbf{C}_{\text{KV}}^{(i)}\mathbf{B}_{\text{lora,K}}^{(i)} \in \mathbb{R}^{n\times 1\times d_h}$. Note that $\mathbf{U}_{\text{KV}}$ and $\mathbf{K}_{\text{RoPE}}$ are shared across the h absorbed-query heads, so the base-path KV cache load is only $n \times \big(d_u + d_h^R\big)$.

Step 2 (Values/Output). After computing the attention scores, we use them to perform a weighted aggregation of compressed KV, then up-project the aggregated output:

$$\tilde{\mathbf{z}}_{\text{base}}^{(i)} = \boldsymbol{\lambda}_{\text{base}}^{(i)}\,\mathbf{U}_{\text{KV}} \in \mathbb{R}^{1\times d_u}, \qquad \mathbf{z}_{\text{base}}^{(i)} = \tilde{\mathbf{z}}_{\text{base}}^{(i)}\,\mathbf{B}_{\text{base,V}}^{(i)} \in \mathbb{R}^{1\times d_h}, \tag{12}$$

$$\tilde{\mathbf{z}}_{\text{lora}}^{(i)} = \boldsymbol{\lambda}_{\text{lora}}^{(i)}\,\mathbf{C}_{\text{KV}}^{(i)} \in \mathbb{R}^{1\times r}, \qquad \mathbf{z}_{\text{lora}}^{(i)} = \alpha\,\tilde{\mathbf{z}}_{\text{lora}}^{(i)}\,\mathbf{B}_{\text{lora,V}}^{(i)} \in \mathbb{R}^{1\times d_h}. \tag{13}$$

By deferring the up-projection for values with $\mathbf{B}_{\text{base,V}}^{(i)}$ and $\mathbf{B}_{\text{lora,V}}^{(i)}$, we never materialize $\mathbf{V}_{\text{base}}^{(i)} = \mathbf{U}_{\text{KV}}\mathbf{B}_{\text{base,V}}^{(i)} \in \mathbb{R}^{n\times 1\times d_h}$ or $\mathbf{V}_{\text{lora}}^{(i)} = \mathbf{C}_{\text{KV}}^{(i)}\mathbf{B}_{\text{lora,V}}^{(i)} \in \mathbb{R}^{n\times 1\times d_h}$. While the KV states are shared ($\mathbf{K} = \mathbf{V}$), FlashMLA (Jiashi Li, 2025) uses online softmax to load them once and avoid reloading.

3.3 Analysis

Translation Equivariance. We analyze the translation equivariance property of MLRA. Let $\mathbf{q}_m^{(i)} = \text{Concat}\left(\mathbf{Q}_{\text{NoPE}}\left[m, i, :\right], \mathbf{Q}_{\text{RoPE}}\left[m, i, :\right]\right)$ and $\mathbf{k}_n^{(i)} = \text{Concat}\left(\mathbf{K}_{\text{NoPE}}\left[n, i, :\right], \mathbf{K}_{\text{RoPE}}\left[n, :\right]\right)$ denote the query and key vectors for head i at positions m and n, respectively. $\mathbf{K}_{\text{NoPE}}$ is either $\mathbf{K}_{\text{base}}$ or $\mathbf{K}_{\text{lora}}$. The attention score for this head is given by the inner product:

$$\left\langle \mathbf{q}_m^{(i)},\, \mathbf{k}_n^{(i)} \right\rangle = \left\langle \mathbf{Q}_{\text{NoPE}}\left[m, i, :\right],\, \mathbf{K}_{\text{NoPE}}\left[n, i, :\right] \right\rangle + \left\langle \mathbf{Q}_{\text{RoPE}}\left[m, i, :\right],\, \mathbf{K}_{\text{RoPE}}\left[n, :\right] \right\rangle. \tag{14}$$

Between the two terms, the second term with RoPE is position-dependent yet translation equivariant, due to RoPE's translation equivariance. The first term is position-independent and thus unchanged under joint translations of m and n. Therefore, the attention score $\left\langle \mathbf{q}_m^{(i)},\, \mathbf{k}_n^{(i)} \right\rangle$ is invariant to translation in input positions. Although MLRA introduces positional inductive bias via partial RoPE and is translation equivariant, we refer to this property as semi-translation equivariance to distinguish it from the full RoPE with translation equivariance. Further details on other attention mechanisms and their translation equivariance analysis are provided in Appendix C.

Table 2: Comparison of arithmetic intensity among attention mechanisms.

Method	MHA	MQA	GQA	MLA	TPA	GLA	MLRA
Arithmetic Intensity	$\frac{4nhd_h}{4nhd_h}$ ≈ 1	$\frac{4nhd_h}{4nd_h}$ $\approx h$	$\frac{4nhd_h}{4ngd_h}$ $\approx \frac{h}{g}$	$\frac{4nhd_c+2nhd_h^R}{2n\left(d_c+d_h^R\right)}$ $\approx 2h$	$\frac{4nhR_{\mathrm{KV}}d_h+4nhd_h}{4nR_{\mathrm{KV}}(h+d_h)}$ $\approx \frac{3h}{4}$	$\frac{2nhd_c+2nhd_h^R}{2n\left(d_c+d_h^R\right)}$ $\approx h$	$\frac{4nhr+4nhd_h+2nhd_h^R}{2n\left(hr+d_h+d_h^R\right)}$ $\approx \frac{h}{2}$

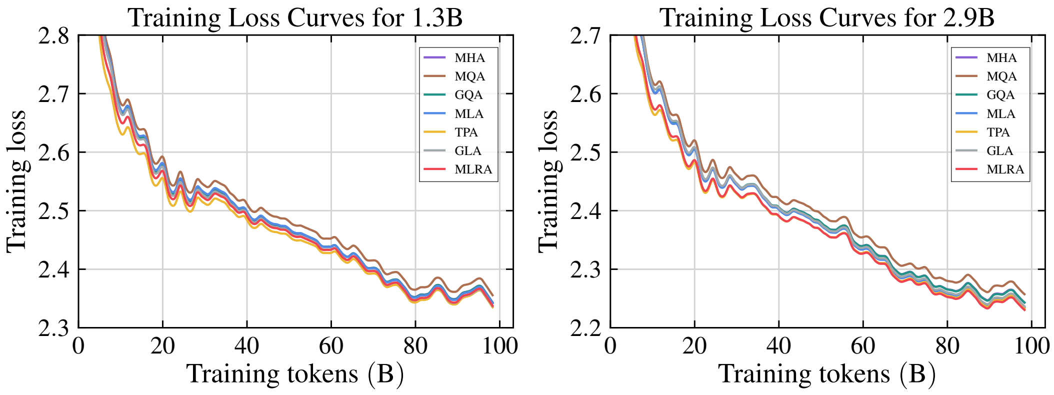

Figure 3: Training loss curves at the 1.3B and 2.9B scales for various attention mechanisms on the FineWeb-Edu 100B dataset.

KV Cache. We cache $\mathbf{U}_{\mathrm{KV}}$, $\mathbf{C}_{\mathrm{KV}}$, and $\mathbf{K}_{\mathrm{RoPE}}$ during inference. Table 1 summarizes the total KV cache and parameters for the different attention mechanisms. While $\mathbf{C}_{\mathrm{KV}}$ can be sharded across heads, $\mathbf{U}_{\mathrm{KV}}$ cannot. To support TP, we place $\mathbf{U}_{\mathrm{KV}}$ on one device and partition $\mathbf{C}_{\mathrm{KV}}$ across several other devices; consequently, $\mathbf{K}_{\mathrm{RoPE}}$ is replicated on every device. Therefore, KV cache under φ-way ($\varphi > 1$) TP is $\frac{d_u+hr}{\varphi} + d_h^R$ per device for the low-rank path and $1.5d_h$ for the base path. We evaluate per-device KV cache under TP using LLaMA-3-70B (Llama et al., 2024) and Grok-2 as base models. Both models have 64 query heads and 8 key-value heads, so $g = 8$ for GQA. For GLA, we follow the paper's setup and consider two latent heads, each with a dimension of $2d_h$. For TPA, we follow the paper's setup and consider $R_{\mathrm{KV}} = 2$.

Table 1 summarizes the per-device KV cache under TP as the number of devices varies. To support TP, the official MLA decoding implementation, FlashMLA (Jiashi Li, 2025), distributes up-projection matrices across devices by head. However, this approach replicates the KV cache on each device, so the per-device KV cache remains $4.5d_h$. TPA constructs its h key-value heads as linear combinations of R_{KV} shared heads. It supports TP only for the combination coefficients, while the shared heads are replicated across devices. So the per-device KV cache is $4d_h + \frac{256}{\varphi}$. GLA splits the latent vector into two smaller latent vectors, and the per-device KV cache is $2.5d_h$ with 2-way TP. Note that for MLA with TP > 1 and for GLA with TP > 2, the KV cache is replicated, so the per-device KV cache remains $4.5d_h$ and $2.5d_h$, respectively. For MLRA, when TP > 4, we can shard $\mathbf{C}_{\mathrm{KV}}$ across additional devices and only replicate the partial RoPE. While GQA needs 8-way TP to reach $2d_h$ per device, MLRA achieves $1.5d_h$ with just 4-way TP, yielding the lowest per-device KV cache and faster decoding than other attention mechanisms.

Arithmetic Intensity. Arithmetic intensity (AI) is measured through the ratio of arithmetic operations to memory access (FLOPs per byte), which helps determine whether a workload is memory-bound or compute-bound (Zadouri et al., 2025). We evaluate the arithmetic intensity of the attention mechanisms with $d_h = h = 128$ and $R_{\mathrm{KV}} = 2$. Our analysis focuses on the long-context decoding, where the context length n dominates all other factors. We report AI of MLA, GLA, and MLRA without KV materialization. Implementation details of MLA decoding are provided in Appendix C.3. As shown in Table 2, our MLRA achieves relatively high AI. Under TP, the workload is imbalanced across devices. In the base path, the AI is approximately $2h$, matching that of MLA. In the low-rank path, if inference uses p devices, the AI is about $\frac{h}{p+6}$. This asymmetry motivates a heterogeneous deployment (Zhao et al., 2024b; Griggs et al., 2024; Jiang et al., 2024c; 2025).

Table 3: Validation perplexity (lower is better) at the 1.3B and 2.9B scales on six datasets: FineWeb-Edu, Wikipedia, C4, CommonCrawl, Pile, and Arxiv. Best is **bold**; second best is underlined.

