Tensor Product Attention Is All You Need

Yifan Zhang[*◇1,2] **Yifeng Liu**[*3] **Huizhuo Yuan**[3] **Zhen Qin**[4]
Yang Yuan[1,2] **Quanquan Gu**[3] **Andrew Chi-Chih Yao**[1,2†]
[1]IIIS, Tsinghua University [2]Shanghai Qi Zhi Institute
[3]University of California, Los Angeles [4]TapTap

Abstract

Scaling language models to handle longer input sequences typically necessitates large key-value (KV) caches, resulting in substantial memory overhead during inference. In this paper, we propose **T**ensor **P**roduct **A**ttention (TPA), a novel attention mechanism that uses tensor decompositions to represent queries, keys, and values compactly, significantly shrinking KV cache size at inference time. By factorizing these representations into contextual low-rank components (contextual factorization) and seamlessly integrating with RoPE, TPA achieves improved model quality alongside memory efficiency. Based on TPA, we introduce the **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6), a new model architecture for sequence modeling. Through extensive empirical evaluation of language modeling tasks, we demonstrate that T6 exceeds the performance of standard Transformer baselines including MHA, MQA, GQA, and MLA across various metrics, including perplexity and a range of renowned evaluation benchmarks. Notably, TPA's memory efficiency enables the processing of significantly longer sequences under fixed resource constraints, addressing a critical scalability challenge in modern language models. The code is available at https://github.com/tensorgi/T6.

1 Introduction

Large language models (LLMs) have revolutionized natural language processing, demonstrating exceptional performance across tasks (Brown et al., 2020; Chowdhery et al., 2023; Touvron et al., 2023; Bubeck et al., 2023). As these models evolve, their ability to process longer contexts becomes increasingly important for sophisticated applications such as document analysis, complex reasoning, and code completions. However, managing longer sequences during inference poses significant computational and memory challenges, particularly due to the storage of key-value (KV) caches (Zhang et al., 2023c; Liu et al., 2024c). Because memory consumption grows linearly with sequence length, the maximum context window is limited by practical hardware constraints.

A variety of solutions have been explored to address this memory bottleneck. Some approaches compress or selectively prune cached states through sparse attention patterns (Child et al., 2019) or token eviction strategies (Zhang et al., 2023c; Xiao et al., 2024; Ribar et al., 2024), though such methods risk discarding tokens that may later prove important. Other work proposes off-chip storage of key-value states (He & Zhai, 2024), at the expense of increased I/O latency. Attention variants like multi-query attention (MQA) (Shazeer, 2019) and grouped-query attention (GQA) (Ainslie et al., 2023) reduce per-token cache requirements by sharing keys and values across heads, but often compromise flexibility or require significant architectural modifications. Meanwhile, low-rank weight factorization methods such as LoRA (Hu et al., 2022) effectively reduce fine-tuning memory, yet do not address the KV cache overhead that dominates runtime. The recently introduced Multi-head Latent Attention (MLA) in Deepseek-V2 (Liu et al., 2024a) caches compressed key-value repre-

[*] Equal contribution; ◇ Tech lead; † Corresponding author.

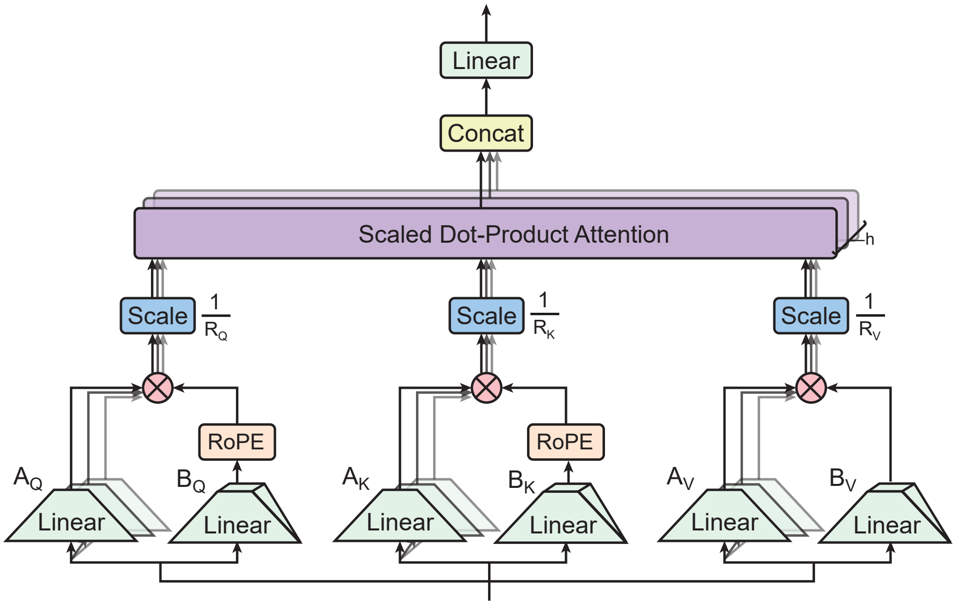


Figure 1: Tensor Product Attention (TPA) in the **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6). Different from multi-head attention, in each layer, firstly the hidden state goes through different linear layers to get the latent factor matrices $\mathbf{A}$'s and $\mathbf{B}$'s for query, key, and value. We additionally apply RoPE to $\mathbf{B}_Q$ and $\mathbf{B}_K$ for query and key. Then the multi-head query, key, and value vectors are attained by the tensor product of $\mathbf{A}_{(\cdot)}$ and $\mathbf{B}_{(\cdot)}$. Finally, the output of TPA is produced by scaled dot-product attention followed by linear projection of concatenated results of multiple heads.

sentations but needs additional position-encoded parameters per head due to incompatibility with Rotary Position Embedding (RoPE) efficiently (Su et al., 2024b).

In order to overcome the limitations of existing approaches, we introduce *Tensor Product Attention* (TPA), as illustrated in Figure 1, a novel architecture that uses higher-order tensors to factorize queries (Q), keys (K), and values (V) during attention computation. By dynamically factorizing *activations* rather than static weights (e.g., LoRA), TPA constructs low-rank, contextual representations that substantially reduce KV cache memory usage with improved representational capacity. In practice, TPA can reduce the memory overhead by an order of magnitude compared to standard multi-head attention (MHA) with lower pretraining validation loss (perplexity) and improved downstream performance.

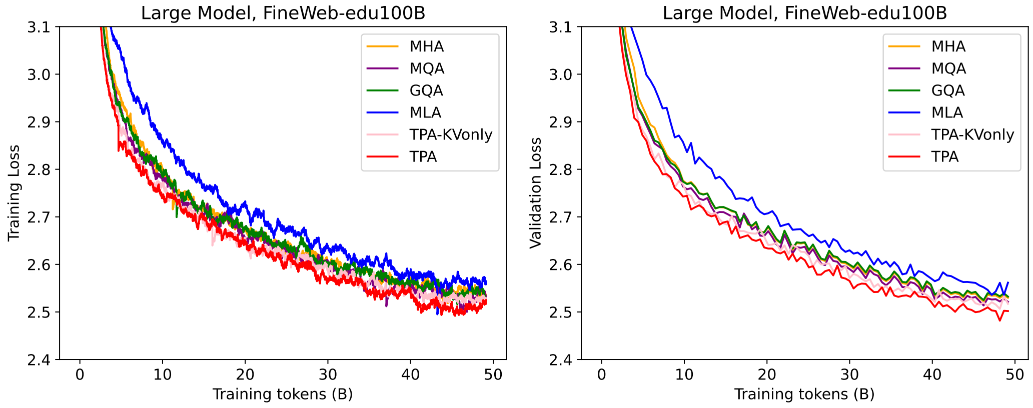
A key advantage of TPA is its native compatibility with rotary positional embeddings (RoPE) (Su et al., 2024b), enabling a straightforward drop-in replacement for multi-head attention (MHA) layers in modern LLM architectures such as LLaMA (Touvron et al., 2023) and Gemma (Team et al., 2024).

Our primary contributions are summarized as follows:

- We propose **Tensor Product Attention (TPA)**, A mechanism that factorizes $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ activations using *contextual* tensor-decompositions to achieve $10\times$ or more reduction in inference-time KV cache size relative to standard attention mechanism (Vaswani et al., 2017) with improved performance compared to previous methods such as MHA, MQA, GQA, and MLA. In addition, we **unify existing attention mechanisms** by revealing that MHA, MQA, and GQA *all* arise naturally as non-contextual variants of TPA.

- We propose **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6), a new TPA-based model architecture for sequence modeling. On language modeling experiments, T6 consistently improves validation perplexity and downstream evaluation performance with reduced KV cache size.

- We show **TPA** integrates seamlessly with RoPE (Su et al., 2024b), facilitating easy adoption in popular foundation model architectures such as LLaMA and Gemma.



(a) Training Loss (b) Validation Loss

Figure 2: Training loss and validation loss of pretraining large-size (773M) models with different attention mechanisms on the FineWeb-Edu-100B dataset.

2 Background

In this section, we review several classical forms of attention: Scaled Dot-Product Attention, Multi-Head Attention (MHA) (Vaswani et al., 2017), Multi-Query Attention (MQA) (Shazeer, 2019), and Grouped Query Attention (GQA) (Ainslie et al., 2023), as well as Rotary Position Embedding (RoPE, Su et al. (2024b)). We also introduce a recent method called Multi-head Latent Attention (MLA) used in DeepSeek-V2 (Liu et al., 2024a) and DeepSeek-V3 (Liu et al., 2024b).

Notations. We use bold uppercase letters (e.g., $\mathbf{X}$, $\mathbf{Q}$) for matrices, bold lowercase (e.g., $\mathbf{a}$, $\mathbf{b}$) for vectors, and italic uppercase (e.g., $\boldsymbol{W}_i^Q$) for learnable parameter matrices. We denote by $[n]$ the set $\{1, \ldots, n\}$ for some positive integer n. We use $\top$ to denote the transpose of a vector or a matrix. Let d_{model} be the embedding dimension, h the number of attention heads, d_h the dimension per head, $\mathbf{x}_t \in \mathbb{R}^d$ the input for the t-th token at a given attention layer, $\mathbf{X} \in \mathbb{R}^{T \times d_{\text{model}}}$ denotes the input embeddings for T tokens, and $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{T \times h \times d_h}$ denote the queries, keys, and values of h heads for T tokens. With a little abuse of notation, $\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i \in \mathbb{R}^{T \times d_h}$ denote the i-th head of queries, keys, and values, and $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{h \times d_h}$ denote the heads of the query, key, and value for t-th token.