1.3B

Method	FineWeb-Edu	Wikipedia	C4	Common Crawl	Pile	Arxiv	Avg
MHA	10.462	16.754	17.851	16.419	13.262	14.039	14.798
MQA	10.592	17.039	18.113	16.580	12.701	14.161	14.864
GQA	10.456	16.857	17.889	16.405	13.197	13.918	14.787
MLA	10.449	**15.881**	17.758	16.143	12.240	13.657	**14.355**
TPA	**10.365**	16.505	17.761	16.156	12.575	13.875	14.539
GLA	10.395	16.548	**17.725**	**16.057**	12.463	**13.595**	14.464
MLRA	10.402	16.170	17.787	16.186	**12.234**	13.705	14.414

2.9B

Method	FineWeb-Edu	Wikipedia	C4	Common Crawl	Pile	Arxiv	Avg
MHA	9.368	14.814	16.372	14.864	11.941	12.600	13.326
MQA	9.499	15.232	16.664	15.173	11.886	12.972	13.571
GQA	9.356	15.089	16.466	14.937	11.858	12.690	13.399
MLA	9.280	14.901	**16.278**	**14.673**	11.190	**12.370**	13.115
TPA	9.235	15.088	16.355	14.804	11.702	12.574	13.293
GLA	9.298	14.368	16.381	14.792	11.242	12.417	13.083
MLRA	**9.224**	**14.214**	16.287	14.702	**11.147**	12.412	**12.998**

We can run the base path on a high-end GPU (e.g., H100) and distribute the low-rank path across multiple lower-cost GPUs (e.g., RTX 4090). Their reduced bandwidth and compute capabilities are sufficient given the low-rank path's low arithmetic intensity and the fact that its KV cache can be sharded, provided that the latency of the low-rank path does not exceed that of the base path.

4 Experiments

4.1 Experimental Setup

We train models at three scales: 354M, 1311M (1.3B), and 2873M (2.9B), on FineWeb-Edu-100B (Penedo et al., 2024). We report main results at the 1.3B and 2.9B scales, and use the 354M model for initialization ablations (Appendix F.2). Each model is pretrained from scratch on 98.3B tokens with an additional 0.1B tokens for validation. We adopt the LLaMA-3 architecture (Appendix E) and use six attention mechanisms as baselines: MHA (Vaswani et al., 2017), MQA (Shazeer, 2019), GQA (Ainslie et al., 2023), MLA (DeepSeek et al., 2024c), TPA (Zhang et al., 2025), and GLA (Zadouri et al., 2025). Architectural hyperparameters and training setup follow the GPT-3 configuration (Appendices F.1 and F.2). All models are trained on 8 NVIDIA H100 80G GPUs. Figure 3 shows the training loss curves for all attention mechanisms at the 1.3B and 2.9B scales.

4.2 Experimental Results

4.2.1 Model Quality

Validation Perplexity. We report validation perplexity for all attention mechanisms across six datasets (FineWeb-Edu (Penedo et al., 2024), Wikipedia, C4, Common Crawl, Pile (Gao et al., 2020), and Arxiv), using 100M tokens per dataset. As Table 3 shows, MLRA achieves the lowest average perplexity at the 2.9B (12.998) scale across six datasets. It achieves the best results on FineWeb-Edu, Wikipedia, and Pile, while placing second on C4, Common Crawl, and Arxiv. At the 1.3B scale, MLRA remains highly competitive (second-best average) and is best on Pile.

Downstream Evaluation. We evaluate zero-shot performance on standard benchmarks, including ARC-Easy (ARC-E) (Yadav et al., 2019), ARC-Challenge (ARC-C), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), PIQA (Bisk et al., 2020), and MMLU (Hendrycks et al., 2021), using the `lm-evaluation-harness` (Gao et al., 2024) package. We report normalized accuracy for ARC-Easy, ARC-Challenge, HellaSwag, and PIQA, and standard accuracy for the remaining tasks. As Table 4 shows, MLRA achieves the highest zero-shot average accuracy at both scales:

Table 4: Downstream evaluation at the 1.3B and 2.9B scales on six datasets: ARC-Easy, ARC-Challenge, BoolQ, HellaSwag, PIQA, and MMLU. Best is **bold**; second best is underlined.

1.3B							
Method	**ARC-E**	**ARC-C**	**BoolQ**	**HellaSwag**	**PIQA**	**MMLU**	**Avg**
MHA	64.69	38.23	**63.39**	57.96	73.50	25.06	53.80
MQA	63.68	37.63	61.50	57.31	73.99	23.32	52.91
GQA	65.57	37.88	59.36	57.89	73.61	23.39	52.95
MLA	64.44	36.52	58.59	57.60	73.61	24.57	52.55
TPA	64.69	**38.48**	61.56	**58.22**	**74.70**	25.37	53.84
GLA	65.53	36.86	63.33	57.93	73.99	**25.69**	53.89
MLRA	**66.08**	37.29	63.09	57.76	74.27	25.26	**53.96**

2.9B							
Method	**ARC-E**	**ARC-C**	**BoolQ**	**HellaSwag**	**PIQA**	**MMLU**	**Avg**
MHA	68.14	41.13	63.39	61.97	75.30	25.79	55.95
MQA	69.49	40.27	63.12	61.11	75.14	23.69	55.47
GQA	69.74	40.36	62.05	61.83	75.52	24.58	55.68
MLA	70.58	41.04	59.24	62.20	75.03	24.65	55.46
TPA	70.12	40.70	**64.50**	62.27	**76.33**	**27.04**	56.83
GLA	69.65	40.96	63.55	**62.44**	75.79	25.35	56.29
MLRA	**70.88**	**43.77**	63.33	62.31	75.63	26.43	**57.06**

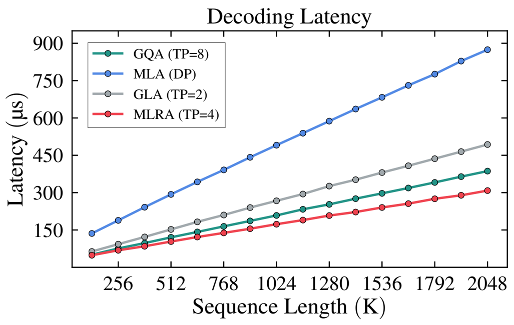

Figure 4: Decoding latency (lower is better) versus sequence length (batch=1) for GQA, MLA, GLA, and MLRA.

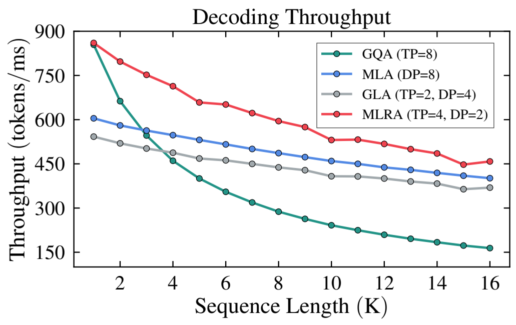

Figure 5: Decoding throughput versus sequence length (batch=128) for GQA, MLA, GLA, and MLRA.

53.96% at 1.3B and 57.06% at 2.9B. At the 2.9B scale, it leads ARC-Easy and ARC-Challenge and places second on HellaSwag and MMLU. At the 1.3B scale, it leads ARC-Easy and places second on PIQA.

4.2.2 Efficiency

Decoding Speed. We benchmark decoding speed for GQA, MLA, GLA, and MLRA in long-context settings. All models use 64 heads with a head dimension of 128; for MLA, GLA, and MLRA, the partial RoPE dimension is 64. MLA is evaluated using DeepSeek's official implementation FlashMLA (Jiashi Li, 2025). GQA and GLA use FlashAttention kernels (Dao et al., 2022; Dao, 2024; Shah et al., 2024). We implement our MLRA kernel based on FlashAttention. We evaluate decoding speed across sequence lengths from 131,072 to 2,097,152 tokens (128K-2M). As shown in Figure 4, MLRA consistently outperforms all baselines at every length, yielding 1.05×-1.26× speedups over GQA. The gap grows with context length against GQA and GLA, while the speedup over MLA remains steady at about 2.8×, indicating that MLRA with TP=4 substantially reduces long-context decoding latency.

Decoding Throughput. We evaluate decoding throughput for GQA, MLA, GLA, and MLRA on eight NVIDIA H100 80G GPUs, fixing the number of attention heads to 128 and the hidden size to 7168, following DeepSeekV3 (DeepSeek et al., 2024a). We set $g = 16$ for GQA. For MLA decoding deployment, there is a trade-off between data parallelism (DP) and tensor parallelism. With DP, we assign different requests to different devices, so attention parameters are replicated across devices and load can become imbalanced due to varying sequence lengths. With TP, the up-projection parameters are sharded by head, but the KV cache is replicated across devices. Following SGLang (Zheng et al., 2024), we otherwise strive to avoid KV cache duplication. Therefore, we use DP=8 for MLA, TP=2/DP=4 for GLA, TP=4/DP=2 for MLRA, and TP=8 for GQA. Throughput is reported for sequence lengths ranging from 1,024 to 16,384 tokens, and our end-to-end measurements include both the pre-attention stage that prepares inputs for the attention kernel and the attention computation itself. We accelerate pre-attention with torch.compile (Paszke et al., 2019) for MLA, GLA, and MLRA, and with custom Triton kernels for GQA. As shown in Figure 5, MLRA achieves the highest decoding throughput across both short and long sequence lengths. This suggests that MLRA with TP=4/DP=2 reduces parameter redundancy relative to MLA's DP=8, while introducing only modest partial RoPE duplication, thereby yielding higher throughput than MLA. For short sequences, GQA outperforms MLA and GLA because pre-attention dominates latency. However, MLRA remains competitive with GQA due to having even fewer query, key, and value parameters, as shown in Appendix F.1.

5 Related Work

KV Cache Compression. Recent works (Liu et al., 2023; Anagnostidis et al., 2023; Zhang et al., 2023b; Ge et al., 2024; Xiao et al., 2024; Kim et al., 2024; Zhang et al., 2024; Nawrot et al., 2024; Tang et al., 2024; Liu et al., 2024b; Dong et al., 2024; Yue et al., 2024; Cai et al., 2024; Liu et al., 2024a; Hooper et al., 2024; Sun et al., 2024; Chen et al., 2024a; Jiang et al., 2024a; Li et al., 2024; Xiao et al., 2025; Sun et al., 2025a; Meng et al., 2025; Tang et al., 2025) don't introduce new attention mechanisms; instead, they compress the KV cache for the pretrained models. Some of these works (Liu et al., 2024b; Hooper et al., 2024; Yue et al., 2024) use quantization to store the KV cache in low-bit formats. Some other approaches (Zhang et al., 2023b; Xiao et al., 2024; Li et al., 2024; Xiao et al., 2025) retain important tokens and discard others to compress the KV cache.

Low-Rank Approximation. Low-rank approximation approaches (Hu et al., 2022; Malladi et al., 2023; Zhang et al., 2023a; Dettmers et al., 2023; Lialin et al., 2024; Zhu et al., 2024; Zeng & Lee, 2024; Chen et al., 2024b; Zhang, 2024; Liang & Li, 2024; Zhao et al., 2024a; Shi et al., 2024; Jiang et al., 2024b; Lin et al., 2025; Wang et al., 2025; Chang et al., 2025; Li et al., 2025) are widely used to compress representations to a low-dimensional latent, then up-project to recover full representations. These methods greatly reduce trainable parameters (Hu et al., 2022; Dettmers et al., 2023) during fine-tuning and decrease the number of parameters (Lin et al., 2025; Wang et al., 2025) for pretrained models.

System for Attention. FlashAttention (Dao et al., 2022; Dao, 2024; Shah et al., 2024) uses tiling and online softmax to minimize reads and writes between high-bandwidth memory and on-chip SRAM, shifting attention from a memory bottleneck to a compute bottleneck. FlashMLA (Jiashi Li, 2025) avoids explicit KV materialization during decoding by absorbing the key and value up-projection matrices into the query and attention output. The following attention computation is similar to MQA with shared KV states. Inspired by classical virtual memory and paging in operating systems, PagedAttention (Kwon et al., 2023) and vLLM use block-level memory management and preemptive request scheduling to reduce fragmentation and redundant duplication.

6 Conclusion

In this work, we propose Multi-Head Low-Rank Attention (MLRA), a TP-friendly dual-path attention mechanism. In the base path, we compress the KV cache into a single latent head and up-project it to produce multiple keys and values. In the low-rank path, we compress the KV cache into several tiny latent heads, each up-projected to generate its key and value. Our MLRA supports TP, reducing the per-device KV cache to $1.5d_h$, which is lower than that of GQA. At the 2.9 B scale, MLRA achieves state-of-the-art model quality across our benchmarks, including perplexity and downstream evaluations. We implement MLRA kernel based on FlashAttention. For long-context decoding (up to 2M tokens), MLRA achieves the lowest latency with 4-way TP and delivers the highest decoding throughput across sequence lengths from 1K to 16K tokens. Future work is to train MLRA at larger scales, such as 70B, to validate its effectiveness.

Ethics Statement

This work proposes an efficiency-oriented attention mechanism (MLRA) and does not involve human subjects or the collection of new datasets. Experiments use established corpora and benchmarks under their licenses.

Reproducibility Statement

Our implementation builds on public GitHub codebases, and we include the modeling code in the supplementary materials. Due to space constraints and the anonymity policy, we cannot include all code and datasets there. Upon acceptance, we will release the full code and datasets in a public GitHub repository.

References

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Empirical Methods in Natural Language Processing*, 2023.

Sotiris Anagnostidis, Dario Pavllo, Luca Biggio, Lorenzo Noci, Aurelien Lucchi, and Thomas Hofmann. Dynamic context pruning for efficient and interpretable autoregressive transformers. In *Advances in Neural Information Processing Systems*, 2023.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. Piqa: Reasoning about physical commonsense in natural language. In *AAAI Conference on Artificial Intelligence*, 2020.

Ruisi Cai, Yuandong Tian, Zhangyang Wang, and Beidi Chen. Lococo: Dropping in convolutions for long context compression. In *International Conference on Machine Learning*, 2024.

Chi-Chih Chang, Wei-Cheng Lin, Chien-Yu Lin, Chong-Yan Chen, Yu-Fang Hu, Pei-Shuo Wang, Ning-Chi Huang, Luis Ceze, Mohamed S. Abdelfattah, and Kai-Chiang Wu. Palu: KV-cache compression with low-rank projection. In *International Conference on Learning Representations*, 2025.

Renze Chen, Zhuofeng Wang, Beiquan Cao, Tong Wu, Size Zheng, Xiuhong Li, Xuechao Wei, Shengen Yan, Meng Li, and Yun Liang. Arkvale: Efficient generative LLM inference with recallable key-value eviction. In *Advances in Neural Information Processing Systems*, 2024a.

Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. LongloRA: Efficient fine-tuning of long-context large language models. In *International Conference on Learning Representations*, 2024b.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *North American Association for Computational Linguistics*, 2019.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations*, 2024.

Tri Dao, Daniel Y Fu, Stefano Ermon, Atri Rudra, and Christopher Re. Flashattention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, 2022.

DeepSeek et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024a.

DeepSeek et al. Deepseek-v3 technical report. https://github.com/deepseek-ai/DeepSeek-V3, 2024b.

DeepSeek et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024c.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems*, 2023.

Harry Dong, Xinyu Yang, Zhenyu Zhang, Zhangyang Wang, Yuejie Chi, and Beidi Chen. Get more with less: Synthesizing recurrence with kv cache compression for efficient llm inference. In *International Conference on Machine Learning*, 2024.

Fabian Fuchs, Daniel Worrall, Volker Fischer, and Max Welling. Se (3)-transformers: 3d roto-translation equivariant attention networks. In *Advances in Neural Information Processing Systems*, 2020.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 2024.

Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *International Conference on Learning Representations*, 2024.

Amir Gholami, Zhewei Yao, Sehoon Kim, Coleman Hooper, Michael W Mahoney, and Kurt Keutzer. Ai and memory wall. *IEEE Micro*, 2024.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

Tyler Griggs, Xiaoxuan Liu, Jiaxiang Yu, Doyoung Kim, Wei-Lin Chiang, Alvin Cheung, and Ion Stoica. Mélange: Cost efficient large language model serving by exploiting gpu heterogeneity. *arXiv preprint arXiv:2404.14527*, 2024.

Jiaao He and Jidong Zhai. Fastdecode: High-throughput gpu-efficient llm serving using heterogeneous pipelines. *arXiv preprint arXiv:2403.11421*, 2024.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Coleman Richard Charles Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W. Mahoney, Sophia Shao, Kurt Keutzer, and Amir Gholami. KVQuant: Towards 10 million context length LLM inference with KV cache quantization. In *Advances in Neural Information Processing Systems*, 2024.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Jingcheng Hu, Houyi Li, Yinmin Zhang, Zili Wang, Shuigeng Zhou, Xiangyu Zhang, Heung-Yeung Shum, and Daxin Jiang. Multi-matrix factorization attention. *arXiv preprint arXiv:2412.19255*, 2024.

Andrei Ivanov, Nikoli Dryden, Tal Ben-Nun, Shigang Li, and Torsten Hoefler. Data movement is all you need: A case study on optimizing transformers. In *Machine Learning and Systems*, 2021.

Huiqiang Jiang, Yucheng Li, Chengruidong Zhang, Qianhui Wu, Xufang Luo, Surin Ahn, Zhenhua Han, Amir H. Abdi, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. MInference 1.0: Accelerating pre-filling for long-context LLMs via dynamic sparse attention. In *Advances in Neural Information Processing Systems*, 2024a.

Ting Jiang, Shaohan Huang, Shengyue Luo, Zihan Zhang, Haizhen Huang, Furu Wei, Weiwei Deng, Feng Sun, Qi Zhang, Deqing Wang, et al. Mora: High-rank updating for parameter-efficient fine-tuning. *arXiv preprint arXiv:2405.12130*, 2024b.

Youhe Jiang, Ran Yan, Xiaozhe Yao, Yang Zhou, Beidi Chen, and Binhang Yuan. Hexgen: Generative inference of large language model over heterogeneous environment. In *International Conference on Machine Learning*, 2024c.

Youhe Jiang, Ran Yan, and Binhang Yuan. Hexgen-2: Disaggregated generative inference of LLMs in heterogeneous environment. In *International Conference on Learning Representations*, 2025.

Shengyu Liu Jiashi Li. Flashmla: Efficient mla decoding kernels. https://github.com/deepseek-ai/FlashMLA, 2025.

Jang-Hyun Kim, Junyoung Yeom, Sangdoo Yun, and Hyun Oh Song. Compressed context memory for online language model interaction. In *International Conference on Learning Representations*, 2024.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Symposium on Operating Systems Principles*, 2023.

Patrick Lewis, Ethan Perez, Aleksandara Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Kuttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, 2020.

Xinlong Li, Weijieying Ren, Wei Qin, Lei Wang, Tianxiang Zhao, and Richang Hong. Analyzing and reducing catastrophic forgetting in parameter efficient tuning. In *International Conference on Acoustics, Speech and Signal Processing*, 2025.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. In *Advances in Neural Information Processing Systems*, 2024.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. ReloRA: High-rank training through low-rank updates. In *International Conference on Learning Representations*, 2024.

Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Chi-Heng Lin, Shangqian Gao, James Seale Smith, Abhishek Patel, Shikhar Tuli, Yilin Shen, Hongxia Jin, and Yen-Chang Hsu. ModeGPT: Modular decomposition for large language model compression. In *International Conference on Learning Representations*, 2025.

Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, Michael Maire, Henry Hoffmann, Ari Holtzman, and Junchen Jiang. Cachegen: Kv cache compression and streaming for fast large language model serving. In *Conference of the ACM Special Interest Group on Data Communication*, 2024a.

Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for LLM KV cache compression at test time. In *Advances in Neural Information Processing Systems*, 2023.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. KIVI: A tuning-free asymmetric 2bit quantization for KV cache. In *International Conference on Machine Learning*, 2024b.

Llama et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Sadhika Malladi, Alexander Wettig, Dingli Yu, Danqi Chen, and Sanjeev Arora. A kernel-based view of language model fine-tuning. In *International Conference on Machine Learning*, 2023.

Fanxu Meng, Pingzhi Tang, Xiaojuan Tang, Zengwei Yao, Xing Sun, and Muhan Zhang. Transmla: Multi-head latent attention is all you need. *arXiv preprint arXiv:2502.07864*, 2025.

Piotr Nawrot, Adrian Łańcucki, Marcin Chochowski, David Tarjan, and Edoardo Ponti. Dynamic memory compression: Retrofitting LLMs for accelerated inference. In *International Conference on Machine Learning*, 2024.

Hiroyuki Ootomo and Rio Yokota. Reducing shared memory footprint to leverage high throughput on tensor cores and its flexible api extension library. In *International Conference on High Performance Computing in Asia-Pacific Region*, 2023.

OpenAI et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In *Machine Learning and Systems*, 2023.

Ranajoy Sadhukhan, Jian Chen, Zhuoming Chen, Vashisth Tiwari, Ruihang Lai, Jinyuan Shi, Ian En-Hsu Yen, Avner May, Tianqi Chen, and Beidi Chen. Magicdec: Breaking the latency-throughput tradeoff for long context generation with speculative decoding. In *International Conference on Learning Representations*, 2025.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International Conference on Machine Learning*, 2021.

Jay Shah, Ganesh Bikshandi, Ying Zhang, Vijay Thakkar, Pradeep Ramani, and Tri Dao. Flashattention-3: Fast and accurate attention with asynchrony and low-precision. In *Advances in Neural Information Processing Systems*, 2024.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Yiming Shi, Jiwei Wei, Yujia Wu, Ran Ran, Chengwei Sun, Shiyuan He, and Yang Yang. Loldu: Low-rank adaptation via lower-diag-upper decomposition for parameter-efficient fine-tuning. *arXiv preprint arXiv:2410.13618*, 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.

Hanshi Sun, Li-Wen Chang, Wenlei Bao, Size Zheng, Ningxin Zheng, Xin Liu, Harry Dong, Yuejie Chi, and Beidi Chen. ShadowKV: KV cache in shadows for high-throughput long-context LLM inference. In *International Conference on Machine Learning*, 2025a.

Luoyang Sun, Cheng Deng, Jiwen Jiang, Xinjian Wu, Haifeng Zhang, Lei Chen, Lionel Ni, and Jun Wang. Gta: Grouped-head latent attention. *arXiv preprint arXiv:2506.17286*, 2025b.

Yutao Sun, Li Dong, Yi Zhu, Shaohan Huang, Wenhui Wang, Shuming Ma, Quanlu Zhang, Jianyong Wang, and Furu Wei. You only cache once: Decoder-decoder architectures for language models. In *Advances in Neural Information Processing Systems*, 2024.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: Query-aware sparsity for efficient long-context LLM inference. In *International Conference on Machine Learning*, 2024.