Throughout the paper, $\boldsymbol{W}^Q, \boldsymbol{W}^K, \boldsymbol{W}^V$ denote projection matrices for queries, keys, and values, respectively. In multi-head attention, each head is associated with its own set of $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K, \boldsymbol{W}_i^V$, and each has dimension $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K, \boldsymbol{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$, where d_k is typically set to d_h, the dimension of each head.[5] Similarly, we have an output projection matrix $\boldsymbol{W}^O \in \mathbb{R}^{(h \cdot d_h) \times d_{\text{model}}}$. For methods like MQA and GQA, some of these are shared or partially shared across heads, but their shapes remain consistent.

We define the tensor product of two vectors as follows: for vectors $\mathbf{a} \in \mathbb{R}^m, \mathbf{b} \in \mathbb{R}^n$, the tensor product of $\mathbf{a}$ and $\mathbf{b}$ is:

$$\mathbf{a} \otimes \mathbf{b} = \mathbf{C} \in \mathbb{R}^{m \times n}, \text{with } C_{ij} = a_i b_j,$$

where a_i and b_j are the i-th and j-th elements of $\mathbf{a}$ and $\mathbf{b}$ respectively, and C_{ij} is the (i, j)-th entry of $\mathbf{C}$. We also define the vectorization of a matrix $\mathbf{C} \in \mathbb{R}^{m \times n}$ by:

$$\text{vec}(\mathbf{C}) = \mathbf{d} \in \mathbb{R}^{mn}, \text{with } d_{i \cdot n + j} = C_{ij},$$

where $d_{i \cdot n + j}$ is the $(i \cdot n + j)$-th element of $\mathbf{d}$.

2.1 Scaled Dot-Product Attention

Scaled dot-product attention (Vaswani et al., 2017) determines how to focus on different parts of an input sequence by comparing queries ($\mathbf{Q}$) and keys ($\mathbf{K}$). It produces a weighted combination of the

[5] Often, one sets $h \times d_h = d_{\text{model}}$, so each head has query/key/value dimension d_h.

values ($\mathbf{V}$). Formally, the attention output is:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V},$$

where each of $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ is an $(n \times d_k)$ matrix for n tokens and key dimension d_k. The division by $\sqrt{d_k}$ stabilizes training by controlling the scale of the inner products.

2.2 Multi-Head Attention (MHA)

Multi-Head Attention (MHA) extends scaled dot-product attention by dividing the model's internal representation into several *heads*. Each head learns different projections for queries, keys, and values, allowing the model to attend to different types of information. For each token embedding $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$, MHA computes each head i as follows:

$$\mathbf{Q}_{t,i} = (\boldsymbol{W}_i^Q)^\top \mathbf{x}_t \in \mathbb{R}^{d_h}, \quad \mathbf{K}_{t,i} = (\boldsymbol{W}_i^K)^\top \mathbf{x}_t \in \mathbb{R}^{d_h}, \quad \mathbf{V}_{t,i} = (\boldsymbol{W}_i^V)^\top \mathbf{x}_t \in \mathbb{R}^{d_h},$$

$$\mathbf{head}_i = \text{Attention}\Big(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i\Big),$$

where $\boldsymbol{W}_i^Q, \boldsymbol{W}_i^K, \boldsymbol{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_h}$ are learnable projection matrices for the i-th head, $\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i \in \mathbb{R}^{T \times d_h}$. After computing each head's attention, the outputs are concatenated and mapped back to the original dimension via another matrix $\boldsymbol{W}^O \in \mathbb{R}^{hd_h \times d_{\text{model}}}$:

$$\text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\big(\mathbf{head}_1, \ldots, \mathbf{head}_h\big)\, \boldsymbol{W}^O.$$

MHA can capture a rich set of dependencies while each head focuses on different subspaces.

2.3 Multi-Query Attention (MQA)

Multi-Query Attention (MQA) (Shazeer, 2019) significantly reduces memory usage by *sharing* keys and values across heads, while still preserving unique query projections. For a sequence of embeddings $\mathbf{X} \in \mathbb{R}^{T \times d_{\text{model}}}$,

$$\mathbf{Q}_i = \mathbf{X}\boldsymbol{W}_i^Q, \quad \mathbf{K}_{\text{shared}} = \mathbf{X}\boldsymbol{W}_{\text{shared}}^K, \quad \mathbf{V}_{\text{shared}} = \mathbf{X}\boldsymbol{W}_{\text{shared}}^V.$$

Hence, each head i only has a distinct query $\mathbf{Q}_i \in \mathbb{R}^{T \times d_k}$, but shares the same key $\mathbf{K}_{\text{shared}} \in \mathbb{R}^{T \times d_k}$ and value $\mathbf{V}_{\text{shared}} \in \mathbb{R}^{T \times d_k}$. In practice, this means:

$$\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, \quad \boldsymbol{W}_{\text{shared}}^K, \boldsymbol{W}_{\text{shared}}^V \in \mathbb{R}^{d_{\text{model}} \times d_k}.$$

The resulting MQA operation is:

$$\text{MQA}(\mathbf{X}) = \text{Concat}\Big(\mathbf{head}_1, \ldots, \mathbf{head}_h\Big)\, \boldsymbol{W}^O,$$

where

$$\mathbf{head}_i = \text{Attention}\big(\mathbf{Q}_i, \mathbf{K}_{\text{shared}}, \mathbf{V}_{\text{shared}}\big).$$

By sharing these key and value projections, MQA cuts down on memory usage (especially for the key-value cache in autoregressive inference) but loses some expressivity since all heads must rely on the same key/value representations.

2.4 Grouped Query Attention (GQA)

Grouped Query Attention (GQA) (Ainslie et al., 2023) generalizes MHA and MQA by *grouping* heads. Specifically, we partition the h total heads into G groups. Each group has a single set of keys and values, but each individual head within that group still retains its own query projection. Formally, if $g(i)$ maps a head $i \in [h]$ to its group index $g \in [G]$, then:

$$\mathbf{K}_{g(i)} = \mathbf{X}\,\boldsymbol{W}_{g(i)}^K, \quad \mathbf{V}_{g(i)} = \mathbf{X}\,\boldsymbol{W}_{g(i)}^V, \quad \mathbf{Q}_i = \mathbf{X}\,\boldsymbol{W}_i^Q,$$

and

$$\text{head}_i = \text{Attention}\Big(\mathbf{Q}_i, \mathbf{K}_{g(i)}, \mathbf{V}_{g(i)}\Big).$$

Again, $\boldsymbol{W}_g^K, \boldsymbol{W}_g^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ for each group g, and $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$ for each head i. The complete output is again a concatenation of all heads:

$$\text{GQA}(\mathbf{X}) = \text{Concat}\Big(\text{head}_1, \ldots, \text{head}_h\Big)\boldsymbol{W}^O.$$

By adjusting G between 1 and h, GQA can interpolate between sharing all key/value projections across heads (i.e., MQA) and having one set of projections per head (i.e., MHA).

2.5 Rotary Position Embedding (RoPE)

Many recent LLMs use rotary position embedding (RoPE; Su et al., 2024b) to encode positional information in the query/key vectors. Specifically, let RoPE_t denote the rotation operator $\mathbf{T}_t \in \mathbb{R}^{d_h \times d_h}$ corresponding to the t-th position. $\mathbf{T}_t$ is a block-diagonal matrix, which consists of block-diagonal matrix $\begin{pmatrix} \cos(t\theta_j) & -\sin(t\theta_j) \\ \sin(t\theta_j) & \cos(t\theta_j) \end{pmatrix}$, $j \in \{1, \cdots, d_h/2\}$, where $\{\theta_j\}$ are pre-defined frequency parameters, e.g., $\theta_j = 1/10000^{2j/d_h}$. Then we define

$$\mathrm{RoPE}\left(\mathbf{Q}_t\right) \triangleq \mathbf{Q}_t \mathbf{T}_t, \quad \text{where } \mathbf{Q}_t \in \mathbb{R}^{h \times d_h}.$$

A fundamental property is that

$$\mathbf{T}_t \, \mathbf{T}_s^\top = \mathbf{T}_{t-s}, \tag{2.1}$$

which ensures that relative positions $(t - s)$ are preserved, thereby providing a form of translation invariance in the rotary position embedding.

2.6 Multi-head Latent Attention (MLA)

Below, we briefly outline the Multi-head Latent Attention (MLA) approach used by DeepSeek-V2 (Liu et al., 2024a) and DeepSeek-V3 (Liu et al., 2024b). MLA introduces a low-rank compression of the keys and values to reduce the Key-Value (KV) caching cost at inference.

$$\begin{aligned}
\mathbf{C}^{KV} &= \mathbf{X}\boldsymbol{W}^{DKV}, \quad (\boldsymbol{W}^{DKV} \in \mathbb{R}^{d_{\text{model}} \times d_c}), \\
\mathrm{Concat}\big(\mathbf{K}_1^C, \mathbf{K}_2^C, \ldots, \mathbf{K}_h^C\big) = \mathbf{K}^C &= \mathbf{C}^{KV}\boldsymbol{W}^{UK}, \quad (\boldsymbol{W}^{UK} \in \mathbb{R}^{d_c \times d_h h}), \\
\mathbf{K}^R &= \mathrm{RoPE}\big(\mathbf{X}\boldsymbol{W}^{KR}\big), \quad (\boldsymbol{W}^{KR} \in \mathbb{R}^{d_{\text{model}} \times d_h^R}), \\
\mathbf{K}_i &= \mathrm{Concat}\big(\mathbf{K}_i^C, \mathbf{K}^R\big), \\
\mathrm{Concat}\big(\mathbf{V}_1^C, \mathbf{V}_2^C, \ldots, \mathbf{V}_h^C\big) = \mathbf{V}^C &= \mathbf{C}^{KV}\boldsymbol{W}^{UV}, \quad (\boldsymbol{W}^{UV} \in \mathbb{R}^{d_c \times d_h h}),
\end{aligned}$$

where $\mathbf{C}^{KV} \in \mathbb{R}^{T \times d_c}$ is the compressed KV latent (with $d_c \ll d_h h$), and $\mathrm{RoPE}(\cdot)$ represents the RoPE transform applied to the separate key embeddings $\mathbf{K}^R$ of dimension d_h^R. Thus, only $\mathbf{C}^{KV}$ and $\mathbf{K}^R$ need to be cached, reducing KV memory usage while largely preserving performance compared to standard MHA (Vaswani et al., 2017).

MLA also compresses the queries, lowering their training-time memory footprint:

$$\begin{aligned}
\mathbf{C}^{Q} &= \mathbf{X}\boldsymbol{W}^{DQ}, \quad (\boldsymbol{W}^{DQ} \in \mathbb{R}^{d_{\text{model}} \times d_c'}), \\
\mathrm{Concat}\big(\mathbf{Q}_1^C, \mathbf{Q}_2^C, \ldots, \mathbf{Q}_h^C\big) = \mathbf{Q}^C &= \mathbf{C}^{Q}\boldsymbol{W}^{UQ}, \quad (\boldsymbol{W}^{UQ} \in \mathbb{R}^{d_c' \times d_h h}), \\
\mathrm{Concat}\big(\mathbf{Q}_1^R, \mathbf{Q}_2^R, \ldots, \mathbf{Q}_h^R\big) = \mathbf{Q}^R &= \mathrm{RoPE}\big(\mathbf{C}^{Q}\boldsymbol{W}^{QR}\big), \quad (\boldsymbol{W}^{QR} \in \mathbb{R}^{d_c' \times d_h^R h}), \\
\mathbf{Q} &= \mathrm{Concat}\big(\mathbf{Q}^C, \mathbf{Q}^R\big).
\end{aligned}$$

Here, $\mathbf{C}^Q \in \mathbb{R}^{T \times d_c'}$ (with $d_c' \ll d_h h$) is the compressed query latent. As above, each $\boldsymbol{W}^{DQ}$, $\boldsymbol{W}^{UQ}$, and $\boldsymbol{W}^{QR}$ connects these lower-dimensional query latents back to h heads of dimension $d_h + d_h^R$.

Given compressed queries, keys, and values, the final attention output for the t-th token is:

$$\begin{aligned}
\mathbf{O}_i &= \mathrm{Softmax}\Big(\frac{\mathbf{Q}_i\mathbf{K}_i^\top}{\sqrt{d_h + d_h^R}}\Big)\, \mathbf{V}_i^C, \\
\mathbf{U} &= \mathrm{Concat}\big(\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_h\big)\boldsymbol{W}^O,
\end{aligned}$$

where $\boldsymbol{W}^O \in \mathbb{R}^{(d_h h) \times d_{\text{model}}}$ is the output projection.

In inference time, $\mathbf{C}^{KV}$ and $\mathbf{K}^R$ can be cached to accelerate decoding. In detail, when RoPE is ignored, the inner product $\mathbf{q}_{t,i}^\top \mathbf{k}_{s,i}$ (where $\mathbf{q}_{t,i}, \mathbf{k}_{s,i} \in \mathbb{R}^d$) of the i-th head between t-th and s-th tokens can be calculated using the hidden state $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$ for t-th token and the cached latent state $\mathbf{c}_s^{KV} \in \mathbb{R}^{d_c}$ for s-th token:

$$\mathbf{q}_{t,i}^\top \mathbf{k}_{s,i} = [(\boldsymbol{W}_i^{UQ})^\top (\boldsymbol{W}_i^{DQ})^\top \mathbf{x}_t]^\top [(\boldsymbol{W}_i^{UK})^\top \mathbf{c}_s^{KV}] = \mathbf{x}_t^\top [\boldsymbol{W}_i^{DQ} \boldsymbol{W}_i^{UQ} (\boldsymbol{W}_i^{UK})^\top] \mathbf{c}_s^{KV}, \tag{2.2}$$

where $\boldsymbol{W}_i^{(\cdot)}$ is the i-th head of the original weight, and $[\boldsymbol{W}_i^{DQ}\boldsymbol{W}_i^{UQ}(\boldsymbol{W}_i^{UK})^\top]$ can be computed previously for faster decoding. However, this process fails when RoPE is considered according to Su (2024). Since RoPE can be considered as multiplication with a block-diagonal matrix $\mathbf{T}_t \in \mathbb{R}^{d_h \times d_h}$ (see Section 2.5), with the property (2.1) that $\mathbf{T}_t\mathbf{T}_s^\top = \mathbf{T}_{t-s}$, then

$$\begin{aligned} \mathbf{q}_{t,i}^\top \mathbf{k}_{s,i} &= [{\mathbf{T}_t}^\top (\boldsymbol{W}_i^{UQ})^\top (\boldsymbol{W}_i^{DQ})^\top \mathbf{x}_t]^\top [{\mathbf{T}_s}^\top (\boldsymbol{W}_i^{UK})^\top \mathbf{c}_s^{KV}] \\ &= \mathbf{x}_t^\top [\boldsymbol{W}_i^{DQ}\boldsymbol{W}_i^{UQ}\mathbf{T}_{t-s}(\boldsymbol{W}_i^{UK})^\top]\mathbf{c}_s^{KV}. \end{aligned} \tag{2.3}$$

Different from (2.2), acceleration by pre-computing $[\boldsymbol{W}_i^{DQ}\boldsymbol{W}_i^{UQ}\mathbf{T}_{t-s}(\boldsymbol{W}_i^{UK})^\top]$ fails since it varies for different (t, s) position pairs. Therefore, MLA adds the additional $\mathbf{k}_t^R$ part with a relatively smaller size for RoPE compatibility. In Section 3.2, we will show that TPA addresses the issue of RoPE-incompatibility by applying tensor product.

3 Tensor Product Attention

In this section, we provide a detailed description of our proposed *Tensor Product Attention* (TPA), which allows *contextual* low-rank factorization for queries, keys, and values. First, we explain how TPA factorizes queries, keys, and values with explicit tensor shapes. Next, we describe how TPA can be integrated into the multi-head attention framework and how it reduces memory consumption in KV caching at inference time. Finally, we show how RoPE can seamlessly integrate with TPA (including a pre-rotated variant).

3.1 Tensor Factorization of Queries, Keys, and Values

Let $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$ for $t = 1, \ldots, T$ be the hidden-state vector corresponding to the t-th token in a sequence of length T. A typical multi-head attention block has h heads, each of dimension d_h, satisfying $d_{\text{model}} = h \times d_h$. Standard attention projects the entire sequence into three tensors, $\mathbf{Q},\ \mathbf{K},\ \mathbf{V}\ \in \mathbb{R}^{T\times h \times d_h}$, where $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{h \times d_h}$ denote the slices for the t-th token.

Contextual Factorization (CF). Instead of forming each head's query, key, or value via a single linear map, TPA factorizes each $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t$ into a sum of (contextual) tensor products whose ranks are R_q, R_k, and R_v, respectively and may differ. Specifically, for each token t, with a small abuse of notation, we define:

$$\mathbf{Q}_t = \frac{1}{R_Q}\sum_{r=1}^{R_Q} \mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathbf{b}_r^Q(\mathbf{x}_t), \qquad \mathbf{a}_r^Q(\mathbf{x}_t) \in \mathbb{R}^h,\ \mathbf{b}_r^Q(\mathbf{x}_t) \in \mathbb{R}^{d_h}, \tag{3.1}$$

$$\mathbf{K}_t = \frac{1}{R_K}\sum_{r=1}^{R_K} \mathbf{a}_r^K(\mathbf{x}_t) \otimes \mathbf{b}_r^K(\mathbf{x}_t), \qquad \mathbf{a}_r^K(\mathbf{x}_t) \in \mathbb{R}^h,\ \mathbf{b}_r^K(\mathbf{x}_t) \in \mathbb{R}^{d_h}, \tag{3.2}$$

$$\mathbf{V}_t = \frac{1}{R_V}\sum_{r=1}^{R_V} \mathbf{a}_r^V(\mathbf{x}_t) \otimes \mathbf{b}_r^V(\mathbf{x}_t), \qquad \mathbf{a}_r^V(\mathbf{x}_t) \in \mathbb{R}^h,\ \mathbf{b}_r^V(\mathbf{x}_t) \in \mathbb{R}^{d_h}. \tag{3.3}$$

Hence, for queries, each tensor product $\mathbf{a}_r^Q(\mathbf{x}_t) \otimes \mathbf{b}_r^Q(\mathbf{x}_t) : \mathbb{R}^h \times \mathbb{R}^{d_h} \to \mathbb{R}^{h \times d_h}$ adds up to form the query slice $\mathbf{Q}_t \in \mathbb{R}^{h \times d_h}$. Similarly, analogous definitions apply to key slice $\mathbf{K}_t$ and value slice $\mathbf{V}_t$.

Latent Factor Maps. Each factor in the tensor product depends on the token's hidden state $\mathbf{x}_t$. For example, for queries, we can write:

$$\mathbf{a}_r^Q(\mathbf{x}_t) = \boldsymbol{W}_r^{a^Q}\,\mathbf{x}_t \in \mathbb{R}^h, \quad \mathbf{b}_r^Q(\mathbf{x}_t) = \boldsymbol{W}_r^{b^Q}\,\mathbf{x}_t \in \mathbb{R}^{d_h},$$

and similarly for keys and values.

One often merges the rank index into a single output dimension. For instance, for queries:

$$\mathbf{a}^Q(\mathbf{x}_t) = \boldsymbol{W}^{a^Q}\,\mathbf{x}_t \in \mathbb{R}^{R_q \cdot h}, \quad \mathbf{b}^Q(\mathbf{x}_t) = \boldsymbol{W}^{b^Q}\,\mathbf{x}_t \in \mathbb{R}^{R_q \cdot d_h},$$

which are then reshaped into $\mathbf{A}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_q \times h}$ and $\mathbf{B}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_q \times d_h}$. Summing over R_q and scaled by $\frac{1}{R_q}$ yields

$$\mathbf{Q}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top\,\mathbf{B}_Q(\mathbf{x}_t) \in \mathbb{R}^{h \times d_h}.$$

Repeating for all tokens reconstitutes $\mathbf{Q} \in \mathbb{R}^{T \times h \times d_h}$. Similar procedures can be applied to obtain $\mathbf{K}$ and $\mathbf{V}$ with ranks R_k and R_v, respectively.

Scaled Dot-Product Attention. Once $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are factorized, multi-head attention proceeds as in standard Transformers. For each head $i \in \{1, \ldots, h\}$:

$$\mathbf{head}_i = \text{Softmax}\Big(\tfrac{1}{\sqrt{d_h}}\, \mathbf{Q}_i\, (\mathbf{K}_i)^\top\Big)\, \mathbf{V}_i, \tag{3.4}$$

where $\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i \in \mathbb{R}^{T \times d_h}$ are the slices along the head dimension. Concatenating these h heads along the last dimension yields an $\mathbb{R}^{T \times (h \cdot d_h)}$ tensor, which is projected back to $\mathbb{R}^{T \times d_{\text{model}}}$ by an output weight matrix $\boldsymbol{W}^O \in \mathbb{R}^{(h \cdot d_h) \times d_{\text{model}}}$:

$$\text{TPA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}\big(\mathbf{head}_1, \ldots, \mathbf{head}_h\big)\boldsymbol{W}^O. \tag{3.5}$$

Parameter Initialization. We initialize the weight matrices $\boldsymbol{W}_r^{a^Q}$, $\boldsymbol{W}_r^{a^K}$, $\boldsymbol{W}_r^{a^V}$, $\boldsymbol{W}_r^{b^Q}$, $\boldsymbol{W}_r^{b^K}$, $\boldsymbol{W}_r^{b^V}$ using Xavier initialization (Glorot & Bengio, 2010). Specifically, each entry of the weight matrix is drawn from a uniform distribution with bounds $[-\sqrt{6/(n_{\text{in}} + n_{\text{out}})}, \sqrt{6/(n_{\text{in}} + n_{\text{out}})}]$, where n_{in} and n_{out} are the input and output dimensions of the respective weight matrices. This initialization strategy helps maintain the variance of activations and gradients across the network.