Xiaojuan Tang, Fanxu Meng, Pingzhi Tang, Yuxuan Wang, Di Yin, Xing Sun, and Muhan Zhang. Tpla: Tensor parallel latent attention for efficient disaggregated prefill & decode inference. *arXiv preprint arXiv:2508.15881*, 2025.

Nathaniel Thomas, Tess Smidt, Steven Kearnes, Lusann Yang, Li Li, Kai Kohlhoff, and Patrick Riley. Tensor field networks: Rotation-and translation-equivariant neural networks for 3d point clouds. *arXiv preprint arXiv:1802.08219*, 2018.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations*, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, 2022.

Maurice Weiler, Mario Geiger, Max Welling, Wouter Boomsma, and Taco S Cohen. 3d steerable cnns: Learning rotationally equivariant features in volumetric data. In *Advances in Neural Information Processing Systems*, 2018.

Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 2009.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *International Conference on Learning Representations*, 2024.

Guangxuan Xiao, Jiaming Tang, Jingwei Zuo, junxian guo, Shang Yang, Haotian Tang, Yao Fu, and Song Han. Duoattention: Efficient long-context LLM inference with retrieval and streaming heads. In *International Conference on Learning Representations*, 2025.

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. In *Empirical Methods in Natural Language Processing*, 2019.

Yuxuan Yue, Zhihang Yuan, Haojie Duanmu, Sifan Zhou, Jianlong Wu, and Liqiang Nie. Wkvquant: Quantizing weight and key/value cache for large language models gains more. *arXiv preprint arXiv:2402.12065*, 2024.

Ted Zadouri, Hubert Strauss, and Tri Dao. Hardware-efficient attention for fast decoding. In *Conference on Language Modeling*, 2025.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In *Association for Computational Linguistics*, 2019.

Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. In *International Conference on Learning Representations*, 2024.

Hengyu Zhang. Sinklora: Enhanced efficiency and chat capabilities for long-context large language models. *arXiv preprint arXiv:2406.05678*, 2024.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *International Conference on Learning Representations*, 2023a.

Yifan Zhang, Yifeng Liu, Huizhuo Yuan, Zhen Qin, Yang Yuan, Quanquan Gu, and Andrew Chi-Chih Yao. Tensor product attention is all you need. *arXiv preprint arXiv:2501.06425*, 2025.

Yuxin Zhang, Yuxuan Du, Gen Luo, Yunshan Zhong, Zhenyu Zhang, Shiwei Liu, and Rongrong Ji. Cam: Cache merging for memory-efficient LLMs inference. In *International Conference on Machine Learning*, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Re, Clark Barrett, Zhangyang Wang, and Beidi Chen. H2o: Heavy-hitter oracle for efficient generative inference of large language models. In *Advances in Neural Information Processing Systems*, 2023b.

Hongbo Zhao, Bolin Ni, Junsong Fan, Yuxi Wang, Yuntao Chen, Gaofeng Meng, and Zhaoxiang Zhang. Continual forgetting for pre-trained vision models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024a.

Juntao Zhao, Borui Wan, Chuan Wu, Yanghua Peng, and Haibin Lin. Poster: Llm-pq: Serving llm on heterogeneous clusters with phase-aware partition and adaptive quantization. In *Symposium on Principles and Practice of Parallel Programming*, 2024b.

Chuanyang Zheng, Jiankai Sun, Yihang Gao, Yuehao Wang, Peihao Wang, Jing Xiong, Liliang Ren, Hao Cheng, Janardhan Kulkarni, Yelong Shen, et al. Sas: Simulated attention score. *arXiv preprint arXiv:2507.07694*, 2025.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systemss*, 2024.

Jiacheng Zhu, Kristjan Greenewald, Kimia Nadjahi, Haitz Sáez De Ocáriz Borde, Rickard Brüel Gabrielsson, Leshem Choshen, Marzyeh Ghassemi, Mikhail Yurochkin, and Justin Solomon. Asymmetry in low-rank adapters of foundation models. In *International Conference on Machine Learning*, 2024.

Appendix

A Theorem

Theorem 1. *Given two tokens with query* $\mathbf{q}$ *and key* $\mathbf{k}$ *at positions* m *and* n*, respectively, and applying RoPE to obtain* $\mathrm{RoPE}\left(\mathbf{q}, m\right)$ *and* $\mathrm{RoPE}\left(\mathbf{k}, n\right)$*, we aim to prove that RoPE is translation equivariant with respect to the inner product. In the definition of translation equivariance (Eq.(1)), this corresponds to setting the output transformation* S_s *as the identity map, i.e.,* $S_s = \mathbf{I}$*. Specifically, we show that translating both positions by an offset* s *leaves the inner product unchanged:*

$$\left\langle \mathrm{RoPE}\left(\mathbf{q}, m+s\right), \mathrm{RoPE}\left(\mathbf{k}, n+s\right)\right\rangle = \left\langle \mathrm{RoPE}\left(\mathbf{q}, m\right), \mathrm{RoPE}\left(\mathbf{k}, n\right)\right\rangle.$$

Proof. We compute the inner product under RoPE as:

$$\left(\mathbf{q}\mathbf{R}_m\right)\left(\mathbf{k}\mathbf{R}_n\right)^\top = \mathbf{q}\mathbf{R}_m\mathbf{R}_n^\top\mathbf{k}^\top = \mathrm{Re}\left[\sum_{i=1}^{d/2} \mathbf{q}_{[2i-1:2i+1]}\mathbf{k}^*_{[2i-1:2i+1]} e^{i(m-n)\theta_i}\right],$$

where $\mathbf{R}_m$ is the rotary matrix at position m. Now consider translating both tokens by an offset s. The difference in positions becomes:

$$(m+s)-(n+s) = m-n,$$

which leaves the term $e^{i(m-n)\theta_i}$ unchanged. Therefore, the inner product remains invariant under a translation of both tokens, and we have:

$$\phi(T_s(\mathbf{x})) = \phi(\mathbf{x}), \quad \text{with } S_s = \mathbf{I},$$

which proves that RoPE is translation equivariant with respect to the inner product. $\square$

B Query Computation in MLRA

The hidden states $\mathbf{H} \in \mathbb{R}^{n\times d}$ are first passed through a linear down-projection $\mathbf{A}_{\mathrm{Q}} \in \mathbb{R}^{d\times\left(d'_u+hr'\right)}$ yielding two parts

$$\left[\mathbf{U}_{\mathrm{Q}},\ \overline{\mathbf{C}}_{\mathrm{Q}}\right] = \mathbf{H}\mathbf{A}_{\mathrm{Q}} \in \mathbb{R}^{n\times\left(d'_u+hr'\right)}.$$

By default, we set $d'_u = 2d_h$ and $r' = \frac{6d_h}{h}$. The base latent head $\mathbf{U}_{\mathrm{Q}} \in \mathbb{R}^{n\times d'_u}$ is up-projected by a weight matrix $\mathbf{B}^{\mathrm{Q}}_{\text{base}} \in \mathbb{R}^{d'_u\times h\left(d_h+d_h^R\right)}$ producing

$$\overline{\mathbf{Q}}_{\text{base}} = \mathbf{U}_{\mathrm{Q}}\,\mathbf{B}^{\mathrm{Q}}_{\text{base}} \in \mathbb{R}^{n\times h\left(d_h+d_h^R\right)},$$

which reshapes and splits into

$$\mathbf{Q}_{\text{base,NoPE}} \in \mathbb{R}^{n\times h\times d_h}, \qquad \mathbf{Q}_{\text{base,RoPE}} \in \mathbb{R}^{n\times h\times d_h^R}.$$

$\overline{\mathbf{C}}_{\mathrm{Q}}$ is reshaped to expose the head dimension to obtain h latent heads,

$$\mathbf{C}_{\mathrm{Q}} = \mathrm{reshape}\left(\overline{\mathbf{C}}_{\mathrm{Q}}, [n,\ h,\ r']\right) \in \mathbb{R}^{n\times h\times r'},$$

and up-projected head-wise by a weight matrix $\mathbf{B}^{\mathrm{Q}}_{\text{lora}} \in \mathbb{R}^{h\times r'\times\left(d_h+d_h^R\right)}$. This produces an intermediate query tensor

$$\mathbf{Q}_{\text{lora}} = \mathrm{einsum}\left(\texttt{"nhr,hrd} \rightarrow \texttt{nhd"}, \mathbf{C}_{\mathrm{Q}},\ \mathbf{B}^{\mathrm{Q}}_{\text{lora}}\right) \in \mathbb{R}^{n\times h\times\left(d_h+d_h^R\right)},$$

which is split into

$$\mathbf{Q}_{\text{lora,NoPE}} \in \mathbb{R}^{n\times h\times d_h}, \qquad \mathbf{Q}_{\text{lora,RoPE}} \in \mathbb{R}^{n\times h\times d_h^R}.$$

The low-rank and base paths are fused:

$$\mathbf{Q}_{\text{NoPE}} = \gamma\mathbf{Q}_{\text{lora,NoPE}} + \mathbf{Q}_{\text{base,NoPE}}, \qquad \widetilde{\mathbf{Q}}_{\text{RoPE}} = \gamma\mathbf{Q}_{\text{lora,RoPE}} + \mathbf{Q}_{\text{base,RoPE}}.$$

RoPE is applied to $\widetilde{\mathbf{Q}}_{\text{RoPE}}$ to obtain $\mathbf{Q}_{\text{RoPE}}$. This result is then concatenated with the other query components to form the final query:

$$\mathbf{Q} = \mathrm{Concat}\left[\mathbf{Q}_{\text{NoPE}},\ \mathbf{Q}_{\text{RoPE}}\right] \in \mathbb{R}^{n\times h\times\left(d_h+d_h^R\right)}.$$

C Attention Mechanism

C.1 Multi-Head Attention (MHA)

Consider a sequence of n tokens with hidden states $\mathbf{H} \in \mathbb{R}^{n\times d}$. We first project these hidden states into query, key, and value representations using projection matrices $\mathbf{W}_{\mathrm{Q}}, \mathbf{W}_{\mathrm{K}}, \mathbf{W}_{\mathrm{V}} \in \mathbb{R}^{d\times(hd_h)}$:

$$\overline{\mathbf{Q}} = \mathrm{RoPE}\left(\mathbf{H}\mathbf{W}_{\mathrm{Q}}\right), \quad \overline{\mathbf{K}} = \mathrm{RoPE}\left(\mathbf{H}\mathbf{W}_{\mathrm{K}}\right), \quad \overline{\mathbf{V}} = \mathbf{H}\mathbf{W}_{\mathrm{V}},$$

where $\overline{\mathbf{Q}}, \overline{\mathbf{K}}, \overline{\mathbf{V}} \in \mathbb{R}^{n\times(hd_h)}$, h is the number of attention heads, and d_h is the dimensionality of each head. Next, we reshape these matrices to separate the head dimension:

$$\mathbf{Q} = \mathrm{reshape}(\overline{\mathbf{Q}}, [n, h, d_h]), \quad \mathbf{K} = \mathrm{reshape}(\overline{\mathbf{K}}, [n, h, d_h]), \quad \mathbf{V} = \mathrm{reshape}(\overline{\mathbf{V}}, [n, h, d_h]),$$

such that $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{n\times h\times d_h}$. In MHA, the keys $\mathbf{K}$ and values $\mathbf{V}$ from previous tokens are cached to accelerate inference.