3.2 RoPE Compatibility and Acceleration

In a typical workflow of adding RoPE to standard multi-head attention, one first computes $\mathbf{Q}_t, \mathbf{K}_s \in \mathbb{R}^{h \times d_h}$ of the t-th token and s-th token and then applies:

$$\mathbf{Q}_t \mapsto \widetilde{\mathbf{Q}_t} = \text{RoPE}_t(\mathbf{Q}_t), \qquad \mathbf{K}_s \mapsto \widetilde{\mathbf{K}_s} = \text{RoPE}_s(\mathbf{K}_s).$$

Direct Integration. A useful optimization is to integrate RoPE directly into the TPA factorization. For example, one can *pre-rotate* the token-dimension factors:

$$\widetilde{\mathbf{B}}_K(\mathbf{x}_t) \;\longleftarrow\; \text{RoPE}_t\big(\mathbf{B}_K(\mathbf{x}_t)\big), \tag{3.6}$$

yielding a *pre-rotated* key representation:

$$\widetilde{\mathbf{K}}_t = \frac{1}{R_K} \sum_{r=1}^{R_K} \mathbf{a}^K_{(r)}(\mathbf{x}_t) \otimes \text{RoPE}_t\big(\mathbf{b}^K_{(s)}(\mathbf{x}_t)\big) = \frac{1}{R_K} \mathbf{A}_K(\mathbf{x}_t)^\top\, \text{RoPE}_t\big(\mathbf{B}_K(\mathbf{x}_t)\big).$$

Thus, each $\mathbf{K}_t$ is already rotated before caching, removing the need for explicit rotation at the decoding time and accelerating autoregressive inference. Depending on hardware and performance requirements, one can also adopt different RoPE integration approaches for training and inference.

Theorem 1 (RoPE's Compatibility with TPA)**.** Let $\mathbf{Q}_t$ be factorized by TPA as

$$\mathbf{Q}_t = \frac{1}{R_Q} \mathbf{A}_Q(\mathbf{x}_t)^\top\, \mathbf{B}_Q(\mathbf{x}_t) \;\in \mathbb{R}^{h \times d_h},$$

where $\mathbf{A}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_Q \times h}$ and $\mathbf{B}_Q(\mathbf{x}_t) \in \mathbb{R}^{R_Q \times d_h}$. Then we have:

$$\text{RoPE}(\mathbf{Q}_t) = \frac{1}{R_Q} \mathbf{A}_Q(\mathbf{x}_t)^\top\, \widetilde{\mathbf{B}}_Q(\mathbf{x}_t), \qquad \text{where } \widetilde{\mathbf{B}}_Q(\mathbf{x}_t) = \text{RoPE}_t\big(\mathbf{B}_Q(\mathbf{x}_t)\big). \tag{3.7}$$

In addition, assume $\mathbf{Q}_t$ and $\mathbf{K}_s$ are factorized by TPA and then rotated by $\text{RoPE}_t, \text{RoPE}_s$. Let $\widetilde{\mathbf{Q}}_t = \text{RoPE}_t(\mathbf{Q}_t)$ and $\widetilde{\mathbf{K}}_s = \text{RoPE}_s(\mathbf{K}_s)$. Then we have

$$\text{RoPE}_{t-s}(\mathbf{Q}_t)\mathbf{K}_s^\top = \widetilde{\mathbf{Q}}_t\, \widetilde{\mathbf{K}}_s^\top,$$

Focusing on individual heads i, the above matrix equality implies:

$$\text{RoPE}_{t-s}\big(\mathbf{q}_{t,i}\big)^\top \mathbf{k}_{s,i} = \widetilde{\mathbf{q}}_{t,i}^\top\, \widetilde{\mathbf{k}}_{s,i}.$$

where $\mathbf{q}_{t,i} \in \mathbb{R}^{d_h}$ is the i-th query head of t-th token, and $\mathbf{k}_{s,i} \in \mathbb{R}^{d_h}$ is the j-th key head of s-th token, and

$$\widetilde{\mathbf{q}}_{t,i} = \text{RoPE}(\mathbf{q}_{t,i}) = \mathbf{T}_t \mathbf{q}_{t,i} \in \mathbb{R}^{d_h}, \quad \widetilde{\mathbf{k}}_{s,i} = \text{RoPE}(\mathbf{k}_{s,i}) = \mathbf{T}_s \mathbf{k}_{s,i} \in \mathbb{R}^{d_h}.$$

Theorem 1 indicates that TPA does not break RoPE's relative translational property. We prove Theorem 1 in Appendix A. In short, RoPE_t acts as a block-diagonal orthogonal transform (i.e., a matrix $\mathbf{T}_t$) on $\mathbf{B}_Q(\mathbf{x}_t)$. Consequently, $\mathbf{A}_Q(\mathbf{x}_t)$ remains unchanged, while each column of $\mathbf{B}_Q(\mathbf{x}_t)$ is rotated appropriately, preserving the TPA structure.

3.3 KV Caching and Memory Reduction

In autoregressive decoding, standard attention caches $\mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{h \times d_h}$ for each past token t. This accumulates to $\mathbb{R}^{T \times h \times d_h}$ for keys and $\mathbb{R}^{T \times h \times d_h}$ for values, i.e., $2\,T\,h\,d_h$ total.

TPA Factorized KV Caching. Instead of storing the full $\mathbf{K}_t$ and $\mathbf{V}_t$, TPA stores only their factorized ranks. Specifically, we keep

$$\mathbf{A}_K(\mathbf{x}_t),\, \widetilde{\mathbf{B}}_K(\mathbf{x}_t) \quad \text{and} \quad \mathbf{A}_V(\mathbf{x}_t),\, \mathbf{B}_V(\mathbf{x}_t),$$

where $\mathbf{A}_K(\mathbf{x}_t) \in \mathbb{R}^{R_K \times h},\ \widetilde{\mathbf{B}}_K(\mathbf{x}_t) \in \mathbb{R}^{R_K \times d_h},\ \mathbf{A}_V(\mathbf{x}_t) \in \mathbb{R}^{R_V \times h},\ \mathbf{B}_V(\mathbf{x}_t) \in \mathbb{R}^{R_V \times d_h}$.

Hence, the memory cost per token is

$$\underbrace{R_K(h + d_h)}_{\text{for K}} + \underbrace{R_V(h + d_h)}_{\text{for V}} = (\,R_K + R_V\,)\,\big(h + d_h\big).$$

Compared to the standard caching cost of $2\,h\,d_h$, the ratio is:

$$\frac{(R_K + R_V)\,(h + d_h)}{2\,h\,d_h}.$$

For large h and d_h (typically $d_h = 64$ or 128), setting $R_K, R_V \ll d_h$ (e.g., 1 or 2) often yields 10× or more reduction.

Table 1: Comparison of different attention mechanisms. Here, R_Q, R_K, and R_V denote the ranks for queries, keys, and values in TPA, respectively. Variants of TPA, such as TPA (KVonly), TPA (Non-contextual A), and TPA (Non-contextual B), are detailed in Section 3.5. For MLA, d_h^R and d_h are the dimensions for RoPE and non-RoPE parts; d_c' and d_c are the dimensions of compressed vectors for query and key-value, respectively.

METHOD	KV CACHE	# PARAMETERS	# QUERY HEADS	# KV HEADS
MHA	$2hd_h$	$4d_{\text{model}}^2$	h	h
MQA	$2d_h$	$(2 + 2/h)d_{\text{model}}^2$	h	1
GQA	$2gd_h$	$(2 + 2g/h)d_{\text{model}}^2$	h	g
MLA	$d_c + d_h^R$	$d_c'(d_{\text{model}} + hd_h + hd_h^R)$ $+d_{\text{model}}d_h^R + d_c(d_{\text{model}} + 2hd_h)$	h	h
TPA	$(R_K + R_V)(h + d_h)$	$d_{\text{model}}(R_Q + R_K + R_V)(h + d_h) + d_{\text{model}}\, hd_h$	h	h
TPA (KVonly)	$(R_K + R_V)(h + d_h)$	$d_{\text{model}}(R_K + R_V)(h + d_h) + 2d_{\text{model}}\, hd_h$	h	h
TPA (Non-contextual A)	$(R_K + R_V)d_h$	$(R_Q + R_K + R_V)(d_{\text{model}}d_h + h) + d_{\text{model}}\, hd_h$	h	h
TPA (Non-contextual B)	$(R_K + R_V)h$	$(R_Q + R_K + R_V)(d_{\text{model}}h + d_h) + d_{\text{model}}\, hd_h$	h	h

3.4 Unifying MHA, MQA, and GQA as Non-contextual TPA

3.4.1 MHA as Non-contextual TPA

Standard multi-head attention (MHA) can be viewed as a specific instance of TPA in which: 1) the rank is set equal to the number of heads; 2) the head dimension factor is non-contextual (i.e., independent of the t-th token embedding $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$); 3) the token dimension factor is a linear function of $\mathbf{x}_t$.

To match MHA with TPA, let $R_Q = R_K = R_V = h$. Focusing on $\mathbf{Q}_t$:

(a) **Non-contextual head factors.** Define

$$\mathbf{a}_i^Q = R_Q \mathbf{e}_i \in \mathbb{R}^h, \quad (\mathbf{e}_i \in \mathbb{R}^h \text{ is the } i\text{-th standard basis vector}), \tag{3.8}$$

so that $\mathbf{e}_i \otimes \cdot$ corresponds to the i-th head of $\mathbf{Q}_t$.

(b) **Contextual token factors.** Define

$$\mathbf{b}_i^Q(\mathbf{x}_t) = (\boldsymbol{W}_i^Q)^\top \mathbf{x}_t \in \mathbb{R}^{d_h}, \tag{3.9}$$

where $\boldsymbol{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_h}$ is the per-head query projection defined before, hence $\mathbf{b}_i^Q(\mathbf{x}_t)$ dependent on $\mathbf{x}_t$.

Substituting (3.8)–(3.9) into (3.1) gives:

$$\mathbf{Q}_t = \sum_{i=1}^{h} \Big[\mathbf{e}_i \otimes \big((\boldsymbol{W}_i^Q)^\top \, \mathbf{x}_t \big) \Big] \in \mathbb{R}^{h \times d_h}. \tag{3.10}$$

Each term $\mathbf{e}_i \otimes \big((\boldsymbol{W}_i^Q)^\top \mathbf{x}_t\big)$ in (3.10) contributes only to the i-th row, reconstituting the usual MHA form of $\mathbf{Q}_t$. Analogous constructions hold for $\mathbf{K}_t$ and $\mathbf{V}_t$ using $\boldsymbol{W}_i^K, \boldsymbol{W}_i^V$. Thus, *MHA is a non-contextual, full-rank variant of TPA*.

TPA with Non-contextual A. More broadly, TPA can use non-contextual head-dimension factors $\mathbf{a}_r^Q, \mathbf{a}_r^K, \mathbf{a}_r^V \in \mathbb{R}^h$ (i.e., independent of $\mathbf{x}_t$), while allowing $\mathbf{b}_r^Q(\mathbf{x}_t), \mathbf{b}_r^K(\mathbf{x}_t), \mathbf{b}_r^V(\mathbf{x}_t)$ to remain context-dependent. Then, for keys:

$$\mathbf{K}_t = \frac{1}{R_K} \sum_{r=1}^{R_K} \mathbf{a}_r^K \otimes \mathbf{b}_r^K(\mathbf{x}_t),$$

and similarly for queries/values. This reduces per-token computations and can be effective when head-dimension relationships are relatively stable across all tokens.

MQA and GQA as Non-Contextual TPA. Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped Query Attention (GQA) (Ainslie et al., 2023) also emerge naturally from TPA by restricting the head-dimension factors to be non-contextual *and* low-rank:

- **MQA as Rank-1 TPA.** In MQA, all heads share a *single* set of keys/values, corresponding to $R_K = R_V = 1$ along the head dimension. Concretely,

  $$\mathbf{K}_t = (1, \ldots, 1)^\top \otimes \mathbf{b}^K(\mathbf{x}_t), \quad \mathbf{V}_t = (1, \ldots, 1)^\top \otimes \mathbf{b}^V(\mathbf{x}_t),$$

  forces every head to use the same $\mathbf{K}_t, \mathbf{V}_t$. Each head retains a distinct query projection, matching the MQA design.
- **GQA as Grouped Rank-1 TPA.** GQA partitions h heads into G groups, each sharing keys/values within that group. In TPA form, each group g has a dedicated non-contextual factor pair $\mathbf{a}_g^K, \mathbf{a}_g^V \in \mathbb{R}^h$, which acts as a "mask" for the heads in that group. Varying G from 1 to h interpolates from MQA to standard MHA.

Hence, by constraining TPA's head-dimension factors to be constant masks (one for MQA; multiple for GQA), these popular variants are recovered as special cases.

3.5 Other Variants of TPA

TPA with Non-contextual B. Conversely, one may fix the token-dimension factors $\mathbf{b}_r^Q, \mathbf{b}_r^K, \mathbf{b}_r^V \in \mathbb{R}^{d_h}$ as learned parameters, while allowing $\mathbf{a}_r^Q(\mathbf{x}_t), \mathbf{a}_r^K(\mathbf{x}_t), \mathbf{a}_r^V(\mathbf{x}_t)$ to adapt to $\mathbf{x}_t$. For keys:

$$\mathbf{K}_t = \frac{1}{R_K} \sum_{r=1}^{R_K} \mathbf{a}_r^K(\mathbf{x}_t) \otimes \mathbf{b}_r^K,$$

and similarly for keys/values. This arrangement is effective if the token-dimension structure remains mostly uniform across the sequence, while the head-dimension factors capture context.

TPA KV Only. One can preserve a standard query mapping,

$$\mathbf{Q}_t = \boldsymbol{W}^Q \mathbf{x}_t \in \mathbb{R}^{h \times d_h},$$

and factorize only the keys and values. This leaves the query projection as the original linear transformation while reducing memory usage via factorized KV caching.

TPA KV with Shared B. Another variant is to share the token-dimension factors of keys and values:

$$\mathbf{b}_r^K(\mathbf{x}_t) = \mathbf{b}_r^V(\mathbf{x}_t),$$

lowering parameter counts and the KV cache footprint. While it constrains $\mathbf{K}$ and $\mathbf{V}$ to be formed from the same token basis, it can still perform well and provide additional memory savings.

Nonlinear Head Factors. Rather than applying purely linear mappings to the head-dimension factors $\mathbf{a}_r^Q, \mathbf{a}_r^K, \mathbf{a}_r^V$, one may introduce element-wise nonlinearities such as $\sigma(\cdot)$ or $\mathrm{softmax}(\cdot)$. This effectively yields a *Mixture of Heads Attention* (MoH Attention), where each component becomes a learned mixture weight modulated by the nonlinearity.

Discussion. These variants illustrate TPA's versatility in balancing memory cost, computational overhead, and representation power. By choosing which dimensions (heads or tokens) remain contextual and adjusting ranks (R_Q, R_K, R_V), TPA unifies multiple existing attention mechanisms—such as MHA, MQA, and GQA—under one framework, while potentially reducing the KV cache size by an order of magnitude during autoregressive inference.

3.6 Model Architectures

We propose a new architecture called **T**ensor Produc**T** A**TT**en**T**ion **T**ransformer (T6), which uses our *Tensor Product Attention* (TPA) in place of standard MHA (multi-head attention) or GQA (grouped-query attention). Building upon the query, key, and value tensors $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times h \times d_h}$ defined in Section 3.1, T6 utilize the overall architecture of LLaMA (Touvron et al., 2023) while changing the self-attention block to our TPA-based version. The feed-forward network (FFN) adopts a SwiGLU layer, as in (Shazeer, 2020; Touvron et al., 2023).

TPA QKV Factorization. Let each token's hidden-state vector be $\mathbf{x}_t \in \mathbb{R}^{d_{\text{model}}}$, and we follow Section 3.1 to project the entire sequence into three tensors $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{T \times h \times d_h}$, where $\mathbf{Q}_t, \mathbf{K}_t, \mathbf{V}_t \in \mathbb{R}^{h \times d_h}$ denote the slices for the t-th token. The factor components $\mathbf{a}_r^Q(\mathbf{x}_t), \mathbf{b}_r^Q(\mathbf{x}_t), \mathbf{a}_r^K(\mathbf{x}_t), \mathbf{b}_r^K(\mathbf{x}_t), \mathbf{a}_r^V(\mathbf{x}_t), \mathbf{b}_r^V(\mathbf{x}_t)$ are produced by linear transformations on $\mathbf{x}_t$. For instance, letting $\boldsymbol{W}_r^{a^Q} \in \mathbb{R}^{h \times d_{\text{model}}}$ and $\boldsymbol{W}_r^{b^Q} \in \mathbb{R}^{d_h \times d_{\text{model}}}$, we have:

$$\mathbf{a}_r^Q(\mathbf{x}_t) = \boldsymbol{W}_r^{a^Q} \mathbf{x}_t, \quad \mathbf{b}_r^Q(\mathbf{x}_t) = \boldsymbol{W}_r^{b^Q} \mathbf{x}_t.$$

In practice, we merge all ranks r into a single dimension of the output, reshape, and sum over rank indices; see Section 3.1 for details. The factorization for K and V follows the same pattern.

Rotary Positional Embedding (RoPE). As discussed in Section 3.2, RoPE (Su et al., 2024b) is applied to the $\mathbf{Q}$ and $\mathbf{K}$. Within TPA, we *pre-rotate* the factor $\mathbf{b}_t^Q(\mathbf{x}_t)$ and $\mathbf{b}_s^K(\mathbf{x}_s)$ directly, so that each $\mathbf{K}_s$ is already rotated prior to caching, see (3.6) and Theorem 1.

Attention Step and Output Projection. Once we have $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ factorized per token with RoPE applied on $\mathbf{Q}$ and $\mathbf{K}$, the attention step proceeds for each head $i \in \{1, \ldots, h\}$ using (3.4). Finally, concatenating these h heads and then projecting them back using an output weight matrix gives the final attention result, as shown in (3.5).

SwiGLU Feed-Forward Network. Following Shazeer (2020); Touvron et al. (2023), our T6 uses a SwiGLU-based Feed-Forward Network (FFN):

$$\mathrm{FFN}(\mathbf{x}) = \big[\sigma(\mathbf{x}\,\boldsymbol{W}_1) \odot (\mathbf{x}\,\boldsymbol{W}_2)\big]\,\boldsymbol{W}_3,$$

where σ is the SiLU (a.k.a., swish) nonlinearity, $\odot$ is element-wise product, and $\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{W}_3$ are learnable parameters. Note that other activation functions can also be used.

Overall T6 Block Structure. Putting everything together, one T6 block consists of:

$$\begin{aligned} \mathbf{x} &\leftarrow \mathbf{x} + \mathrm{TPA}\big(\mathrm{RMSNorm}(\mathbf{x})\big), \\ \mathbf{x} &\leftarrow \mathbf{x} + \mathrm{SwiGLU\text{-}FFN}\big(\mathrm{RMSNorm}(\mathbf{x})\big). \end{aligned}$$

We place norm layers (e.g., RMSNorm) before each sub-layer. Stacking L such blocks yields a T6 model architecture with L layers.

4 Experiments

4.1 Language Modeling Tasks

All experiments reported in this paper are implemented on the `nanoGPT` code base (Karpathy, 2022), using the FineWeb-Edu 100B dataset (Lozhkov et al., 2024). The dataset contains 100 billion tokens for training and 0.1 billion tokens for validation. We compare T6 against the baseline Llama architecture (Touvron et al., 2023) with SwiGLU activation (Shazeer, 2020) and RoPE embeddings (Su et al., 2024a), as well as Llama variants that replace Multi-Head Attention (MHA; Vaswani et al., 2017) with Multi-Query Attention (MQA; Shazeer, 2019), Grouped Query Attention (GQA; Ainslie et al., 2023), or Multi-head Latent Attention (MLA; Liu et al., 2024a). In our experiments, the number of heads h is adjusted for each attention mechanism to ensure that all attention mechanisms have the same number of parameters as the standard Multi-Head Attention (MHA), which has $4d_{\text{model}}^2$ parameters per attention layer. We train models at four scales: *small* (124M parameters), *medium* (353M), and *large* (773M). Details on architecture hyperparameters and training hardware appear in Appendix B.1.

Training Setup. We follow the `nanoGPT` training configuration. In particular, we use the AdamW (Loshchilov, 2017) optimizer with $(\beta_1, \beta_2) = (0.9, 0.95)$, a weight decay of 0.1, and gradient clipping at 1.0. We follow the same setting as `nanoGPT` that the learning rate is managed by a cosine annealing scheduler (Loshchilov & Hutter, 2016) with 2,000 warmup steps and a (total) global batch size of 480. For the *small*, *medium*, and *large* models, we set maximum learning rates of 6×10^{-4}, 3×10^{-4}, and 2×10^{-4} (respectively), and minimum learning rates of 3×10^{-5}, 3×10^{-5}, and 1×10^{-5} (respectively).

Training & Validation Curves. Figures 2 and 3 compare training and validation loss curves for the *large* (773M) and *medium* (353M) models on FineWeb-Edu-100B. Overall, **TPA** (red curves) and its simpler variant **TPA-KVonly** (pink curves) converge as fast as or faster than the baselines (MHA, MQA, GQA, MLA) while also achieving visibly lower final losses. For instance, in Figure 2(b), TPA and TPA-KVonly remain below the MHA baseline in terms of validation loss at nearly all training stages. Meanwhile, Multi-Head Latent Attention (MLA) (Liu et al., 2024a) (blue curves) generally trains more slowly and yields higher losses.