Remark 2. *Let $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n\times h\times d_h}$ denote the RoPE-encoded queries and keys after projection and reshaping, and define per-head query and key vectors as:*

$$\mathbf{q}_m^{(i)} := \mathbf{Q}[m, i, :], \quad \mathbf{k}_n^{(i)} := \mathbf{K}[n, i, :],$$

where m and n are token positions and i is the attention head index. Then, for any translation t, it follows from Theorem 1 that:

$$\left\langle \mathbf{q}_{m+t}^{(i)},\ \mathbf{k}_{n+t}^{(i)} \right\rangle = \left\langle \mathbf{q}_{m}^{(i)},\ \mathbf{k}_{n}^{(i)} \right\rangle.$$

C.2 Multi-Query Attention (MQA) and Grouped-Query Attention (GQA)

Both MQA and GQA reduce the number of key and value heads compared to MHA, while maintaining the full number of query heads. MQA takes this to the extreme by using a single key-value head for all query heads, whereas GQA partitions the query heads into groups that each share a key-value head. Given a sequence of n tokens with hidden states $\mathbf{H} \in \mathbb{R}^{n\times d}$, the queries are computed using the same projection as in MHA:

$$\overline{\mathbf{Q}} = \mathrm{RoPE}\left(\mathbf{H}\mathbf{W}_{\mathrm{Q}}\right), \quad \overline{\mathbf{Q}} \in \mathbb{R}^{n\times(hd_h)}.$$

To reduce KV cache footprint, we use projection matrices $\mathbf{W}_{\mathrm{K}}, \mathbf{W}_{\mathrm{V}} \in \mathbb{R}^{d\times d_h g}$, where $g < h$ (e.g., $g = 1$ for MQA), to compute:

$$\overline{\mathbf{K}}_{\mathrm{C}} = \mathrm{RoPE}\left(\mathbf{H}\mathbf{W}_{\mathrm{K}}\right), \quad \overline{\mathbf{V}}_{\mathrm{C}} = \mathbf{H}\mathbf{W}_{\mathrm{V}}, \quad \overline{\mathbf{K}}_{\mathrm{C}}, \overline{\mathbf{V}}_{\mathrm{C}} \in \mathbb{R}^{n\times d_h g}.$$

These are then reshaped into per-head form:

$$\mathbf{K}_{\mathrm{C}} = \mathrm{reshape}(\overline{\mathbf{K}}_{\mathrm{C}},\ [n, g, d_h]), \quad \mathbf{V}_{\mathrm{C}} = \mathrm{reshape}(\overline{\mathbf{V}}_{\mathrm{C}},\ [n, g, d_h]).$$

We cache $\overline{\mathbf{K}}_{\mathrm{C}}$ and $\overline{\mathbf{V}}_{\mathrm{C}}$ during inference. During decoding, we first repeat them along the head axis to match the h query heads (repeat factor $r = h/g$):

$$\mathbf{K} = \mathrm{RepeatInterleave}\left(\mathbf{K}_{\mathrm{C}},\ r,\ \mathrm{dim} = 1\right) \in \mathbb{R}^{n\times h\times d_h}, \quad \mathbf{V} = \mathrm{RepeatInterleave}\left(\mathbf{V}_{\mathrm{C}},\ r,\ \mathrm{dim} = 1\right).$$

Remark 3. *Let $\mathbf{Q} \in \mathbb{R}^{n\times h\times d_h}$ and $\mathbf{K}_C \in \mathbb{R}^{n\times g\times d_h}$ denote the RoPE-encoded queries and keys, respectively. For attention head i, we define the query and key vectors at positions m and n as:*

$$\mathbf{q}_m^{(i)} := \mathbf{Q}[m, i, :], \quad \mathbf{k}_n^{(i)} := \mathbf{K}_C[n, j, :] \textit{ for some } j \in \{1, \ldots, g\}.$$

Since each $\mathbf{k}_n^{(j)}$ is obtained via RoPE, and RoPE preserves inner products under joint position translations, we have, for any offset t:

$$\left\langle \mathbf{q}_{m+t}^{(i)},\ \mathbf{k}_{n+t}^{(i)} \right\rangle = \left\langle \mathbf{q}_{m}^{(i)},\ \mathbf{k}_{n}^{(i)} \right\rangle,$$

where $\mathbf{k}_n^{(i)} = \mathbf{K}_C[n, j, :]$ for some $j \in \{1, \ldots, g\}$.

C.3 Multi-Head Latent Attention (MLA)

Given a sequence of n tokens with hidden states $\mathbf{H} \in \mathbb{R}^{n \times d}$, MLA first computes queries as:

$$\mathbf{C}_{\text{Q}} = \mathbf{H}\mathbf{A}_{\text{Q}}, \quad \overline{\mathbf{Q}}_{\text{NoPE}} = \mathbf{C}_{\text{Q}}\mathbf{B}_{\text{Q}}, \quad \overline{\mathbf{Q}}_{\text{RoPE}} = \text{RoPE}\left(\mathbf{C}_{\text{Q}}\mathbf{B}_{\text{QR}}\right),$$
$$\mathbf{A}_{\text{Q}} \in \mathbb{R}^{d \times d'_c}, \quad \mathbf{B}_{\text{Q}} \in \mathbb{R}^{d'_c \times (hd_h)}, \quad \mathbf{B}_{\text{QR}} \in \mathbb{R}^{d'_c \times d_h^R h},$$

where $d'_c \ll hd_h$ is a low-rank latent dimension.

We then reshape the query to separate heads:

$$\mathbf{Q}_{\text{NoPE}} = \text{reshape}(\overline{\mathbf{Q}}_{\text{C}},\ [n,\ h,\ d_h]), \quad \mathbf{Q}_{\text{RoPE}} = \text{reshape}(\overline{\mathbf{Q}}_{\text{RoPE}},\ [n,\ h,\ d_h^R]),$$

where $\mathbf{Q}_{\text{NoPE}} \in \mathbb{R}^{n \times h \times d_h}$ and $\mathbf{Q}_{\text{RoPE}} \in \mathbb{R}^{n \times h \times d_h^R}$. These are concatenated along the last dimension to form the final query:

$$\mathbf{Q} = \text{Concat}\left[\mathbf{Q}_{\text{NoPE}},\ \mathbf{Q}_{\text{RoPE}}\right] \in \mathbb{R}^{n \times h \times (d_h + d_h^R)}.$$

To reduce the KV cache, MLA obtains shared compressed KV states via a down-projection:

$$\mathbf{C}_{\text{KV}} = \mathbf{H}\mathbf{A}_{\text{KV}}, \quad \mathbf{A}_{\text{KV}} \in \mathbb{R}^{d \times d_c}, \quad \mathbf{C}_{\text{KV}} \in \mathbb{R}^{n \times d_c},$$
$$\mathbf{K}_{\text{RoPE}} = \text{RoPE}\left(\mathbf{H}\mathbf{A}_{\text{KR}}\right), \quad \mathbf{A}_{\text{KR}} \in \mathbb{R}^{d \times d_h^R}, \quad \mathbf{K}_{\text{RoPE}} \in \mathbb{R}^{n \times d_h^R}.$$

where $d_c \ll hd_h$ is the latent dimension. Both $\mathbf{C}_{\text{KV}}$ and $\mathbf{K}_{\text{RoPE}}$ are stored in the KV cache during inference.

At runtime, MLA reconstructs the full keys and values using learned up-projection matrices:

$$\overline{\mathbf{K}}_{\text{NoPE}} = \mathbf{C}_{\text{KV}}\mathbf{B}_{\text{K}}, \quad \overline{\mathbf{V}} = \mathbf{C}_{\text{KV}}\mathbf{B}_{\text{V}}, \quad \mathbf{B}_{\text{K}}, \mathbf{B}_{\text{V}} \in \mathbb{R}^{d_c \times (hd_h)}.$$

These are reshaped into per-head form:

$$\mathbf{K}_{\text{NoPE}} = \text{reshape}(\overline{\mathbf{K}}_{\text{NoPE}},\ [n,\ h,\ d_h]), \quad \mathbf{V} = \text{reshape}(\overline{\mathbf{V}},\ [n,\ h,\ d_h]),$$

where $\mathbf{K}_{\text{NoPE}} \in \mathbb{R}^{n \times h \times d_h}$ and $\mathbf{V} \in \mathbb{R}^{n \times h \times d_h}$.

To obtain per-head position-aware keys, MLA repeats the position encoding across heads:

$$\mathbf{K} = \text{Concat}\left[\mathbf{K}_{\text{NoPE}},\ \text{repeat}\left(\mathbf{K}_{\text{RoPE}},\ h\right)\right].$$

The translation equivariance analysis is the same as in MLRA.

Decoding without KV materialization. Given a query $\mathbf{q} = \text{Concat}\left[\mathbf{q}_{\text{NoPE}}, \mathbf{q}_{\text{RoPE}}\right] \in \mathbb{R}^{h \times \left(d_h + d_h^R\right)}$ and KV cache: $\mathbf{C}_{\text{KV}} \in \mathbb{R}^{n \times d_c}, \mathbf{K}_{\text{RoPE}} \in \mathbb{R}^{n \times d_h^R}$, MLA avoids KV materialization and instead operates in the compressed space. Let $\mathbf{B}_{\text{K}}, \mathbf{B}_{\text{V}} \in \mathbb{R}^{d_c \times (hd_h)}$ be partitioned per head as $\mathbf{B}_{\text{K}} = \left[\mathbf{B}_{\text{K}}^{(1)} \cdots \mathbf{B}_{\text{K}}^{(h)}\right], \mathbf{B}_{\text{V}} = \left[\mathbf{B}_{\text{V}}^{(1)} \cdots \mathbf{B}_{\text{V}}^{(h)}\right]$ with $\mathbf{B}_{\text{K}}^{(i)}, \mathbf{B}_{\text{V}}^{(i)} \in \mathbb{R}^{d_c \times d_h}$. For head i we compute with scale $\tau = \frac{1}{\sqrt{d_h + d_h^R}}$:

$$\begin{aligned}
&\text{Softmax}\left(\tau\,\text{Concat}\left[\mathbf{q}_{\text{NoPE}}^{(i)}, \mathbf{q}_{\text{RoPE}}^{(i)}\right]\left(\text{Concat}\left[\mathbf{C}_{\text{KV}}\mathbf{B}_{\text{K}}^{(i)}, \mathbf{K}_{\text{RoPE}}\right]\right)^{\top}\right)\left(\mathbf{C}_{\text{KV}}\mathbf{B}_{\text{V}}^{(i)}\right), \\
=&\,\text{Softmax}\left(\tau\mathbf{q}_{\text{NoPE}}^{(i)}\left(\mathbf{C}_{\text{KV}}\mathbf{B}_{\text{K}}^{(i)}\right)^{\top} + \tau\mathbf{q}_{\text{RoPE}}^{(i)}\mathbf{K}_{\text{RoPE}}^{\top}\right)\left(\mathbf{C}_{\text{KV}}\mathbf{B}_{\text{V}}^{(i)}\right), \\
=&\,\text{Softmax}\left(\tau\mathbf{q}_{\text{NoPE}}^{(i)}\left(\mathbf{B}_{\text{K}}^{(i)}\right)^{\top}\mathbf{C}_{\text{KV}}^{\top} + \tau\mathbf{q}_{\text{RoPE}}^{(i)}\mathbf{K}_{\text{RoPE}}^{\top}\right)\left(\mathbf{C}_{\text{KV}}\mathbf{B}_{\text{V}}^{(i)}\right), \\
=&\,\text{Softmax}\left(\tau\,\underbrace{\mathbf{q}_{\text{NoPE}}^{(i)}\left(\mathbf{B}_{\text{K}}^{(i)}\right)^{\top}}_{\tilde{\mathbf{q}}^{(i)} \in \mathbb{R}^{d_c}}\mathbf{C}_{\text{KV}}^{\top} + \tau\mathbf{q}_{\text{RoPE}}^{(i)}\mathbf{K}_{\text{RoPE}}^{\top}\right)\mathbf{C}_{\text{KV}}\mathbf{B}_{\text{V}}^{(i)}.
\end{aligned}$$

We compute attention without materializing per-head $\mathbf{K}, \mathbf{V}$ by exploiting matrix multiplication associativity.