Validation Perplexity. Figure 4 shows the validation perplexities of the *medium*- and *large*-scale models. Mirroring the loss curves, **TPA** and **TPA-KVonly** steadily outperform MHA, MQA, GQA, and MLA over the course of training. By the end of pretraining (around 49B tokens), TPA-based approaches achieve the lowest perplexities in most configurations.

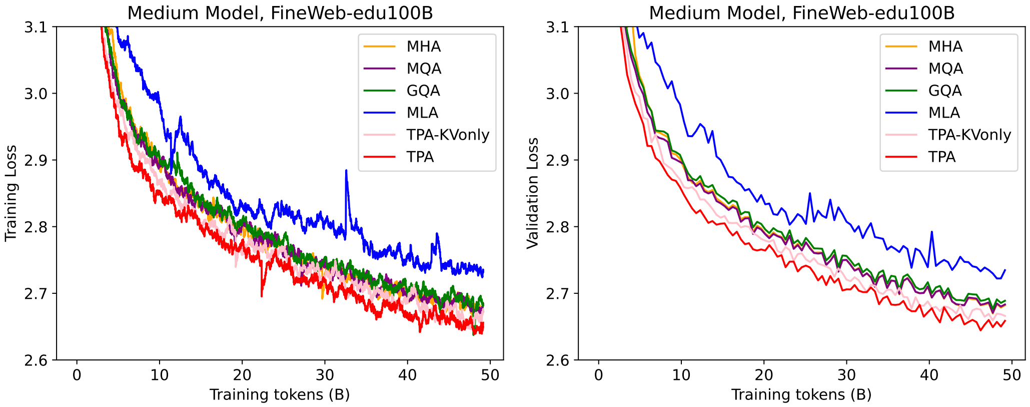
Downstream Evaluation. We evaluate zero-shot and two-shot performance on standard benchmarks, including ARC (Yadav et al., 2019), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OBQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), WinoGrande (Sakaguchi et al., 2020) and MMLU (Hendrycks et al., 2021), using the `lm-evaluation-harness` codebase (Gao et al., 2024). For ARC-E, ARC-C, HellaSwag, OBQA, PIQA, and SciQ, we report accuracy norm; for other tasks, we report standard accuracy. Tables 8–9 in the appendix present results for *small* models; Tables 2–3 for *medium* models; Tables 4–5 for *large* models;

For the *medium*-size (353M) models (Tables 2–3), **TPA** generally ties or outperforms all competing methods, achieving, for example, an average of 51.41% in zero-shot mode versus MHA's 50.11%, MQA's 50.44%, and MLA's 48.96%. When given two-shot prompts, TPA again leads with 53.12% average accuracy. A similar trend appears for the *large*-size (773M) models (Tables 4–5), where **TPA-KVonly** attains the highest average (53.52% zero-shot, 55.33% two-shot), closely followed by full TPA.

Our experiments confirm that **TPA** consistently matches or exceeds the performance of established attention mechanisms (MHA, MQA, GQA, MLA) across *medium* and *large* model scales. The fully factorized TPA excels on mid-scale models, while **TPA-KVonly** can rival or surpass it at larger scales. In both cases, factorizing the attention activations shrinks autoregressive KV cache requirements by up to $5\times$–$10\times$, thus enabling much longer context windows under fixed memory budgets. In summary, tensor product attention provides a flexible, memory-efficient alternative to standard multi-head attention, advancing the scalability of modern language models.

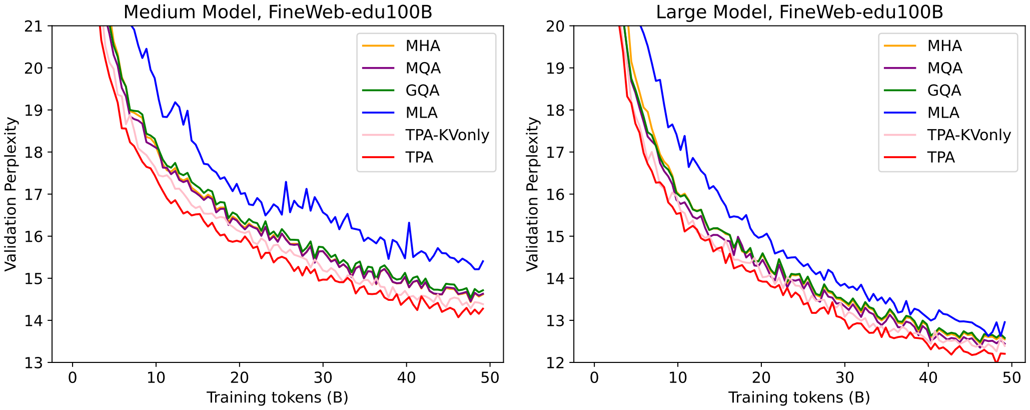
5 Related Work

Transformers and Attention. As a sequence-to-sequence architecture Transformer (Vaswani et al., 2017) introduced Multi-Head Attention (MHA), enabling more effective capture of long-range dependencies. Subsequent work has explored a variety of attention mechanisms aimed at improving scalability and efficiency, including sparse patterns (Child et al., 2019; Shi et al., 2023; Han et al., 2024; Liang et al., 2024a; Li et al., 2024; Liang et al., 2024b), kernel-based projections (Choromanski et al., 2021), and linearized transformers (Tsai et al., 2019; Katharopoulos et al., 2020; Schlag



(a) Training Loss (b) Validation Loss

Figure 3: The training loss and validation loss of medium-size (353M) models with different attention mechanisms on the FineWeb-Edu 100B dataset.



(a) Validation Perplexity of Medium Models (b) Validation Perplexity of Large Models

Figure 4: The validation perplexity of medium-size (353M) models and large-size (773M) models with different attention mechanisms on the FineWeb-Edu 100B dataset.

Table 2: The evaluation results of medium models with different attention mechanisms pretrained using the FineWeb-Edu 100B dataset (0-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	56.52	29.27	58.84	44.06	35.00	68.44	51.07	25.35	76.40	49.44
MQA	55.68	28.24	60.86	44.17	35.20	68.66	52.72	25.14	72.90	49.29
GQA	54.88	29.61	56.36	43.77	35.20	68.82	52.57	**25.41**	74.80	49.05
MLA	55.30	29.27	58.96	41.92	**35.40**	67.25	51.78	25.20	75.60	48.96
TPA-KVonly	57.11	30.03	**61.25**	44.83	34.60	69.04	**54.54**	23.35	74.60	49.93
TPA	**59.30**	**31.91**	60.98	**45.57**	34.60	**69.48**	53.91	24.93	**77.20**	**50.88**

et al., 2021; Zhang et al., 2023b; Sun et al., 2023; Zhang et al., 2024). To decrease memory usage and circumvent the limitation of memory bandwidth in training, Shazeer (2019) proposed Multi-Query Attention (MQA) where multiple query heads share the same key head and value head. To tackle with the issue of quality degradation and instability in training, Grouped-Query Attention (GQA) (Ainslie et al., 2023) divides queries into several groups, and each group of queries shares a single key head and value head. Recently, DeepSeek-V2 (Liu et al., 2024a) applied multihead latent attention (MLA) to achieve better performance than MHA while reducing KV cache in inference time by sharing the same low-rank representation of key and value. In comparison to the approaches above, TPA applied a low-rank tensor product to compute the queries, keys, and values where the

Table 3: The evaluation results of medium models with different attention mechanisms pre-trained using the FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	64.44	32.85	59.05	44.18	33.20	68.72	50.12	26.01	87.40	49.44
MQA	64.27	32.94	57.71	44.36	31.80	68.01	51.70	25.99	86.00	49.29
GQA	61.70	32.17	52.81	43.99	33.80	68.50	53.35	24.44	86.40	50.80
MLA	62.75	30.80	**59.17**	42.02	**34.80**	67.08	52.41	**26.11**	84.80	51.10
TPA-KVonly	65.99	33.70	57.49	44.47	34.20	**69.53**	53.28	24.23	86.50	49.93
TPA	**66.54**	**34.47**	58.96	**45.35**	33.00	69.21	**53.99**	24.51	**91.30**	**53.04**

Table 4: The evaluation results of large models with different attention mechanisms pre-trained using the FineWeb-Edu 100B dataset (0-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	59.93	33.62	61.93	50.63	36.00	71.06	55.41	22.87	**81.20**	52.52
MQA	60.73	33.62	57.34	50.09	37.00	69.97	55.49	25.30	79.60	52.13
GQA	61.66	34.30	58.72	49.85	**38.40**	71.16	53.75	25.23	77.60	52.30
MLA	60.73	31.57	**61.74**	48.96	35.40	69.59	55.09	**26.39**	76.70	51.80
TPA-KVonly	**63.26**	34.13	**61.96**	50.66	37.20	**72.09**	55.25	26.06	81.10	**53.52**
TPA	63.22	**35.58**	60.03	**51.26**	36.80	71.44	**55.56**	24.77	79.60	53.10

Table 5: The evaluation results of large models with different attention mechanisms pre-trained using the FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSwag = HellaSwag, WG = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSwag	OBQA	PIQA	WG	MMLU	SciQ	Avg.
MHA	67.85	36.35	59.82	50.22	35.00	70.67	53.35	23.92	**91.10**	54.25
MQA	68.86	36.09	53.79	50.50	37.00	70.89	54.70	25.01	88.00	53.87
GQA	69.15	36.09	58.84	50.29	36.20	70.73	54.22	**26.08**	90.00	54.62
MLA	68.56	35.41	**60.12**	49.18	**38.00**	69.21	**55.25**	25.29	88.20	54.36
TPA-KVonly	**71.34**	**37.71**	59.76	51.10	36.00	**71.49**	54.62	25.83	90.10	**55.33**
TPA	70.41	**37.71**	60.06	**51.30**	34.00	71.06	54.54	25.79	90.30	55.02

cached representations of keys and values are much smaller than those in MHA, achieving better reduction on memory assumption of KV cache in inference time.

Low-Rank Factorizations. Low-rank approximations have been applied to compress model parameters and reduce complexity including LoRA (Hu et al., 2022), which factorizes weight updates during fine-tuning, and its derivatives for other training scenarios such as efficient pretraining (ReLoRA (Lialin et al., 2023), MoRA (Jiang et al., 2024)), long-context training (LongLoRA (Chen et al., 2024), SinkLoRA (Zhang, 2024)), as well as continual training (InfLoRA (Liang & Li, 2024), GS-LoRA (Zhao et al., 2024), I-LoRA (Ren et al., 2024)). These approaches typically produce static low-rank expansions that do not explicitly depend on the input context. And Malladi et al. (2023); Zeng & Lee (2024) provided theoretical proof of the expressiveness of low-rank approximation. For the initialization of factorization matrices, OLoRA (Büyükakyüz, 2024) applied QR-decomposition of pretrained weight to achieve better performance of language models while LoLDU (Shi et al., 2024) used LDU-decomposition to accelerate training of LoRA. Moreover, AdaLoRA (Zhang et al., 2023a) utilized Singular Value Decomposition (SVD) of the pretrained weight and introduced importance score for each parameter as a measurement to achieve dynamic adjustment of rank. TPA, by contrast, constructs Q, K, and V as contextually factorized tensors, enabling dynamic adaptation.