Step 1 (Keys/Logits): We absorb the key's up-projection matrix into the query, resulting in a score computation that mimics MQA with shared KV states. For head i,

$$\tilde{\mathbf{q}}^{(i)} = \mathbf{q}^{(i)}_{\text{NoPE}}\big(\mathbf{B}^{(i)}_{\text{K}}\big)^{\top}, \qquad \boldsymbol{\lambda}^{(i)} = \mathrm{Softmax}\left(\tau\tilde{\mathbf{q}}^{(i)}\mathbf{C}^{\top}_{\text{KV}} + \tau\mathbf{q}^{(i)}_{\text{RoPE}}\mathbf{K}^{\top}_{\text{RoPE}}\right).$$

This avoids forming $\mathbf{K}^{(i)}_{\text{NoPE}} = \mathbf{C}_{\text{KV}}\mathbf{B}^{(i)}_{\text{K}}$, and the RoPE keys $\mathbf{K}_{\text{RoPE}}$ are shared across heads (no head-wise repeat).

Step 2 (Values/Output): Aggregate with shared KV, then up-project once:

$$\tilde{\mathbf{z}}^{(i)} = \boldsymbol{\lambda}^{(i)}\,\mathbf{C}_{\text{KV}}, \qquad \mathbf{z}^{(i)} = \tilde{\mathbf{z}}^{(i)}\,\mathbf{B}^{(i)}_{\text{V}}.$$

By deferring the right multiplication by $\mathbf{B}^{(i)}_{\text{V}}$, we never materialize $\mathbf{V}^{(i)} = \mathbf{C}_{\text{KV}}\mathbf{B}^{(i)}_{\text{V}}$. Consequently, decoding maintains only $\mathbf{C}_{\text{KV}} \in \mathbb{R}^{n\times d_c}$ and $\mathbf{K}_{\text{RoPE}} \in \mathbb{R}^{n\times d_h^R}$ caches instead of per-head full $\mathbf{K}, \mathbf{V}$.

C.4 Grouped Latent Attention (GLA)

Given a sequence of n tokens with hidden states $\mathbf{H} \in \mathbb{R}^{n\times d}$, GLA divides the h attention heads into g groups (e.g., $g = 2$), where each group has $h_g = h/g$ heads.

Similar to MLA, queries are computed via low-rank compression:

$$\mathbf{C}_{\text{Q}} = \mathbf{H}\mathbf{A}_{\text{Q}}, \quad \overline{\mathbf{Q}}_{\text{NoPE}} = \mathbf{C}_{\text{Q}}\mathbf{B}_{\text{Q}}, \quad \overline{\mathbf{Q}}_{\text{RoPE}} = \mathrm{RoPE}\left(\mathbf{C}_{\text{Q}}\mathbf{B}_{\text{QR}}\right),$$
$$\mathbf{A}_{\text{Q}} \in \mathbb{R}^{d\times d'_c}, \quad \mathbf{B}_{\text{Q}} \in \mathbb{R}^{d'_c\times(hd_h)}, \quad \mathbf{B}_{\text{QR}} \in \mathbb{R}^{d'_c\times d_h^R h}.$$

Instead of a single compressed KV state, GLA computes g independent compressed states:

$$\mathbf{C}^{(i)}_{\text{KV}} = \mathbf{H}\mathbf{A}^{(i)}_{\text{KV}}, \quad \mathbf{A}^{(i)}_{\text{KV}} \in \mathbb{R}^{d\times(d_c/g)}, \quad \mathbf{C}^{(i)}_{\text{KV}} \in \mathbb{R}^{n\times(d_c/g)}, \quad i \in \{1,\ldots,g\}, \tag{15}$$

where d_c is the total latent dimension and each group uses d_c/g dimensions.

The RoPE keys remain shared across all groups:

$$\mathbf{K}_{\text{RoPE}} = \mathrm{RoPE}\left(\mathbf{H}\mathbf{A}_{\text{KR}}\right), \quad \mathbf{A}_{\text{KR}} \in \mathbb{R}^{d\times d_h^R}, \quad \mathbf{K}_{\text{RoPE}} \in \mathbb{R}^{n\times d_h^R}.$$

During inference, we cache $\left\{\mathbf{C}^{(1)}_{\text{KV}},\ldots,\mathbf{C}^{(g)}_{\text{KV}}\right\}$ and $\mathbf{K}_{\text{RoPE}}$ with total KV cache size of $d_c + d_h^R$ per token.

Each group independently reconstructs its keys and values:

$$\overline{\mathbf{K}}^{(i)}_{\text{NoPE}} = \mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{K}}, \quad \overline{\mathbf{V}}^{(i)} = \mathbf{C}^{(i)}_{\text{KV}}\mathbf{B}^{(i)}_{\text{V}},$$
$$\mathbf{B}^{(i)}_{\text{K}}, \mathbf{B}^{(i)}_{\text{V}} \in \mathbb{R}^{(d_c/g)\times(h_g d_h)}, \quad i \in \{1,\ldots,g\},$$

where $h_g = h/g$ is the number of heads per group.

Reshape into per-head form for each group:

$$\mathbf{K}^{(i)}_{\text{NoPE}} = \mathrm{reshape}\left(\overline{\mathbf{K}}^{(i)}_{\text{NoPE}}, [n, h_g, d_h]\right), \quad \mathbf{V}^{(i)} = \mathrm{reshape}\left(\overline{\mathbf{V}}^{(i)}, [n, h_g, d_h]\right).$$

Construct position-aware keys for each group by repeating the shared RoPE keys:

$$\mathbf{K}^{(i)} = \mathrm{Concat}\left[\mathbf{K}^{(i)}_{\text{NoPE}}, \mathrm{repeat}\left(\mathbf{K}_{\text{RoPE}}, h_g\right)\right] \in \mathbb{R}^{n\times h_g\times(d_h+d_h^R)}.$$

Partition queries into g groups along the head dimension:

$$\mathbf{Q} = \left[\mathbf{Q}^{(1)},\ldots,\mathbf{Q}^{(g)}\right], \quad \mathbf{Q}^{(i)} \in \mathbb{R}^{n\times h_g\times(d_h+d_h^R)}.$$

Compute attention independently for each group:

$$\mathbf{O}^{(i)} = \mathrm{Attention}\left(\mathbf{Q}^{(i)}, \mathbf{K}^{(i)}, \mathbf{V}^{(i)}\right) \in \mathbb{R}^{n\times h_g\times d_h}, \quad i \in \{1,\ldots,g\}.$$

Concatenate outputs from all groups:

$$\mathbf{O} = \mathrm{Concat}\left[\mathbf{O}^{(1)},\ldots,\mathbf{O}^{(g)}\right] \in \mathbb{R}^{n\times h\times d_h}.$$

C.5 Tensor Product Attention (TPA)

TPA achieves KV cache compression through low-rank factorization. It decomposes the keys and values into two components: a set of r shared basis vectors (or "attention heads") and token-specific coefficient matrices. The keys and values are reconstructed by linearly combining the shared basis vectors according to these coefficients. This allows storing only the coefficients and basis vectors in the cache.

Given a sequence of n tokens with hidden states $\mathbf{H} \in \mathbb{R}^{n\times d}$, TPA first computes the query, key, and value as follows:

$$\begin{aligned}
\overline{\mathbf{A}}_{\mathrm{Q}} &= \mathbf{H}\mathbf{W}_{\mathrm{AQ}}, \quad \mathbf{W}_{\mathrm{AQ}} \in \mathbb{R}^{d\times R_{\mathrm{Q}}h}, \quad \overline{\mathbf{A}}_{\mathrm{Q}} \in \mathbb{R}^{n\times R_{\mathrm{Q}}h},\\
\overline{\mathbf{B}}_{\mathrm{Q}} &= \mathbf{H}\mathbf{W}_{\mathrm{BQ}}, \quad \mathbf{W}_{\mathrm{BQ}} \in \mathbb{R}^{d\times R_{\mathrm{Q}}d_h}, \quad \overline{\mathbf{B}}_{\mathrm{Q}} \in \mathbb{R}^{n\times R_{\mathrm{Q}}d_h},\\
\overline{\mathbf{A}}_{\mathrm{K}} &= \mathbf{H}\mathbf{W}_{\mathrm{AK}}, \quad \mathbf{W}_{\mathrm{AK}} \in \mathbb{R}^{d\times R_{\mathrm{KV}}h}, \quad \overline{\mathbf{A}}_{\mathrm{K}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}h},\\
\overline{\mathbf{B}}_{\mathrm{K}} &= \mathbf{H}\mathbf{W}_{\mathrm{BK}}, \quad \mathbf{W}_{\mathrm{BK}} \in \mathbb{R}^{d\times R_{\mathrm{KV}}d_h}, \quad \overline{\mathbf{B}}_{\mathrm{K}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}d_h},\\
\overline{\mathbf{A}}_{\mathrm{V}} &= \mathbf{H}\mathbf{W}_{\mathrm{AV}}, \quad \mathbf{W}_{\mathrm{AV}} \in \mathbb{R}^{d\times R_{\mathrm{KV}}h}, \quad \overline{\mathbf{A}}_{\mathrm{V}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}h},\\
\overline{\mathbf{B}}_{\mathrm{V}} &= \mathbf{H}\mathbf{W}_{\mathrm{BV}}, \quad \mathbf{W}_{\mathrm{BV}} \in \mathbb{R}^{d\times R_{\mathrm{KV}}d_h}, \quad \overline{\mathbf{B}}_{\mathrm{V}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}d_h},
\end{aligned}$$