KV Cache Optimization. During the inference time of Transformers, key and value tensors of the previous tokens are repeatedly computed due to their auto-regressive nature. To enhance efficiency, firstly proposed by Ott et al. (2019), these tensors can be cached in memory for future decoding, referred to as the KV cache. However, the KV cache requires additional memory usage and may add to more latencies due to the bandwidth limitation (Adnan et al., 2024). Therefore, previous studies have explored diverse approaches to mitigate these issues, including KV cache eviction to discard less significant tokens (Zhang et al., 2023c; Xiao et al., 2024; Cai et al., 2024; Adnan et al., 2024), dynamic sparse attention among selected keys and values (Ribar et al., 2024; Tang et al., 2024; Singhania et al., 2024), KV cache offloading to CPU (He & Zhai, 2024; Lee et al., 2024; Sun et al., 2024), as well as quantization of KV cache (Xiao et al., 2023; Liu et al., 2024c; Hooper et al., 2024). Besides these methods, it is also effective to reduce the amount of KV cache for each token, by approaches such as reducing the number of KV heads (Ren et al., 2024; Ainslie et al., 2023), cross-layer KV re-usage (Xiao et al., 2019; Mu et al., 2024; Wu et al., 2024), and low-rank KV representation (Saxena et al., 2024). Different from the methods above, TPA reduces the size of the KV cache by using tensor-decomposed keys and values.

6 Conclusion

We introduced *Tensor Product Attention* (TPA), which factorizes query, key, and value matrices into rank-R tensor products dependent on the token's hidden state. Storing only the factorized key/value components during autoregressive decoding substantially decreases the kv memory size with improved performance compared with MHA, MQA, GQA, and MLA. The approach is fully compatible with RoPE (and can store pre-rotated keys). Variants of TPA include factorizing only the key/value or sharing basis vectors across tokens. Overall, TPA offers a powerful mechanism for compressing KV storage while improving the model performance, thereby enabling longer sequence contexts under constrained memory.

References

Muhammad Adnan, Akhil Arunkumar, Gaurav Jain, Prashant Nair, Ilya Soloveychik, and Purushotham Kamath. Keyformer: Kv cache reduction through key tokens selection for efficient generative inference. *Proceedings of Machine Learning and Systems*, 6:114–127, 2024.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: training generalized multi-query transformer models from multi-head checkpoints. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 4895–4901. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.298. URL `https://doi.org/10.18653/v1/2023.emnlp-main.298`.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 7432–7439. AAAI Press, 2020.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*, 2023.

Kerim Büyükakyüz. Olora: Orthonormal low-rank adaptation of large language models. *arXiv preprint arXiv:2406.01775*, 2024.

Zefan Cai, Yichi Zhang, Bofei Gao, Yuliang Liu, Tianyu Liu, Keming Lu, Wayne Xiong, Yue Dong, Baobao Chang, Junjie Hu, et al. Pyramidkv: Dynamic kv cache compression based on pyramidal information funneling. *arXiv preprint arXiv:2406.02069*, 2024.

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 *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Krzysztof Marcin Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Quincy Davis, Afroz Mohiuddin, Lukasz Kaiser, David Benjamin Belanger, Lucy J. Colwell, and Adrian Weller. Rethinking attention with performers. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *J. Mach. Learn. Res.*, 24:240:1–240:113, 2023.

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 Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 2924–2936. Association for Computational Linguistics, 2019.

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, 07 2024. URL `https://zenodo.org/records/12608602`.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256. JMLR Workshop and Conference Proceedings, 2010.

Insu Han, Rajesh Jayaram, Amin Karbasi, Vahab Mirrokni, David P. Woodruff, and Amir Zandieh. Hyperattention: Long-context attention in near-linear time. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=Eh0Od2BJIM`.

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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun Sophia Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *arXiv preprint arXiv:2401.18079*, 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 *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*, 2022.

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*, 2024.

Andrej Karpathy. NanoGPT. `https://github.com/karpathy/nanoGPT`, 2022.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International conference on machine learning*, pp. 5156–5165. PMLR, 2020.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. {InfiniGen}: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 155–172, 2024.

Xiaoyu Li, Yingyu Liang, Zhenmei Shi, and Zhao Song. A tighter complexity analysis of sparsegpt. *arXiv preprint arXiv:2408.12151*, 2024.

Vladislav Lialin, Sherin Muckatira, Namrata Shivagunde, and Anna Rumshisky. Relora: High-rank training through low-rank updates. In *The Twelfth International Conference on Learning Representations*, 2023.

Yan-Shuo Liang and Wu-Jun Li. Inflora: Interference-free low-rank adaptation for continual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23638–23647, 2024.

Yingyu Liang, Heshan Liu, Zhenmei Shi, Zhao Song, Zhuoyan Xu, and Junze Yin. Conv-basis: A new paradigm for efficient attention inference and gradient computation in transformers. *arXiv preprint arXiv:2405.05219*, 2024a.

Yingyu Liang, Jiangxuan Long, Zhenmei Shi, Zhao Song, and Yufa Zhou. Beyond linear approximations: A novel pruning approach for attention matrix. *arXiv preprint arXiv:2410.11261*, 2024b.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.

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 *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024c.

I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

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

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL `https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu`.

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*, pp. 23610–23641. PMLR, 2023.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (eds.), *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pp. 2381–2391. Association for Computational Linguistics, 2018. doi: 10.18653/V1/D18-1260. URL `https://doi.org/10.18653/v1/d18-1260`.

Yongyu Mu, Yuzhang Wu, Yuchun Fan, Chenglong Wang, Hengyu Li, Qiaozhi He, Murun Yang, Tong Xiao, and Jingbo Zhu. Cross-layer attention sharing for large language models. *arXiv preprint arXiv:2408.01890*, 2024.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In Waleed Ammar, Annie Louis, and Nasrin Mostafazadeh (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Demonstrations*, pp. 48–53. Association for Computational Linguistics, 2019.

Weijieying Ren, Xinlong Li, Lei Wang, Tianxiang Zhao, and Wei Qin. Analyzing and reducing catastrophic forgetting in parameter efficient tuning. *arXiv preprint arXiv:2402.18865*, 2024.

Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient LLM inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pp. 8732–8740. AAAI Press, 2020.

Utkarsh Saxena, Gobinda Saha, Sakshi Choudhary, and Kaushik Roy. Eigen attention: Attention in low-rank space for KV cache compression. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pp. 15332–15344. Association for Computational Linguistics, 2024.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pp. 9355–9366. PMLR, 2021.

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

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

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.

Zhenmei Shi, Jiefeng Chen, Kunyang Li, Jayaram Raghuram, Xi Wu, Yingyu Liang, and Somesh Jha. The trade-off between universality and label efficiency of representations from contrastive learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*, 2023.

Prajwal Singhania, Siddharth Singh, Shwai He, Soheil Feizi, and Abhinav Bhatele. Loki: Low-rank keys for efficient sparse attention. *arXiv preprint arXiv:2406.02542*, 2024.

Jianlin Su. The extreme pull between cache and effect: From MHA, MQA, GQA to MLA. `https://spaces.ac.cn/archives/10091`, May 2024.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024a.

Jianlin Su, Murtadha H. M. Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024b.

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. *arXiv preprint arXiv:2410.21465*, 2024.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models. *arXiv preprint arXiv:2307.08621*, 2023.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. QUEST: query-aware sparsity for efficient long-context LLM inference. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4344–4353, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

You Wu, Haoyi Wu, and Kewei Tu. A systematic study of cross-layer kv sharing for efficient llm inference. *arXiv preprint arXiv:2410.14442*, 2024.

Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International Conference on Machine Learning*, pp. 38087–38099. PMLR, 2023.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

Tong Xiao, Yinqiao Li, Jingbo Zhu, Zhengtao Yu, and Tongran Liu. Sharing attention weights for fast transformer. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 5292–5298. ijcai.org, 2019.

Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pp. 2578–2589. Association for Computational Linguistics, 2019. doi: 10.18653/V1/D19-1260. URL `https://doi.org/10.18653/v1/D19-1260`.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez (eds.), *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pp. 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1472. URL `https://doi.org/10.18653/v1/p19-1472`.

Yuchen Zeng and Kangwook Lee. The expressive power of low-rank adaptation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 2024.

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

Michael Zhang, Kush Bhatia, Hermann Kumbong, and Christopher Ré. The hedgehog & the porcupine: Expressive linear attentions with softmax mimicry. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*, 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 *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023a.

Ruiqi Zhang, Spencer Frei, and Peter L Bartlett. Trained transformers learn linear models in-context. *arXiv preprint arXiv:2306.09927*, 2023b.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023c.

Hongbo Zhao, Bolin Ni, Junsong Fan, Yuxi Wang, Yuntao Chen, Gaofeng Meng, and Zhaoxiang Zhang. Continual forgetting for pre-trained vision models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 28631–28642, 2024.

Appendix

A Proofs of Theorems

Proof of Theorem 1.

Proof. Because RoPE is a linear orthogonal transform, we can write

$$\widetilde{\mathbf{Q}}_t = \mathbf{Q}_t \, \mathbf{T}_t = \frac{1}{R_Q}\big(\mathbf{A}_Q(\mathbf{x}_t)^\top \, \mathbf{B}_Q(\mathbf{x}_t)\big) \, \mathbf{T}_t = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \big(\mathbf{B}_Q(\mathbf{x}_t) \, \mathbf{T}_t\big),$$

where $\mathbf{T}_t$ is the block-diagonal matrix encoding RoPE. This allows us to define

$$\widetilde{\mathbf{B}}_Q(\mathbf{x}_t) = \mathbf{B}_Q(\mathbf{x}_t) \, \mathbf{T}_t,$$

thereby obtaining

$$\mathrm{RoPE}(\mathbf{Q}_t) = \frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \widetilde{\mathbf{B}}_Q(\mathbf{x}_t).$$

Similarly, for the key tensor $\mathbf{K}_s$, we have

$$\widetilde{\mathbf{K}}_s = \mathbf{K}_s \, \mathbf{T}_s = \frac{1}{R_K}\big(\mathbf{A}_K(\mathbf{x}_s)^\top \, \mathbf{B}_K(\mathbf{x}_s)\big) \, \mathbf{T}_s = \frac{1}{R_K}\mathbf{A}_K(\mathbf{x}_s)^\top \big(\mathbf{B}_K(\mathbf{x}_s) \, \mathbf{T}_s\big),$$

which defines

$$\widetilde{\mathbf{B}}_K(\mathbf{x}_s) = \mathbf{B}_K(\mathbf{x}_s) \, \mathbf{T}_s,$$

and thus

$$\mathrm{RoPE}(\mathbf{K}_s) = \frac{1}{R_K}\mathbf{A}_K(\mathbf{x}_s)^\top \widetilde{\mathbf{B}}_K(\mathbf{x}_s).$$

Now, consider the product of the rotated queries and keys:

$$\begin{aligned}\widetilde{\mathbf{Q}}_t \, \widetilde{\mathbf{K}}_s^\top &= \frac{1}{R_Q R_K}\Big(\mathbf{A}_Q(\mathbf{x}_t)^\top \widetilde{\mathbf{B}}_Q(\mathbf{x}_t)\Big)\Big(\mathbf{A}_K(\mathbf{x}_s)^\top \widetilde{\mathbf{B}}_K(\mathbf{x}_s)\Big)^\top \\ &= \frac{1}{R_Q R_K}\mathbf{A}_Q(\mathbf{x}_t)^\top \widetilde{\mathbf{B}}_Q(\mathbf{x}_t) \widetilde{\mathbf{B}}_K(\mathbf{x}_s)^\top \mathbf{A}_K(\mathbf{x}_s),\end{aligned}$$

Since $\mathbf{T}_t$ and $\mathbf{T}_s$ encode positional rotations, the product $\mathbf{T}_t\mathbf{T}_s^\top$ corresponds to a relative rotation $\mathbf{T}_{t-s}$. Therefore, we can express the above as

$$\begin{aligned}\widetilde{\mathbf{Q}}_t \, \widetilde{\mathbf{K}}_s^\top &= \frac{1}{R_Q R_K}\mathbf{A}_Q(\mathbf{x}_t)^\top \left(\mathbf{B}_Q(\mathbf{x}_t)\mathbf{T}_t\mathbf{T}_s^\top \mathbf{B}_K(\mathbf{x}_s)^\top\right) \mathbf{A}_K(\mathbf{x}_s) \\ &= \frac{1}{R_Q R_K}\mathbf{A}_Q(\mathbf{x}_t)^\top \left(\mathbf{B}_Q(\mathbf{x}_t)\mathbf{T}_{t-s}\mathbf{B}_K(\mathbf{x}_s)^\top\right) \mathbf{A}_K(\mathbf{x}_s) \\ &= \frac{1}{R_Q R_K}\mathbf{A}_Q(\mathbf{x}_t)^\top \left(\mathbf{B}_Q(\mathbf{x}_t)\mathbf{T}_{t-s}\right) \left(\mathbf{B}_K(\mathbf{x}_s)^\top \mathbf{A}_K(\mathbf{x}_s)\right) \\ &= \left(\frac{1}{R_Q}\mathbf{A}_Q(\mathbf{x}_t)^\top \mathbf{B}_Q(\mathbf{x}_t)\mathbf{T}_{t-s}\right)\left(\frac{1}{R_K}\mathbf{A}_K(\mathbf{x}_s)^\top \mathbf{B}_K(\mathbf{x}_s)\right)^\top,\end{aligned}$$

This shows that

$$\mathrm{RoPE}_{t-s}(\mathbf{Q}_t)\mathbf{K}_s^\top = \widetilde{\mathbf{Q}}_t \, \widetilde{\mathbf{K}}_s^\top,$$

Focusing on individual heads i, the above matrix equality implies:

$$\mathrm{RoPE}_{t-s}(\mathbf{q}_{t,i})^\top \mathbf{k}_{s,i} = \widetilde{\mathbf{q}}_{t,i}^\top \widetilde{\mathbf{k}}_{s,i},$$

where

$$\widetilde{\mathbf{q}}_{t,i} = \mathrm{RoPE}(\mathbf{q}_{t,i}) = \mathbf{T}_t \mathbf{q}_{t,i} \in \mathbb{R}^{d_h}, \quad \widetilde{\mathbf{k}}_{s,i} = \mathrm{RoPE}(\mathbf{k}_{s,i}) = \mathbf{T}_s \mathbf{k}_{s,i} \in \mathbb{R}^{d_h}.$$

This equality confirms that the relative positional encoding between queries and keys is preserved under TPA's factorization and RoPE's rotation. Thus, TPA maintains compatibility with RoPE. This completes the proof of Theorem 1. □

B More on Experiments

B.1 Experimental Settings

We list the main architecture hyper-parameters and training devices in Table 6. We fix $d_h = 64$ for all the models. Moreover, we fix the number of KV heads with 2 for GQA models; $d_h^R = 32$ for MLA models; and $R_k = R_v = 2$, $R_q = 6$ for TPA and TPA-KV only models. Other hyper-parameters are listed in Table 7.

Table 6: The architecture hyper-parameters and training devices of models. Abbreviations: BS. = Batch Size, GAS. = Gradient Accumulation Steps.

Model Size	#Param	Devices	Micro BS.	GAS.	#Layer	d_{model}
Small	124M	4× A100 GPUs	24	5	12	768
Medium	353M	8× A100 GPUs	20	3	24	1024
Large	772M	8× A100 GPUs	15	4	36	1280

Table 7: The architecture hyper-parameters for different models.

Model Size	Small	Medium	Large
h (MHA)	12	16	20
h (MQA)	23	31	39
h (GQA)	22	30	38
n_h (MLA)	12	23	34
h (TPA-KVonly)	22	29	37
h (TPA)	34	47	61
d_c (MLA)	256	512	512
d_c' (MLA)	512	1024	1024

B.2 Additional Experimental Results

We display the evaluation results for small-size (124M) models in Tables 8-9.

Table 8: The evaluation results of small models with different attention mechanisms pre-trained using FineWeb-Edu 100B dataset (0-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

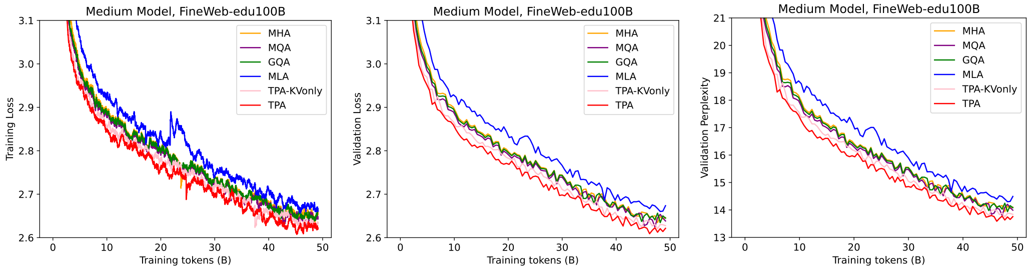
Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	50.63	26.96	59.39	36.18	32.00	64.96	**51.85**	23.40	70.30	46.19
MQA	49.62	25.34	55.72	35.94	31.40	64.85	51.30	23.37	68.70	45.14
GQA	48.70	25.68	56.15	35.58	31.40	64.91	51.62	23.12	68.20	45.04
MLA	49.66	26.45	**61.22**	33.94	32.40	62.73	50.43	23.29	71.50	45.74
TPA-KVonly	51.05	26.54	57.25	**36.77**	32.60	**65.02**	50.91	23.64	69.70	45.94
TPA	**51.26**	**27.39**	57.00	36.68	**32.80**	64.47	49.72	**24.61**	**72.00**	**46.21**

B.3 Ablation Studies on Learning Rates

We implement a set of parallel experiments for medium models with learning rate 6×10^{-4}, and the curves for training loss, validation loss and validation perplexity are displayed in Figure 5. We also show the performance of these models on the benchmarks described in Section 4 in Tables 10-11. The results show that TPA and TPA-KVonly models can also outperform other types of attention with different learning rates.

Table 9: The evaluation results of small models with different attention mechanisms on FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	**57.66**	**28.24**	**57.28**	36.43	29.60	64.09	51.14	**26.57**	**82.00**	**48.11**
MQA	53.79	26.35	44.95	34.18	28.80	62.79	52.01	25.91	78.10	45.21
GQA	55.01	25.94	55.72	35.68	**31.80**	**65.29**	51.93	25.27	77.80	47.16
MLA	52.78	26.19	57.25	33.19	29.60	63.98	50.43	24.90	76.00	46.04
TPA-KVonly	54.25	27.90	57.06	36.36	**31.80**	64.31	**53.59**	26.18	79.20	47.85
TPA	57.53	28.07	56.33	**36.49**	**31.80**	64.36	51.14	25.92	79.70	47.93



(a) Training Loss

(b) Validation Loss

(c) Validation Perplexity

Figure 5: The training loss, validation loss and validation perplexity of medium-size (353M) models with learning rate 6×10^{-4} and different attention mechanisms on the FineWeb-Edu 100B dataset.

Table 10: The evaluation results of medium models (learning rate=6×10^{-4}) with different attention mechanisms pre-trained using FineWeb-Edu 100B dataset (0-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	**59.51**	29.52	59.60	45.68	34.20	68.82	53.43	23.33	76.90	50.11
MQA	57.62	**31.91**	59.45	45.69	35.40	69.31	53.51	**26.47**	74.60	50.44
GQA	28.67	31.48	58.29	45.45	35.20	68.50	**54.46**	24.58	76.50	47.01
MLA	57.49	29.44	**59.97**	44.09	25.77	68.66	53.04	25.77	76.40	48.96
TPA-KVonly	58.01	30.12	58.01	45.95	35.60	69.10	53.12	25.39	75.10	50.04
TPA	58.38	31.57	59.39	**46.83**	**37.00**	**70.02**	54.06	25.52	**79.90**	**51.41**

Table 11: The evaluation results of medium models (learning rate 6×10^{-4}) with different attention mechanisms pre-trained using FineWeb-Edu 100B dataset (2-shot with lm-evaluation-harness). The best scores in each column are **bolded**. Abbreviations: HellaSw. = HellaSwag, W.G. = WinoGrande.

Method	ARC-E	ARC-C	BoolQ	HellaSw.	OBQA	PIQA	W.G.	MMLU	SciQ	Avg.
MHA	64.73	32.42	58.29	45.89	34.20	68.50	53.20	**25.86**	88.00	52.34
MQA	64.98	33.62	55.02	45.81	34.00	69.59	53.43	24.30	85.20	51.77
GQA	65.24	33.19	56.54	45.41	34.80	69.04	**55.72**	24.73	87.90	52.51
MLA	63.80	31.06	58.50	44.19	**35.40**	68.44	51.62	25.22	88.50	51.86
TPA-KVonly	64.69	32.34	**59.48**	46.23	**35.40**	**70.08**	54.06	25.64	86.30	52.69
TPA	**67.97**	**34.56**	57.22	**46.87**	34.60	69.91	52.01	25.07	**89.90**	**53.12**