We reshape the projections into 3D tensors:

$$\begin{aligned}
\mathbf{A}_{\mathrm{Q}} &= \mathrm{reshape}\left(\overline{\mathbf{A}}_{\mathrm{Q}},\ [n,\ R_{\mathrm{Q}},\ h]\right), \quad \mathbf{B}_{\mathrm{Q}} = \mathrm{reshape}\left(\overline{\mathbf{B}}_{\mathrm{Q}},\ [n,\ R_{\mathrm{Q}},\ d_h]\right),\\
\mathbf{A}_{\mathrm{K}} &= \mathrm{reshape}\left(\overline{\mathbf{A}}_{\mathrm{K}},\ [n,\ R_{\mathrm{KV}},\ h]\right), \quad \mathbf{B}_{\mathrm{K}} = \mathrm{reshape}\left(\overline{\mathbf{B}}_{\mathrm{K}},\ [n,\ R_{\mathrm{KV}},\ d_h]\right),\\
\mathbf{A}_{\mathrm{V}} &= \mathrm{reshape}\left(\overline{\mathbf{A}}_{\mathrm{V}},\ [n,\ R_{\mathrm{KV}},\ h]\right), \quad \mathbf{B}_{\mathrm{V}} = \mathrm{reshape}\left(\overline{\mathbf{B}}_{\mathrm{V}},\ [n,\ R_{\mathrm{KV}},\ d_h]\right),
\end{aligned}$$

so that $\mathbf{A}_{\mathrm{Q}} \in \mathbb{R}^{n\times R_{\mathrm{Q}}\times h}$, $\mathbf{B}_{\mathrm{Q}} \in \mathbb{R}^{n\times R_{\mathrm{Q}}\times d_h}$, $\mathbf{A}_{\mathrm{K}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}\times h}$, $\mathbf{B}_{\mathrm{K}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}\times d_h}$, $\mathbf{A}_{\mathrm{V}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}\times h}$, and $\mathbf{B}_{\mathrm{V}} \in \mathbb{R}^{n\times R_{\mathrm{KV}}\times d_h}$. The final query, key, and value are computed as:

$$\begin{aligned}
\mathbf{Q}_{[i,:,:]} &= \frac{1}{R_{\mathrm{Q}}}\mathbf{A}_{\mathrm{Q}}\left[i,:,:\right]^{\top} \mathrm{RoPE}\left(\mathbf{B}_{\mathrm{Q}}\left[i,:,:\right]\right), \quad \mathbf{Q}_{[i,:,:]} \in \mathbb{R}^{h\times d_h},\\
\mathbf{K}_{[i,:,:]} &= \frac{1}{R_{\mathrm{KV}}}\mathbf{A}_{\mathrm{K}}\left[i,:,:\right]^{\top} \mathrm{RoPE}\left(\mathbf{B}_{\mathrm{K}}\left[i,:,:\right]\right), \quad \mathbf{K}_{[i,:,:]} \in \mathbb{R}^{h\times d_h},\\
\mathbf{V}_{[i,:,:]} &= \frac{1}{R_{\mathrm{KV}}}\mathbf{A}_{\mathrm{V}}\left[i,:,:\right]^{\top} \mathbf{B}_{\mathrm{V}}\left[i,:,:\right], \quad \mathbf{V}_{[i,:,:]} \in \mathbb{R}^{h\times d_h},
\end{aligned}$$

where $\mathbf{A}_{\mathrm{K}}$, $\mathbf{B}_{\mathrm{K}}$, $\mathbf{B}_{\mathrm{V}}$, and $\mathbf{A}_{\mathrm{V}}$ are cached during inference.

Remark 4. *We analyze the translation equivariance of TPA by focusing on a single attention head. Let $\mathbf{q}_m \in \mathbb{R}^{d_h}$ and $\mathbf{k}_n \in \mathbb{R}^{d_h}$ denote the query and key vectors for the i-th head at positions m and n, respectively. In TPA, they are computed as:*

$$\mathbf{q}_m^{(i)} = \frac{1}{R_Q}\sum_{r_q=1}^{R_Q} \alpha_Q^{(m,r_q,i)} \cdot \tilde{\mathbf{b}}_Q^{(m,r_q)}, \quad \mathbf{k}_n^{(i)} = \frac{1}{R_{KV}}\sum_{r_{kv}=1}^{R_{KV}} \alpha_K^{(n,r_{kv},i)} \cdot \tilde{\mathbf{b}}_K^{(n,r_{kv})},$$

where we define the shorthand notations:

$$\begin{aligned}
\alpha_Q^{(m,r_q,i)} &:= \mathbf{A}_Q\left[m, r_q, i\right], \quad \alpha_K^{(n,r_{kv},i)} := \mathbf{A}_K\left[n, r_{kv}, i\right],\\
\tilde{\mathbf{b}}_Q^{(m,r_q)} &:= \mathrm{RoPE}\left(\mathbf{B}_Q\left[m, r_q, :\right]\right), \quad \tilde{\mathbf{b}}_K^{(n,r_{kv})} := \mathrm{RoPE}\left(\mathbf{B}_K\left[n, r_{kv}, :\right]\right).
\end{aligned}$$

Then, the inner product between query and key becomes:

$$\begin{aligned}
\left\langle \mathbf{q}_m^{(i)},\ \mathbf{k}_n^{(i)} \right\rangle &= \left\langle \frac{1}{R_Q}\sum_{r_q=1}^{R_Q} \alpha_Q^{(m,r_q,i)} \cdot \tilde{\mathbf{b}}_Q^{(m,r_q)},\ \frac{1}{R_{KV}}\sum_{r_{kv}=1}^{R_{KV}} \alpha_K^{(n,r_{kv},i)} \cdot \tilde{\mathbf{b}}_K^{(n,r_{kv})} \right\rangle\\
&= \frac{1}{R_Q R_{KV}} \sum_{r_q=1}^{R_Q}\sum_{r_{kv}=1}^{R_{KV}} \alpha_Q^{(m,r_q,i)} \cdot \alpha_K^{(n,r_{kv},i)} \cdot \left\langle \tilde{\mathbf{b}}_Q^{(m,r_q)},\ \tilde{\mathbf{b}}_K^{(n,r_{kv})} \right\rangle.
\end{aligned}$$

Algorithm 1 MLRA Decoding without KV Materialization

```
Require: Query $\mathbf{q} = \mathrm{Concat}[\mathbf{q}_{\text{NoPE}}, \mathbf{q}_{\text{RoPE}}]$ where $\mathbf{q}_{\text{NoPE}} \in \mathbb{R}^{h \times d_h}$, $\mathbf{q}_{\text{RoPE}} \in \mathbb{R}^{h \times d_h^R}$
Require: KV cache: $\mathbf{U}_{\text{KV}} \in \mathbb{R}^{n \times d_u}$, $\mathbf{C}_{\text{KV}} \in \mathbb{R}^{n \times h \times r}$, $\mathbf{K}_{\text{RoPE}} \in \mathbb{R}^{n \times d_h^R}$
Require: Up-projection matrices: $\mathbf{B}^{(i)}_{\text{base,K}}, \mathbf{B}^{(i)}_{\text{base,V}} \in \mathbb{R}^{d_u \times d_h}$, $\mathbf{B}^{(i)}_{\text{lora,K}}, \mathbf{B}^{(i)}_{\text{lora,V}} \in \mathbb{R}^{r \times d_h}$ for $i \in [1, h]$
Require: Scaling factors: $\tau = 1/\sqrt{d_h + d_h^R}$, $\alpha$ (tunable hyperparameter)
Ensure: Attention output $\mathbf{O} \in \mathbb{R}^{h \times d_h}$
 1:
 2: // Step 1: Absorb key up-projections into queries and compute logits
 3: for $i = 1$ to $h$ do
 4:     // Base path: absorb key up-projection
 5:     $\tilde{\mathbf{q}}^{(i)}_{\text{base}} \leftarrow \mathbf{q}^{(i)}_{\text{NoPE}} \left(\mathbf{B}^{(i)}_{\text{base,K}}\right)^\top$ {$\tilde{\mathbf{q}}^{(i)}_{\text{base}} \in \mathbb{R}^{1 \times d_u}$}
 6:     $\text{logits}^{(i)}_{\text{base}} \leftarrow \tau \tilde{\mathbf{q}}^{(i)}_{\text{base}} \mathbf{U}_{\text{KV}}^\top + \tau \mathbf{q}^{(i)}_{\text{RoPE}} \mathbf{K}_{\text{RoPE}}^\top$ {$\in \mathbb{R}^{1 \times n}$}
 7:     $\boldsymbol{\lambda}^{(i)}_{\text{base}} \leftarrow \mathrm{Softmax}\left(\text{logits}^{(i)}_{\text{base}}\right)$
 8:
 9:     // Low-rank path: absorb key up-projection
10:     $\tilde{\mathbf{q}}^{(i)}_{\text{lora}} \leftarrow \alpha \mathbf{q}^{(i)}_{\text{NoPE}} \left(\mathbf{B}^{(i)}_{\text{lora,K}}\right)^\top$ {$\tilde{\mathbf{q}}^{(i)}_{\text{lora}} \in \mathbb{R}^{1 \times r}$}
11:     $\text{logits}^{(i)}_{\text{lora}} \leftarrow \tau \tilde{\mathbf{q}}^{(i)}_{\text{lora}} \left(\mathbf{C}^{(i)}_{\text{KV}}\right)^\top + \tau \mathbf{q}^{(i)}_{\text{RoPE}} \mathbf{K}_{\text{RoPE}}^\top$ {$\in \mathbb{R}^{1 \times n}$}
12:     $\boldsymbol{\lambda}^{(i)}_{\text{lora}} \leftarrow \mathrm{Softmax}\left(\text{logits}^{(i)}_{\text{lora}}\right)$
13: end for
14:
15: // Step 2: Aggregate compressed KV and up-project to obtain outputs
16: for $i = 1$ to $h$ do
17:     // Base path: aggregate then up-project values
18:     $\tilde{\mathbf{z}}^{(i)}_{\text{base}} \leftarrow \boldsymbol{\lambda}^{(i)}_{\text{base}} \mathbf{U}_{\text{KV}}$ {Weighted sum in compressed space: $\in \mathbb{R}^{1 \times d_u}$}
19:     $\mathbf{z}^{(i)}_{\text{base}} \leftarrow \tilde{\mathbf{z}}^{(i)}_{\text{base}} \mathbf{B}^{(i)}_{\text{base,V}}$ {Up-project to output: $\in \mathbb{R}^{1 \times d_h}$}
20:
21:     // Low-rank path: aggregate then up-project values
22:     $\tilde{\mathbf{z}}^{(i)}_{\text{lora}} \leftarrow \boldsymbol{\lambda}^{(i)}_{\text{lora}} \mathbf{C}^{(i)}_{\text{KV}}$ {Weighted sum in compressed space: $\in \mathbb{R}^{1 \times r}$}
23:     $\mathbf{z}^{(i)}_{\text{lora}} \leftarrow \alpha \tilde{\mathbf{z}}^{(i)}_{\text{lora}} \mathbf{B}^{(i)}_{\text{lora,V}}$ {Up-project to output: $\in \mathbb{R}^{1 \times d_h}$}
24:
25:     // Combine base and low-rank paths
26:     $\mathbf{O}^{(i)} \leftarrow \mathbf{z}^{(i)}_{\text{base}} + \mathbf{z}^{(i)}_{\text{lora}}$
27: end for
28:
29: return $\mathbf{O} = \left[\mathbf{O}^{(1)}, \ldots, \mathbf{O}^{(h)}\right] \in \mathbb{R}^{h \times d_h}$
```

Since α_Q and α_K are position-independent and RoPE satisfies translation equivariance, we have, for any r_q and r_{kv}:

$$\left\langle \tilde{\mathbf{b}}_Q^{(m+t, r_q)},\ \tilde{\mathbf{b}}_K^{(n+t, r_{kv})} \right\rangle = \left\langle \tilde{\mathbf{b}}_Q^{(m, r_q)},\ \tilde{\mathbf{b}}_K^{(n, r_{kv})} \right\rangle.$$

Therefore, the query-key inner product is invariant under joint position translations:

$$\left\langle \mathbf{q}^{(i)}_{m+t},\ \mathbf{k}^{(i)}_{n+t} \right\rangle = \left\langle \mathbf{q}^{(i)}_{m},\ \mathbf{k}^{(i)}_{n} \right\rangle.$$

This confirms that TPA preserves translation equivariance.

D PSEUDOCODE

E Llama-3 Architecture

Given hidden states $\mathbf{H} \in \mathbb{R}^{n\times d}$ for a sequence of n tokens, we first compute the attention output

$$\mathbf{O}_{\text{attn}} = \text{Attention}(\mathbf{H}) \in \mathbb{R}^{n\times(hd_h)},$$

then project back to the model dimension and add a residual:

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{O}_{\text{attn}}\mathbf{W}_{\text{O,attn}}, \qquad \mathbf{W}_{\text{O,attn}} \in \mathbb{R}^{(hd_h)\times d}.$$

Next, an MLP block (gated form) is applied:

$$\mathbf{O}_{\text{mlp}} = \sigma\left(\mathbf{H}\mathbf{W}_1\right) \odot \left(\mathbf{H}\mathbf{W}_2\right), \qquad \mathbf{W}_1, \mathbf{W}_2 \in \mathbb{R}^{d\times d_i},$$

followed by the output projection and residual:

$$\mathbf{H} \leftarrow \mathbf{H} + \mathbf{O}_{\text{mlp}}\mathbf{W}_{\text{O,mlp}}, \qquad \mathbf{W}_{\text{O,mlp}} \in \mathbb{R}^{d_i\times d},$$

where $\sigma(\cdot)$ is an elementwise nonlinearity function such as SiLU and $\odot$ denotes elementwise multiplication.

F Experimental Setup

F.1 Model Hyperparameters

We adopt a Llama-3 architecture and a GPT-2 tokenizer with a 50k vocabulary. Following Zadouri et al. (2025), we initialize our MHA baseline using the GPT-3 configuration at each target model size; this baseline serves as the anchor for parameter count. For all other attention variants, we widen the MLP layers until each model's total parameters match the MHA baseline. We report architecture hyperparameters for MHA, MQA, GQA, MLA, TPA, MLA, and MLRA in Tables 5, 6, 7, 8, 9, 10, 11. For GQA we set $g = \frac{h}{4}$. For MLA, we follow the paper's setup with $d_c' = 12d_h$, $d_c' = 4d_h$, and $d_h^R = 0.5d_h$. For GLA, we follow the paper's setup with $d_c' = 8d_h$, $d_c' = 4d_h$, and $d_h^R = 0.5d_h$. For TPA, we follow the paper's setup with $R_{\text{Q}} = 6$ and $R_{\text{KV}} = 2$. For MLRA, we set $d_u' = 2d_h$, $d_u' = d_h$, $r' = \frac{3d_h}{h}$, and $r = \frac{6d_h}{h}$.

Table 5: Model configuration for the three model sizes for MHA in our experiments.

Model Size	# Parameters	# Layers	h	d	d_h	d_i
354M	353.94M	24	16	1024	64	2736
1.3B	1311.48M	24	16	2048	128	5464
2.9B	2872.59M	24	24	3072	128	8192

Table 6: Model configuration for the three model sizes for MQA in our experiments.

Model Size	# Parameters	# Layers	h	g	d	d_h	d_i
354M	353.94M	24	16	1	1024	64	3376
1.3B	1311.48M	24	16	1	2048	128	6744
2.9B	2872.00M	24	24	1	3072	128	10152

Table 7: Model configuration for the three model sizes for GQA in our experiments.

Model Size	# Parameters	# Layers	h	g	d	d_h	d_i
354M	353.94M	24	16	4	1024	64	3248
1.3B	1311.48M	24	16	4	2048	128	6488
2.9B	2872.59M	24	24	6	3072	128	9728

Table 8: Model configuration for the three model sizes for MLA in our experiments.

Model Size	# Parameters	# Layers	h	d'_c	d_c	d	d_h	d_h^R	d_i
354M	353.58M	24	16	768	256	1024	64	32	2848
1.3B	1311.13M	24	16	1536	512	2048	128	64	5696
2.9B	2872.05M	24	24	1536	512	3072	128	64	9448

Table 9: Model configuration for the three model sizes for TPA in our experiments.

Model Size	# Parameters	# Layers	h	R_{Q}	R_{KV}	d	d_h	d_i
354M	354.14M	24	16	6	2	1024	64	3496
1.3B	1311.48M	24	16	6	2	2048	128	7032
2.9B	2873.18M	24	24	6	2	3072	128	10760

Table 10: Model configuration for the three model sizes for GLA in our experiments.

Model Size	# Parameters	# Layers	h	d'_c	d_c	d	d_h	d_h^R	d_i
354M	353.96M	24	16	512	256	1024	64	32	3152
1.3B	1311.51M	24	16	1024	512	2048	128	64	6296
2.9B	2872.63M	24	24	1024	512	3072	128	64	10048

Table 11: Model configuration for the two model sizes for MLRA in our experiments.

Model Size	# Parameters	# Layers	h	d'_u	d_u	r'	r	d	d_h	d_h^R	d_i	α	γ
1.3B	1311.88M	24	16	256	128	48	24	2048	128	64	6728	2	2
2.9B	2872.01M	24	24	256	128	32	16	3072	128	64	10488	2	1

F.2 Training Setup

Training Configuration. For pretraining, we adopt GPT-3 settings. We use AdamW (Loshchilov & Hutter, 2017) with $(\beta_1, \beta_2) = (0.9, 0.95), \epsilon = 10^{-8}$, weight decay 0.1 , and gradient clipping at 1.0 . The learning rate is linearly warmed up for the first 2,000 steps, then annealed with cosine decay (Loshchilov & Hutter, 2016) to 10% of the peak. Peak learning rates are $3 \times 10^{-4}, 2 \times 10^{-4}$, and 1.6×10^{-4} for the 354M , 1.3B, and 2.9B models, respectively. We use a context length of 2,048 tokens and a global batch of 240 sequences, yielding 491,520 tokens per step ($\approx$ 0.5M). We train for 200,000 steps, totaling 98.3B tokens. By contrast, GPT-3 uses 1.0M tokens/step for 1.3B and 2.9B experiments.

Table 12: Training configuration for the three model sizes in our experiments.

Model Size	Micro-batch Size	Batch Size	Peak Learning Rate	Total Steps
354M	120	240	3×10^{-4}	200,000
1.3B	40	240	2×10^{-4}	200,000
2.9B	8	240	1.6×10^{-4}	200,000

Initialization. For MHA, MQA, GQA, MLA, and GLA, all learnable parameters are initialized with a normal distribution $\mathcal{N}(0, \sigma = 0.02)$.

For TPA, we follow the paper's setup. The matrices $\mathbf{W}_{\text{AQ}}, \mathbf{W}_{\text{BQ}}, \mathbf{W}_{\text{AK}}, \mathbf{W}_{\text{BK}}, \mathbf{W}_{\text{AV}}, \mathbf{W}_{\text{BV}}$ use Xavier uniform initialization (Glorot & Bengio, 2010): each entry is sampled from $\mathcal{U}(-\text{bound}, \text{bound})$ with bound $= \sqrt{\frac{6}{d_{\text{in}}+d_{\text{out}}}}$, where d_{in} and d_{out} are the input and output dimensions of the respective matrix. The output projections $\mathbf{W}_{\text{O, attn}}$ and $\mathbf{W}_{\text{O, mlp}}$ are *zero-initialized*, while all remaining parameters use $\mathcal{N}(0, \sigma = 0.02)$.

In MLRA, we also zero-initialize $\mathbf{W}_{\text{O, attn}}$ and $\mathbf{W}_{\text{O, mlp}}$; all other parameters are initialized with $\mathcal{N}(0, \sigma = 0.02)$.

We conduct an ablation study on the baseline mechanisms comparing zero vs. $\mathcal{N}(0, \sigma = 0.02)$ initialization for $\mathbf{W}_{\text{O,attn}}$ and $\mathbf{W}_{\text{O,mlp}}$. Medium-size models are trained for 100B tokens using the configuration in Appendix F.2. We evaluate perplexity on 6 datasets: FineWeb-Edu (Penedo et al., 2024), Wikipedia, C4, Common Crawl, Pile (Gao et al., 2020), and ArXiv, each evaluated using 100M tokens. Results are reported in Table 13. We find that $\mathcal{N}(0, 0.02)$ performs better for MHA, MQA, GQA, MLA, and GLA, whereas zero initialization is better for TPA.

Table 13: Ablation study on initialization ($\mathcal{N}(0, \sigma = 0.02)$ vs. zero) for $\mathbf{W}_{\text{O, attn}}$ and $\mathbf{W}_{\text{O, mlp}}$.

Method	Initialization	FineWeb-Edu	Wikipedia	C4	Common Crawl	Pile	ArXiv	Avg
MHA	$\mathcal{N}(0, \sigma = 0.02)$	13.159	20.369	21.732	20.230	15.974	17.836	18.217
MHA	0	13.604	20.579	22.432	20.838	15.812	18.255	18.586
MQA	$\mathcal{N}(0, \sigma = 0.02)$	13.375	20.261	22.157	20.625	15.380	17.947	18.291
MQA	0	13.563	21.468	22.471	21.129	16.157	18.632	18.903
GQA	$\mathcal{N}(0, \sigma = 0.02)$	13.158	19.862	21.810	20.239	16.253	17.969	18.215
GQA	0	13.352	20.360	22.074	20.604	15.730	17.861	18.330
MLA	$\mathcal{N}(0, \sigma = 0.02)$	13.095	19.693	21.604	19.860	14.914	17.176	17.724
MLA	0	13.184	20.599	21.768	20.136	15.609	17.228	18.087
TPA	0	13.140	19.856	21.770	20.187	15.138	17.554	17.941
TPA	$\mathcal{N}(0, \sigma = 0.02)$	13.486	20.511	22.336	20.788	15.279	17.849	18.375
GLA	$\mathcal{N}(0, \sigma = 0.02)$	13.057	18.932	21.575	19.926	14.477	17.140	17.518
GLA	0	13.133	19.651	21.684	19.975	14.680	17.217	17.723

G THE USE OF LARGE LANGUAGE MODELS

We use LLMs to polish the writing and reorganize tables and figures.